NCAR Graphics 4.1 Reference Manual welcome

The NCAR Graphics 4.1 Reference Manual describes how the NCAR Graphics 4.1 software operates. The information provided here is written from the point of view of the software developers and is very useful to experienced programmers:

- HLU reference documents
- NCL reference documents

In all modules for the Reference Manual, the Roadmap and Contents navigation tools show Reference Manual contents only.

Reference Manual table of contents

Reference Manual Credits

Acknowledgments

Authors: Ethan Alpert, Dave Brown, Jeff Boote.

Editor: Brian Bevirt

Production: Jacque Marshall

Copyright

Copyright © 1987-1999 University Corporation for Atmospheric Research
The use of this software and documentation is governed by a License Agreement.

Trademarks

NCAR Graphics is a registered trademark of the University Corporation for Atmospheric Research.

All brand and product names are trademarks or registered trademarks of their respective holders. Reference to a company or product name does not imply approval or recommendation of that company or product to the exclusion of others.

Published by

National Center for Atmospheric Research, Scientific Computing Division, P.O. Box 3000, Boulder, CO 80307-3000. The National Center for Atmospheric Research is operated by the University Corporation for Atmospheric Research and is sponsored by the National Science Foundation. Any opinions, findings, and conclusions or recommendations expressed in this publication are those of the authors and do not necessarily reflect the views of the National Science Foundation.

Copyright

Copyright 1987-1999 University Corporation for Atmospheric Research
The use of this Software is governed by a License Agreement.

NCAR Graphics is a registered trademark of the University Corporation for Atmospheric Research.

NCAR Graphics environment variables

NCAR Graphics uses a set of environment variables to determine where each of its parts is installed, and to configure some of the runtime options.

You can execute the **ncargpath** program from the shell prompt to determine what values NCAR Graphics is going to use for these environment variables.

TMPDIR
: This environment variable indicates what directory NCARG should use for temporary files.

  Default: /tmp
  NOTE: This default is configurable at compilation time, so your system administrator may have changed it.

FONTCAP
: This environment variable indicates the default fontcap to use for ctrans.

  Default: pwritex database

GRAPHCAP
: This environment variable is used to select the graphcap for ctrans to use.

  Default: none
  If this environment variable is not set, then command line flags must be used.

NCARG_GKS_OUTPUT
: This environment variable is used to indicate the default name of the metafile the ncgm workstation outputs to.

  Default: gmeta

NCARG_USRRESFILE
: This environment variable is used to specify the user's HLU resource file. This file can be used to specify resources for all HLU programs.

  Default: ~/.hluresfile
  NOTE: This default is configurable at compilation time, so your system administrator may have changed it.

The rest of the environment variables indicate where different portions of NCAR Graphics are installed. In most cases, only the NCARG_ROOT environment variable needs to be specified. All the other variables will be determined from it.

NCARG_ROOT

This environment variable is used to indicate where the "root" of the NCARG install tree is. If things are installed properly, most of the other environment variables are determined relative to the value of this variable.

Default: /usr/local

NOTE: This default is configurable at compilation time, so your system administrator may have changed it.

NCARG_BIN

This environment variable is used to indicate the location of the "bin" directory in the NCARG install tree.

Default: $(NCARG_ROOT)/bin

NCARG_MAN

This environment variable is used to indicate the location of the "man" pages in the NCARG install tree.

Default: $(NCARG_ROOT)/man

NCARG_INCLUDE

This environment variable is used to indicate where the NCAR Graphics "include" files are installed.

Default: $(NCARG_ROOT)/include

NCARG_LIB

This environment variable is used to indicate where the NCAR Graphics libraries are installed.

Default: $(NCARG_ROOT)/lib

NCARG_NCARG

This environment variable is used to indicate the locations of the NCAR Graphics examples, databases, resource files, and so on.

Default: $(NCARG_LIB)/ncarg

NCARG_CONFIG

This environment variable is used to indicate where the NCAR Graphics config files are installed. (These config files are only installed if NCAR Graphics is installed from source, they are not needed for a binary installation.)

Default: $(NCARG_NCARG)/config

NCARG_DATABASE

This environment variable is used to indicate where the NCAR Graphics database files are installed. (Map databases etc.)

Default: $(NCARG_NCARG)/database

NCARG_DATA

This environment variable is used to indicate where the data files for the NCAR Graphics HLU and NCL examples are installed.

Default: $(NCARG_NCARG)/data

NCARG_FONTCAPS

This environment variable is used to indicate where the NCAR Graphics fontcaps are installed.

Default: $(NCARG_NCARG)/fontcaps

NCARG_GRAPHCAPS

This environment variable is used to indicate where the NCAR Graphics graphcaps are installed.

Default: $(NCARG_NCARG)/graphcaps

NCARG_HLUEX

This environment variable is used to indicate where the NCAR Graphics C and Fortran HLU examples are installed.

Default: $(NCARG_NCARG)/hluex

NCARG_NCLEX

This environment variable is used to indicate where the NCAR Graphics NCL examples are installed.

Default: $(NCARG_NCARG)/nclex

NCARG_RESFILES

This environment variable is used to indicate where the resource files for the NCAR Graphics NCL and HLU examples are installed.

Default: $(NCARG_NCARG)/resfiles

NCARG_EXAMPLES

This environment variable is used to indicate where the NCAR Graphics LLU examples are installed.

Default: $(NCARG_NCARG)/examples

NCARG_TESTS

This environment variable is used to indicate where the NCAR Graphics LLU test examples are installed.

Default: $(NCARG_NCARG)/tests

NCARG_TUTORIAL

This environment variable is used to indicate where the NCAR Graphics LLU tutorial examples are installed.

Default: $(NCARG_NCARG)/tutorial

NCARG_XAPP

This environment variable is used to indicate where the NCAR Graphics X application resource files are installed.

Default: $(NCARG_NCARG)/xapp

NCARG_SYSRESFILE

This environment variable is used to indicate where the NCAR Graphics HLU resource file is installed.

Default: $(NCARG_NCARG)/sysresfile

NCARG_SYSAPPRES

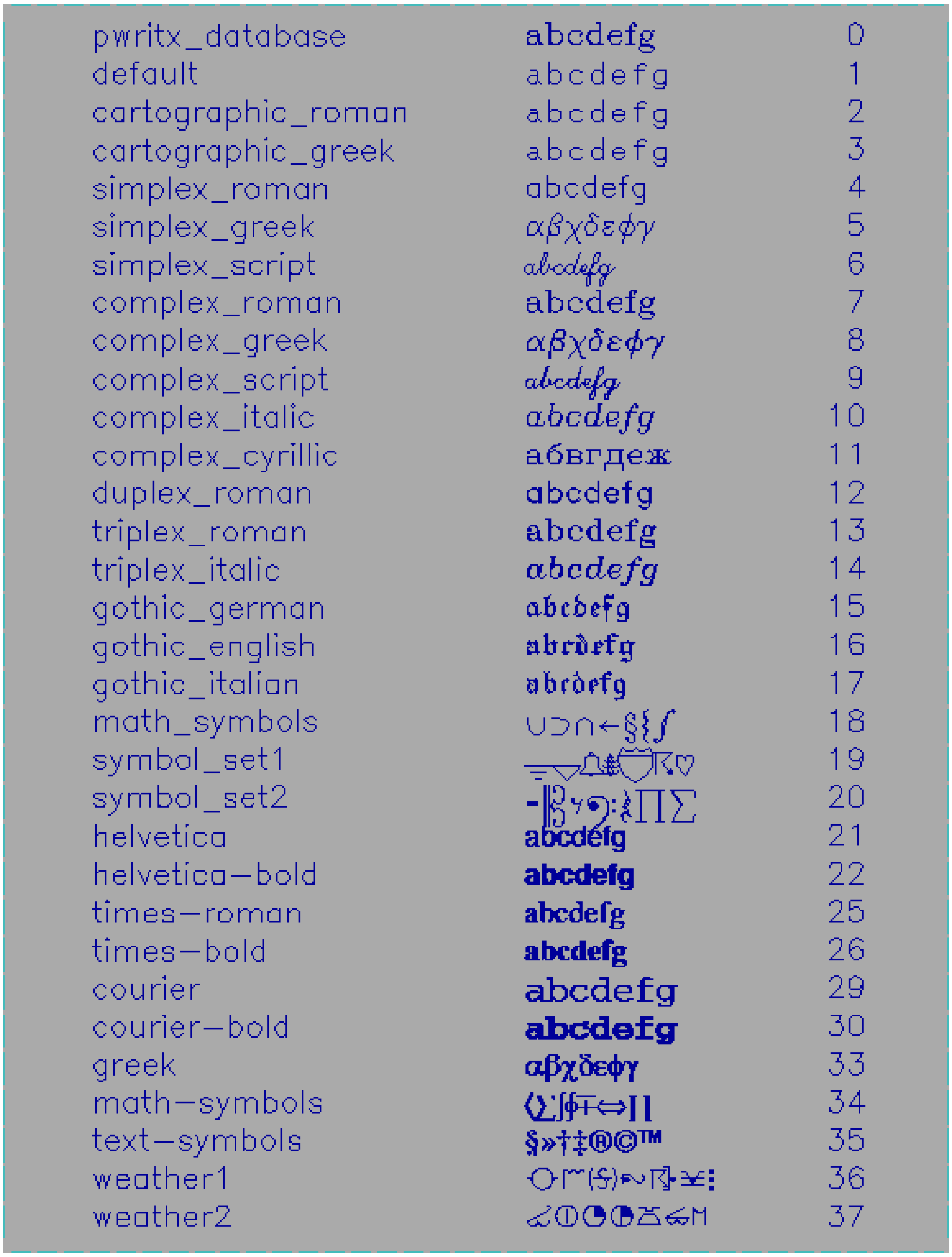
This environment variable is used to indicate where the NCAR Graphics HLU Application resource files are installed.

Default: $(NCARG_NCARG)/sysappres

Font table

This module lists the NCAR Graphics font tables. The first column lists the font name, the second column lists a few of the font's characters, and the third column lists the font index.

Click on one of the font names below to display the complete character set for that font.



Color tables

Pre-defined color tables

Several pre-defined colormaps are available in the NCAR Graphics package. You can change the colormap of your workstation by setting the wkColorMap resource to one of the following strings.

- "default" - This is the default colormap with 32 entries.
- "cyclic" - This colormap has 8 entries containing the primary colors and their complements.
- "gscyclic" - This is a grayscale version of the cyclic colormap with 8 entries.
- "gsltod" - This colormap contains 33 grayscale entries that range from light to dark.
- "gsdtol" - This colormap contains 33 grayscale entries that range from dark to light.
- "uniform" (2nd frame) - The uniform colormap contains 175 entries that cycle through 5 levels of red, 7 levels of green, and 5 levels of blue.
- "temp1" - The temp1 colormap contains 63 entries that are arranged in a continuous color table with "cool" colors at the bottom and "warm" colors at the top. The colors range from white to blue to green to red.
- "psgcap" (2nd frame)- This colormap contains 231 entries. It is based on the colormap that is used in the color PostScript graphcap for the NCAR Graphics ctrans utility.
- "example" - The example colormap contains 115 entries with hues of yellow, red, blue, and green.

For more information about setting color resources, see the "Color format specifications" section.

Color specification formats

Color specification formats

It is possible to use *named colors* and other color specification formats in place of color index values when setting various types of color resources. There are four different types of resources for setting single or arrays of color values: `NhlTColorIndex`, `NhlTColorIndexGenArray`, `NhlTColorMap`, and `NhlTColorDefinitionGenArray`. Setting color values for each of these types of resources is covered below.

For complete examples showing how to use named colors, see the *Quick Start Guide* examples cn05, cn14, cn17, xy07, and xy16.

- **`NhlTColorIndex`**

  The accepted formats for a color resource of type `NhlTColorIndex` are:

n	an integer color index
"(/*x*,*y*,*z*/)" (/*x*,*y*,*z*/)	an RGB triplet with values from 0.0 to 1.0 inclusive
"*color_name*"	a color name (from the last column in rgb.txt)
"*enumerated_name*"	one of the strings "transparent", "background", or "foreground"

For example, if you want to set the font color for the main title (`tiMainFontColor`), and color index 2 is red, you can set it by using any one of the following calls from an NCL script:

```
"tiMainFontColor" : 2
"tiMainFontColor" : "red"
"tiMainFontColor" : "(/1.,0.,0./)"
"tiMainFontColor" : (/1.,0.,0./)
```

In an HLU C program, you would use one of the following lines:

```
NhlRLSetInteger(srlist,NhlNtiMainFontColor,2);
NhlRLSetString(srlist,NhlNtiMainFontColor,"(/1.,0.,0./)");
NhlRLSetString(srlist,NhlNtiMainFontColor,"red");
```

and in a resource file you would use (note the lack of double quotes):

```
*tiMainFontColor : 2
*tiMainFontColor : red
*tiMainFontColor : (/1.,0.,0./)
```

- **`NhlTColorIndexGenArray`**

  The accepted format for a resource of type `NhlTColorIndexGenArray` is an array of `NhlTColorIndex` elements. You can mix different color specification types, but if you do this, each type must be enclosed in double quotes.

  For example, if you want to set the fill colors for various contour levels (resource `cnFillColors`), and your color map is defined as follows:

```
(/(/1.,1.,1./),(/0.,0.,0./),(/1.,0.,0./),(/0.,1.,0./),(/0.,0.,1./),\
  (/0.,1.,1./),(/1.,0.,1./),(/1.,1.,0./)/)
```

  (which translates to white, black, red, green, blue, cyan, magenta, and yellow) then either of the two NCL code settings will produce identical results:

```
"cnFillColors" : (/2,3,4,5,6,7,-1/)
"cnFillColors" : (/"(/1.,0.,0./)","(/0.,1.,0./)","blue","(/0.,1.,1./)",\
                   "(/1.,0.,1./)","yellow","transparent"/)
```

- **`NhlTColorMap`**

  The accepted formats for a resource of type `NhlTColorMap` are:

(/(/x1,y1,z1/),(/x2,y2,z2/),...,(/xn,yn,zn/)/)	a 2-dimensional (*n* x 3) array of RGB triplets with values from 0.0 to 1.0 inclusive
"(/*x,y,z*/)" "*color_name*"	a 1-dimensional array of any combination of these strings
"*color_map_name*"	a single string with a color map name (see the "Color tables" section for a list of the valid names)

Here's an NCL example of how to define a color map (resource `wkColorMap`) using a combination of the different formats described above:

```
wks = create "x11" xWorkstationClass defaultapp
  "wkColorMap" : (/"(/0.0,0.0,0.0/)","(/1.0,1.0,1.0/)","(/1.0,0.0,0.0/)",\
                   "(/0.7,0.1,1.0/)","(/0.2,1.0,0.1/)","(/1.0,1.0,0.2/)",\
                   "(/0.3,0.4,0.8/)","(/0.4,0.5,0.9/)","(/0.5,0.8,0.9/)",\
                   "HotPink","DeepSkyBlue","YellowGreen","Grey56"/)
end create
```

Note that you can mix color specification types, but you must enclose each one in double quotes when you mix them. The above color map could have also been specified with the following code segment:

```
wks = create "x11" xWorkstationClass defaultapp
  "wkColorMap" : (/"white","black","red","(/0.7,0.1,1.0/)",\
                   "(/0.2,1.0,0.1/)","(/1.0,1.0,0.2/)","(/0.3,0.4,0.8/)",\
                   "(/0.4,0.5,0.9/)","(/0.5,0.8,0.9/)","HotPink",\
                   "DeepSkyBlue","YellowGreen","Grey56"/)
end create
```

or with pure RGB values (no quotes are needed here because there are no named colors being referenced):

```
wks = create "x11" xWorkstationClass defaultapp
  "wkColorMap" : (/(/0.000,0.000,0.000/),(/1.000,1.000,1.000/),\
                   (/1.000,0.000,0.000/),(/0.700,0.100,1.000/),\
                   (/0.200,1.000,0.100/),(/1.000,1.000,0.200/),\
                   (/0.300,0.400,0.800/),(/0.400,0.500,0.900/),\
                   (/0.500,0.800,0.900/),(/1.000,0.412,0.706/),\
                   (/0.000,0.749,1.000/),(/0.604,0.804,0.196/),\
                   (/0.561,0.561,0.561/)/)
end create
```

The RGB values for "HotPink", "DeepSkyBlue", "YellowGreen", and "Grey56" were calculated from the file $NCARG_ROOT/lib/ncarg/database/rgb.txt, which contains integer RGB values for each named color. To convert these integer values to float values that range from 0.0 to 1.0, divide each one by 255.

- **`NhlTColorDefinitionGenArray`**

The accepted formats for a resource of type `NhlTColorDefinitionGenArray` are:

"(/*x*,*y*,*z*/)" (/*x*,*y*,*z*/)	an RGB triplet with values from 0.0 to 1.0 inclusive
"*color_name*"	a color name (from the last column in rgb.txt)

For example, to set the resources `wkBackgroundColor` and `wkForegroundColor` to white and black respectively, you might use one of the following types of NCL code snippets:

```
wks = create "x11" xWorkstationClass defaultapp
  "wkBackgroundColor" : "white"
  "wkForegroundColor" : "black"
end create
```

or

```
wks = create "x11" xWorkstationClass defaultapp
  "wkBackgroundColor" : (/1.,1.,1./)
  "wkForegroundColor" : (/0.,0.,0./)
end create
```

Important notes about color specification formats

- When you access a color by anything other than an integer index value, the current color map is searched for the closest match to that color. This means that if you access a color that doesn't have a close match in your color map, you may not get the color you are expecting.

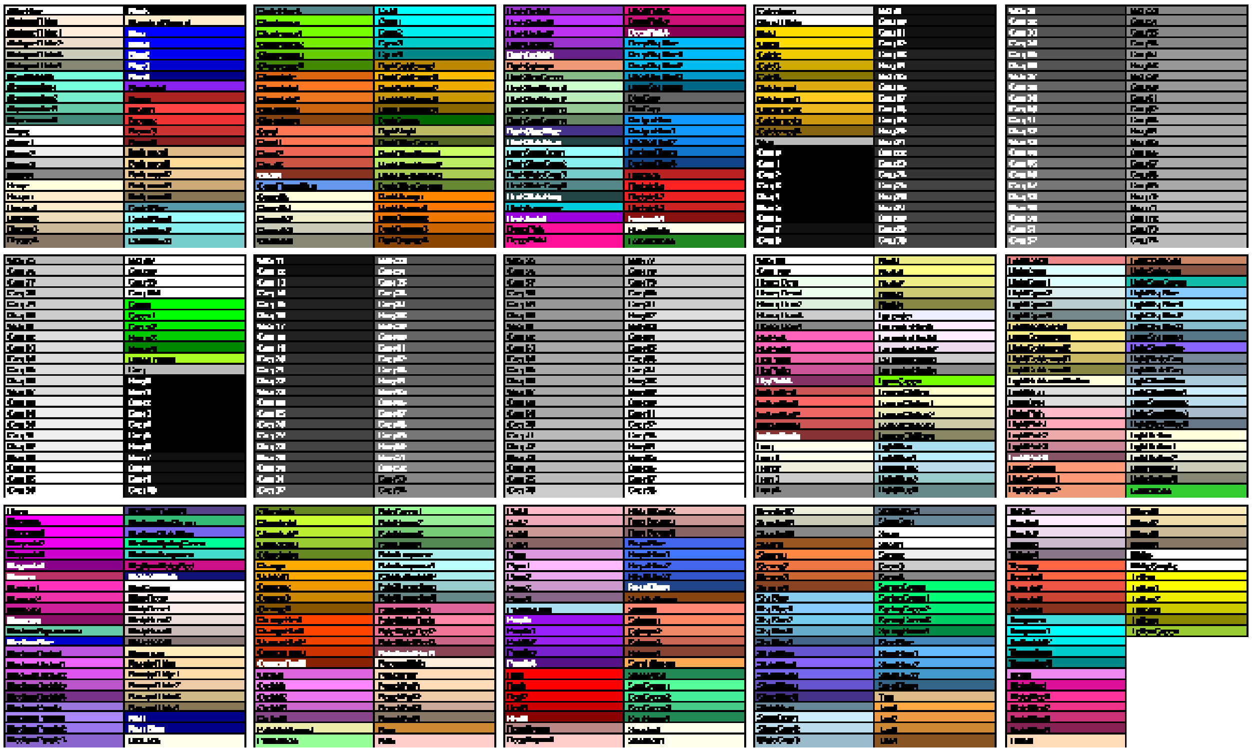
- If you create two workstations each with a different colormap, and then set up one or more plots using named colors, then when you call the procedure for changing the workstation and redraw the plots, the colors may not come out as you expected in the second workstation. For an example, try running the following NCL script:

```
begin
  wks1 = create "wks1" xWorkstationClass defaultapp
    "wkColorMap" : (/"white","black","blue","red","green"/)
  end create

  wks2 = create "wks2" xWorkstationClass defaultapp
    "wkColorMap" : (/"white","black","red","green","blue"/)
  end create

  text = create "text" textItemClass wks1
    "txString"    : "hello, world"
    "txFontColor" : "blue"
  end create

  draw(text)
  frame(wks1)
  NhlChangeWorkstation(text,wks2)
  draw(text)
  frame(wks2)
end
```

In the second workstation, the text should be red instead of blue, because in the second color map, red is occupying the same location in the color map that blue occupied for the first color map. To get around this problem, you need to make sure that named colors occupy the same locations in the color maps, or else you have to call **setvalues** on *text* and set `txFontColor` to "blue" again.

Tables of named colors

To see the valid 650 named colors, see the ASCII file $NCARG_ROOT/lib/ncarg/database/rgb.txt or click on any one of the 15 tables below to see the color names and their associated colors:

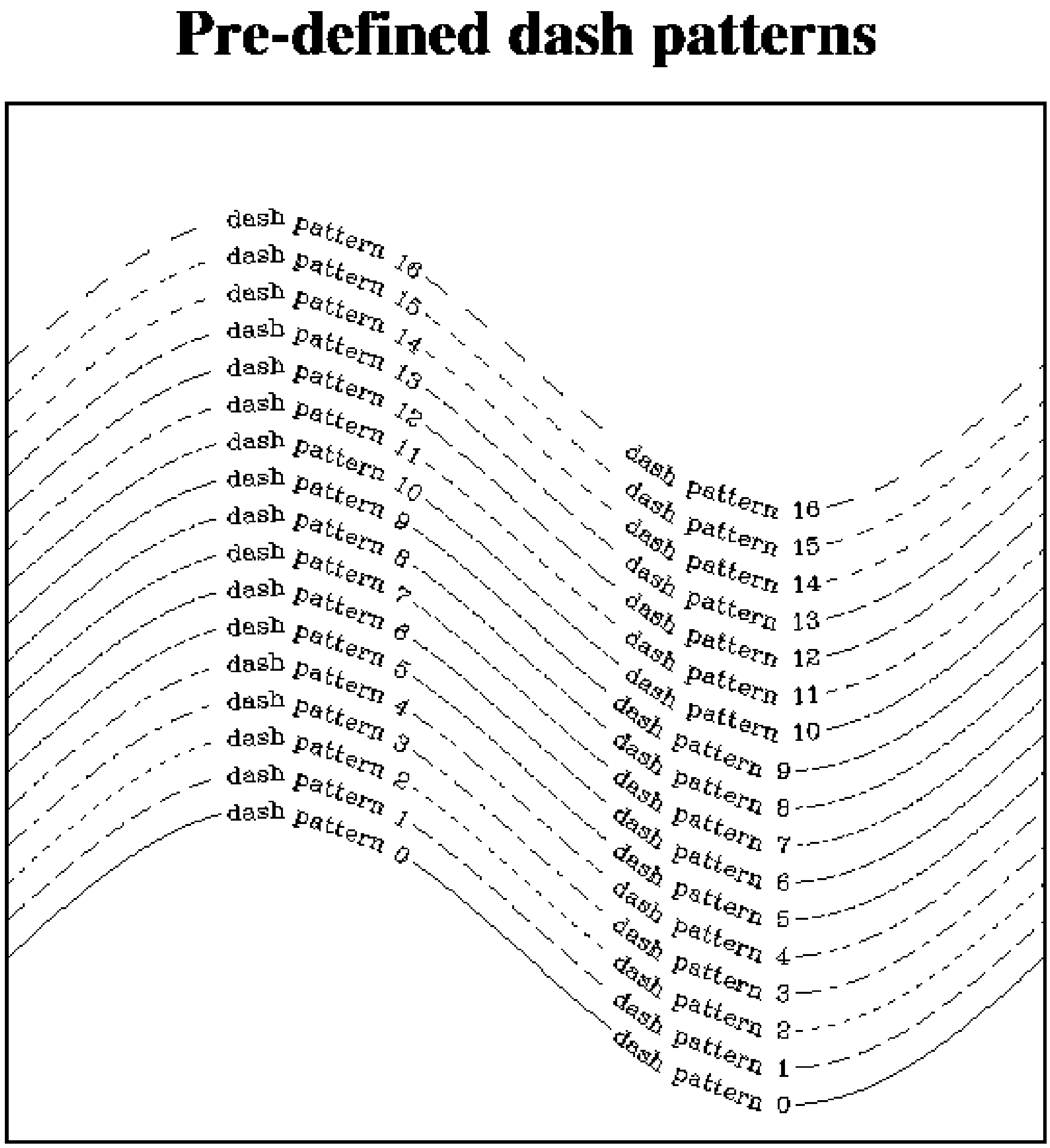


MIT copyright


```
***************************************************************************
        Copyright    Massachusetts Institute of Technology    1985

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

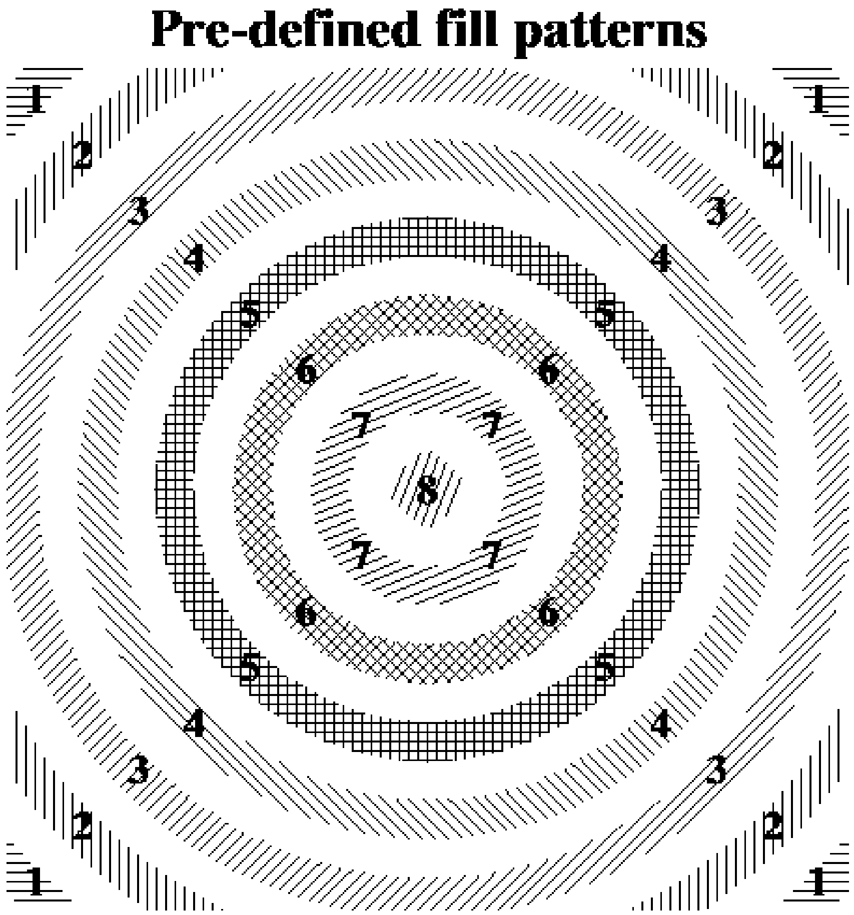
Dash patterns

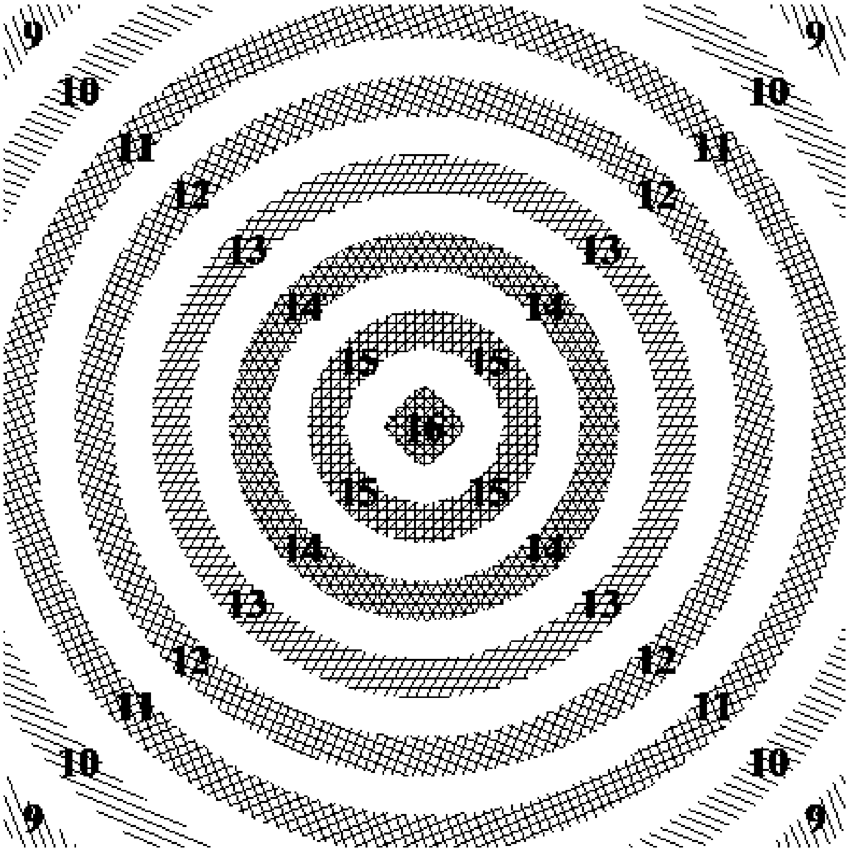
There are 16 pre-defined dash patterns available in the NCAR Graphics package. You can change the dash pattern in a number of resources by using an integer from 1 to 16.



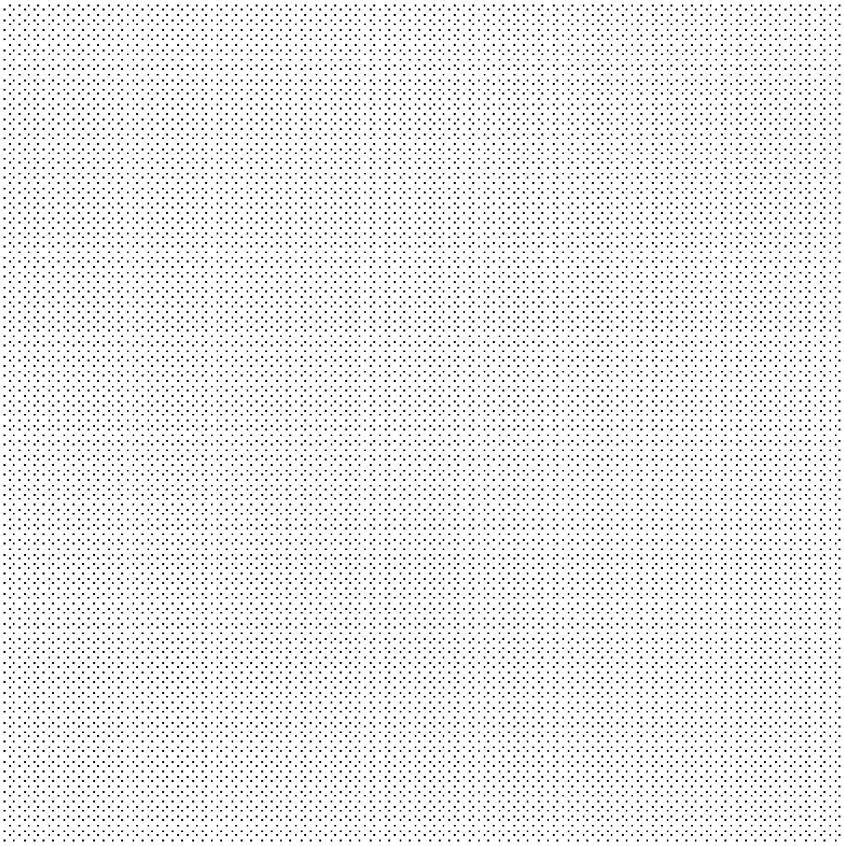
Fill (hatching) patterns

There are 17 pre-defined fill (hatching) patterns available in the NCAR Graphics package. You can change the fill pattern in a number of resources by using an integer from 1 to 17.



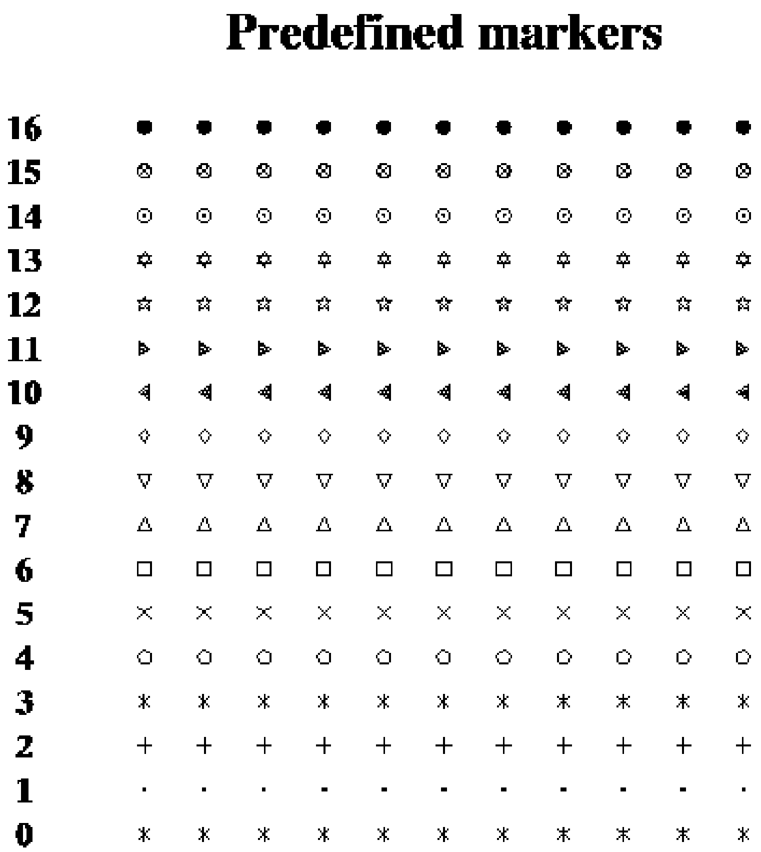


Fill pattern #17 is a special stippling pattern:



Markers

There are 17 predefined markers available in the NCAR Graphics package. You can change the marker type in a number of resources by using an integer from 0 to 16.



HLU library reference manual

The HLU library consists of classes, functions, resources, and types. These groups provide access to detailed information about each of the individual entries:

- HLU classes - alphabetically
  - HLU class reference page layout
- HLU API - alphabetically
  - HLU API reference page layout
- HLU resources - alphabetically
- HLU types - alphabetically
- HLU enumeration values - alphabetically

HLU classes - alphabetically

AnnoManager
: The **AnnoManager** object allows an arbitrary user-created **View** object to be managed as an annotation of a plot object.

App
: The **App** object is used to manage resource databases.

Base
: The **Base** object provides the base functionality used by all objects in the HLU library.

ContourPlot
: The **ContourPlot** object draws line or filled contours with labels and annotations.

CoordArrTable
: The **CoordArrTable** object provides a way to describe X/Y coordinate data to the HLU library.

CoordArrays
: The **CoordArrays** object provides a way to describe X/Y coordinate data to the HLU library.

DataComm
: The **DataComm** object provides the functionality for its Subclasses to communicate with **DataItem** objects.

DataItem
: The **DataItem** objects provide a way to describe data to the HLU library.

DataSpec
: The **DataSpec** objects provide a way to describe data- specific plot attributes.

Error
: The **Error** object provides configuration options for the error reporting module of the HLU library.

GraphicStyle
: The **GraphicStyle** class provides grouped attributes for drawing primitives.

IrregularPlot
: The **IrregularPlot** object draws immediate mode lines and may be used to map overlay plots into an irregular rectangular coordinate space.

IrregularTransformation
: The **IrregularTransformation** object manages forward and reverse transformations in an irregular rectangular coordinate space.

LabelBar
: The **LabelBar** object draws an annotation for fill styles with an optional title and/or border.

Legend
: The **Legend** object draws an annotation for line and/or marker styles with an optional title and/or border.

LogLinPlot
: The **LogLinPlot** object draws immediate mode lines and may be used to map overlay plots into a log or linear coordinate space.

LogLinTransformation
: The **LogLinTransformation** object manages forward and reverse transformations in log or linear coordinate space.

MapPlot
: The **MapPlot** object draws outlined and/or filled maps of arbitrary regions of the globe and may be used to map overlay plots into a number of standard projections.

MapTransformation
: The **MapTransformation** object manages forward and reverse transformations in the coordinate

space defined by various map projections.

NcgmWorkstation
: The **NcgmWorkstation** manages communication with the NCAR Graphcs CGM Workstation output device.

Obj
: The **Obj** class is a simplified version of the **Base** class.

PSWorkstation
: The **PSWorkstation** object manages communication with the NCAR Graphics PS Workstation output device.

PlotManager
: The **PlotManager** allows a plot object to manage overlays and annotations.

ScalarField
: The **ScalarField** object supplies two dimensional scalar field data.

StreamlinePlot
: The **StreamlinePlot** class represents a vector field by drawing streamlines.

Style
: The **Style** class provides its subclasses with the ability to group attributes.

TextItem
: The **TextItem** object draws arbitrary text.

TickMark
: The **TickMark** object draws tickmarks along any of the sides of a plot.

Title
: The **Title** object draws main, X and Y axis titles for a parent plot or overlay object.

Transformation
: The **Transformation** classes manage forward and backward transformations for the **Transform** classes.

Transform
: The **Transform** class provides its sub-classes with the ability to handle coordinate transformations.

VectorField
: The **VectorField** class supplies two-dimensional vector field data.

VectorPlot
: The **VectorPlot** class represents a vector field by drawing arrows at points of a grid.

View
: The **View** class provides its sub-classes with an ability to be drawn.

Workspace
: The **Workspace** object manages large blocks of memory on behalf a number of HLU library objects.

Workstation
: A **Workstation** object manages an instance of an output device.

XWorkstation
: The **XWorkstation** manages communication with the NCAR Graphcs X Workstation input/output device.

XyDataSpec
: The **XyDataSpec** object is used to specify data-specific plot attributes to the **XyPlot** object.

XyPlot
: The **XyPlot** object draws curves made up of X/Y coordinate pairs with tick marks, titles, and a

legend.

HLU API - alphabetically

NHLPERROR
The Fortran name of this function is **NhlFPError**.

Error handling functions for the HLU library.

NhlAddAnnotation
The Fortran name of this function is **NhlFAddAnnotation**.

This function adds an arbitrary **View** object to a plot object as an external annotation, and returns an **AnnoManager** object to manage it.

NhlAddData
The Fortran name of this function is **NhlFAddData**.

This function is used to associate a **DataItem** object with a data resource in a **DataComm** object.

NhlAddOverlay
The Fortran name of this function is **NhlFAddOverlay**.

This function adds an overlay plot member to a base plot.

NhlAppGetDefaultParentId
The Fortran name of this function is **NhlFAppGetDefaultParentId**.

This function retrieves the Id of the current *default parent* **App** object.

NhlChangeWorkstation
The Fortran name of this function is **NhlFChangeWorkstation**.

This function is used to change the Workstation parent of any graphical HLU object. It is provided as an access function to objects that are Base class.

NhlClassName
The Fortran name of this function is **NhlFClassName**.

Returns the name of the class of an object.

NhlClearWorkstation
The Fortran name of this function is **NhlFClearWorkstation**.

This function clears the drawing area of an instance of a **Workstation** object.

NhlClose
The Fortran name of this function is **NhlFClose**.

HLU library clean-up function.

NhlCreate
The Fortran name of this function is **NhlFCreate**.

This function is used to create an instance of any HLU object. It is provided as an access function to objects that are Base class.

NhlDataPolygon

The Fortran name of this function is **NhlFDataPolygon**.

Given arrays containing the X and Y coordinates of points in data coordinate space, this function outputs an immediate-mode polygon connecting each of the points.

NhlDataPolyline

The Fortran name of this function is **NhlFDataPolyline**.

Given arrays containing the X and Y coordinates of points in data coordinate space, this function outputs an immediate-mode polyline connecting each of the points.

NhlDataPolymarker

The Fortran name of this function is **NhlFDataPolymarker**.

Given arrays containing the X and Y coordinates of points in data coordinate space, this function outputs an immediate-mode marker at each of the points.

NhlDataToNDC

The Fortran name of this function is **NhlFDataToNDC**.

Given arrays containing the X and Y coordinates of points in data coordinate space, this function returns arrays containing the X and Y coordinates of the same points in NDC space.

NhlDestroy

The Fortran name of this function is **NhlFDestroy**.

This function is used to destroy an HLU object.

NhlDraw

The Fortran name of this function is **NhlFDraw**.

This function is used to make an HLU graphical object draw.

NhlErrAddTable

No corresponding Fortran function.

Error handling functions for the HLU library.

NhlErrClearMsgs

The Fortran name of this function is **NhlFErrCleanMsgs**.

Error handling functions for the HLU library.

NhlErrFPrintMsg

The Fortran name of this function is **NhlFErrFPrintMsg**.

Error handling functions for the HLU library.

NhlErrGetId

The Fortran name of this function is **NhlFErrGetId**.

Error handling functions for the HLU library.

NhlErrGetMsg

The Fortran name of this function is **NhlFErrGetMsg**.

Error handling functions for the HLU library.

NhlErrNumMsgs

The Fortran name of this function is **NhlFErrNumMsgs**.

Error handling functions for the HLU library.

NhlErrSPrintMsg

The Fortran name of this function is **NhlFErrSprintMsg**.

Error handling functions for the HLU library.

NhlFrame

The Fortran name of this function is **NhlFFrame**.

This function first updates and then clears the drawing area of an instance of a **Workstation** object.

NhlFree

No corresponding Fortran function.

Memory allocation functions.

NhlFreeColor

The Fortran name of this function is **NhlFFreeColor**.

This function removes a color, specified by its HLU index, from the **Workstation** color map.

NhlGetBB

The Fortran name of this function is **NhlFGetBB**.

This function returns the coordinates of the left, right, top, and bottom edges of a **View** class object's bounding box in NDC units.

NhlGetGksCi

The Fortran name of this function is **NhlFGetGksCi**.

This function returns the low-level GKS color index from an HLU color index and an instance of a **Workstation** object.

NhlGetValues

The Fortran name of this function is **NhlFGetValues**.

This function is used to retrieve the resource values of an instance of any HLU object. It is provided as an access function to objects that are Base class.

NhlGetWorkspaceObjectId

The Fortran name of this function is **NhlFGetWorkspaceObjectId**.

This function retrieves the **Workspace** objects id.

NhlInitialize

The Fortran names of this function is **NhlFInitialize**.

HLU library initialization functions.

NhlIsAllocatedColor

The Fortran name of this function is **NhlFIsAllocatedColor**.

This boolean function returns True if a color has been assigned to a particular HLU color index, and False otherwise.

NhlIsApp

The Fortran name of this function is **NhlFIsApp**.

This boolean function returns True if the given id identifies an **App** class object, and False otherwise.

NhlIsDataComm

The Fortran name of this function is **NhlFIsDataComm**.

This boolean function returns True if the given id identifies a **DataComm** class object, and False otherwise.

NhlIsDataItem

The Fortran name of this function is **NhlFIsDataItem**.

This boolean function returns True if the given id identifies a **DataItem** class object, and False otherwise.

NhlIsDataSpec

The Fortran name of this function is **NhlFIsDataSpec**.

This boolean function returns True if the given id identifies a **DataSpec** class object, and False otherwise.

NhlIsTransform

The Fortran name of this function is **NhlFIsTransform**.

This boolean function returns True if the given id identifies a **Transform** class object, and False otherwise.

NhlIsView

The Fortran name of this function is **NhlFIsView**.

This boolean function returns True if the given id identifies a **View** class object, and False otherwise.

NhlIsWorkstation

The Fortran name of this function is **NhlFIsWorkstation**.

This boolean function returns True if the given id identifies a **Workstation** class object, and False otherwise.

NhlMalloc

No corresponding Fortran function.

Memory allocation functions.

NhlNDCPolygon

The Fortran name of this function is **NhlFNDCPolygon**.

Given arrays containing the X and Y coordinates of points in NDC space, this function outputs an immediate-mode polygon connecting each of the points.

NhlNDCPolyline

The Fortran name of this function is **NhlFNDCPolyline**.

Given arrays containing the X and Y coordinates of points in NDC space, this function outputs an immediate-mode polyline connecting each of the points.

NhlNDCPolymarker

The Fortran name of this function is **NhlFNDCPolymarker**.

Given arrays containing the X and Y coordinates of points in NDC space, this function outputs an immediate-mode marker at each of the points.

NhlNDCToData

The Fortran name of this function is **NhlFNDCToData**.

Given arrays containing the X and Y coordinates of points in NDC space, this function returns arrays containing the X and Y coordinates of the same points in data coordinate space.

NhlName

The Fortran name of this function is **NhlFName**.

This function returns the "name" of an object.

NhlNewColor

The Fortran name of this function is **NhlFNewColor**.

This function adds a color specified as an RGB triple to the colormap of a **Workstation** instance and returns its HLU color index.

NhlNumber

No corresponding Fortran function.

This cpp macro determines the number of elements in a fixed-size array.

NhlOpen

The Fortran name of this function is **NhlFOpen**.

HLU library initialization functions.

NhlPError

The Fortran name of this function is **NhlFPError**.

Error handling functions for the HLU library.

NhlRLClear

The Fortran name of this function is **NhlFRLClear**.

ResList support function.

NhlRLCreate

The Fortran name of this function is **NhlFRLCreate**.

ResList support functions.

NhlRLDestroy

The Fortran name of this function is **NhlFRLDestroy**.

ResList support functions.

NhlRLGet

No corresponding Fortran function.

Used to add the given *name* and *val_addr* to the *ResList*.
NhlRLGetArray
No corresponding Fortran function.

Used to add the given *name* and *val_addr* to the *ResList*.
NhlRLGetDouble
The Fortran name of this function is **NhlFRLGetDouble**.

Used to add the given *name* and *val_addr* to the *ResList*.
NhlRLGetDoubleArray
The Fortran name of this function is **NhlFRLGetDoubleArray**.

Used to add the given *name* and *val_addr* to the *ResList*.
NhlRLGetFloat
The Fortran name of this function is **NhlFRLGetFloat**.

Used to add the given *name* and *val_addr* to the *ResList*.
NhlRLGetFloatArray
The Fortran name of this function is **NhlFRLGetFloatArray**.

Used to add the given *name* and *val_addr* to the *ResList*.
NhlRLGetInteger
The Fortran name of this function is **NhlFRLGetInteger**.

Used to add the given *name* and *val_addr* to the *ResList*.
NhlRLGetIntegerArray
The Fortran name of this function is **NhlFRLGetIntegerArray**.

Used to add the given *name* and *val_addr* to the *ResList*.
NhlRLGetMDArray
No corresponding Fortran function.

Used to add the given *name* and *val_addr* to the *ResList*.
NhlRLGetMDDoubleArray
The Fortran name of this function is **NhlFRLGetMDDoubleArray**.

Used to add the given *name* and *val_addr* to the *ResList*.
NhlRLGetMDFloatArray
The Fortran name of this function is **NhlFRLGetMDFloatArray**.

Used to add the given *name* and *val_addr* to the *ResList*.
NhlRLGetMDIntegerArray
The Fortran name of this function is **NhlFRLGetMDIntegerArray**.

Used to add the given *name* and *val_addr* to the *ResList*.
NhlRLGetString

The Fortran name of this function is **NhlFRLGetString**.

Used to add the given *name* and *val_addr* to the *ResList*.

NhlRLGetStringArray

The Fortran name of this function is **NhlFRLGetStringArray**.

Used to add the given *name* and *val_addr* to the *ResList*.

NhlRLIsSet

The Fortran name of this function is **NhlFRLIsSet**.

ResList support functions.

NhlRLSet

No corresponding Fortran function.

Used to add the given *name* and *value* to the *ResList*.

NhlRLSetArray

No corresponding Fortran function.

Used to add the given *name* and *value* array to the *ResList*.

NhlRLSetDouble

The Fortran name of this function is **NhlFRLSetDouble**.

Used to add the given *name* and *value* to the *ResList*.

NhlRLSetDoubleArray

The Fortran name of this function is **NhlFRLSetDoubleArray**.

Used to add the given *name* and *value* array to the *ResList*.

NhlRLSetFloat

The Fortran name of this function is **NhlFRLSetFloat**.

Used to add the given *name* and *value* to the *ResList*.

NhlRLSetFloatArray

The Fortran name of this function is **NhlFRLSetFloatArray**.

Used to add the given *name* and *value* array to the *ResList*.

NhlRLSetInteger

The Fortran name of this function is **NhlFRLSetInteger**.

Used to add the given *name* and *value* to the *ResList*.

NhlRLSetIntegerArray

The Fortran name of this function is **NhlFRLSetIntegerArray**.

Used to add the given *name* and *value* array to the *ResList*.

NhlRLSetMDArray

No corresponding Fortran function.

Used to add the given *name* and *value* array to the *ResList*.

NhlRLSetMDDoubleArray

The Fortran name of this function is **NhlFRLSetMDDoubleArray**.

Used to add the given *name* and *value* array to the *ResList*.

NhlRLSetMDFloatArray

The Fortran name of this function is **NhlFRLSetMDFloatArray**.

Used to add the given *name* and *value* array to the *ResList*.

NhlRLSetMDIntegerArray

The Fortran name of this function is **NhlFRLSetMDIntegerArray**.

Used to add the given *name* and *value* array to the *ResList*.

NhlRLSetString

The Fortran name of this function is **NhlFRLSetString**.

Used to add the given *name* and *value* to the *ResList*.

NhlRLSetStringArray

The Fortran name of this function is **NhlFRLSetStringArray**.

Used to add the given *name* and *value* array to the *ResList*.

NhlRLUnSet

The Fortran name of this function is **NhlFRLUnSet**.

ResList support functions.

NhlRealloc

No corresponding Fortran function.

Memory allocation functions.

NhlRemoveAnnotation

The Fortran name of this function is **NhlFRemoveAnnotation**.

This function removes an arbitrary **View** object annotation from a Plot Object, destroying its **AnnoManager** object.

NhlRemoveData

The Fortran name of this function is **NhlFRemoveData**.

This function is used to dissociate a **DataItem** object from a data resource in a **DataComm** object.

NhlRemoveOverlay

The Fortran name of this function is **NhlFRemoveOverlay**.

This function removes an overlay plot member from a base plot.

NhlSetColor

The Fortran name of this function is **NhlFSetColor**.

This function sets the RGB color value for a specified HLU color index.

NhlSetValues

The Fortran name of this function is **NhlFSetValues**.

This function is used to change the resource values of an instance of any HLU object. It is provided as an access function to objects that are Base class.

NhlUpdateData

The Fortran name of this function is **NhlFUpdateData**.

This function is used to force a **DataComm** object to update its internal state to any changes that have occurred in associated **DataItems**.

NhlUpdateWorkstation

The Fortran name of this function is **NhlFUpdateWorkstation**.

This function updates the drawing area of an instance of a **Workstation** object.

AnnoManager class

The **AnnoManager** class allows an arbitrary user-created **View** object to be managed as an annotation of a plot object.

Synopsis

```
Header File:             ncarg/hlu/AnnoManager.h
Class Name:              annoManagerClass
Class Pointer:           <Not Referenceable>
Fortran Class Function:  <Not Referenceable>
Super-Class:             Obj
Composite-Classes:       <None>
```

Resources

Local resources

AnnoManager Resource Set

NAME	CLASS	TYPE	DEFAULT	ACCESS
amOn	AmOn	NhlTBoolean	True	RCSG
amViewId	AmViewId	NhlTObjId	<dynamic>	CG
amResizeNotify	AmResizeNotify	NhlTBoolean	False	RCSG
amTrackData	AmTrackData	NhlTBoolean	False	RCSG

```
|--------------------------------------------------------------|
| amZone                        NhlTInteger           RCSG     |
|       AmZone                          0                      |
|--------------------------------------------------------------|
| amSide                        NhlTPosition          RCSG     |
|       AmSide                          Bottom                 |
|--------------------------------------------------------------|
| amOrthogonalPosF              NhlTFloat             RCSG     |
|       AmOrthogonalPosF                0.0                    |
|--------------------------------------------------------------|
| amParallelPosF                NhlTFloat             RCSG     |
|       AmParallelPosF                  0.0                    |
|--------------------------------------------------------------|
| amJust                        NhlTJustification     RCSG     |
|       AmJust                          CenterCenter           |
|--------------------------------------------------------------|
| amDataXF                      NhlTFloat             RCSG     |
|       AmDataXF                        0.0                    |
|--------------------------------------------------------------|
| amDataYF                      NhlTFloat             RCSG     |
|       AmDataYF                        0.0                    |
+--------------------------------------------------------------+
```

Composite resources

The **AnnoManager** object class has no composite class resources.

Superclass resources

There are no resources defined by the superclass of the **AnnoManager** object class.

Description

AnnoManager objects allow you to treat arbitrary user-created **View** objects as annotations of a plot object. The resources provided by the **AnnoManager** allow these external annotations to be manipulated in a very similar manner to the intrinsic annotations provided by the **PlotManager**, or the embedded annotations supplied by certain **Transform** subclasses.

You may turn any **View** object, from a simple **TextItem** to a plot object with plot members of its own, into an annotation. As a user, you do not create **AnnoManager** objects directly. You only need to create the views you wish to have managed as annotations. When you inform a plot object about these views, it will create a separate **AnnoManager** object to manage each of them.

When you draw a plot object, any of its external annotations whose **AnnoManager** resource *amOn* is set True are automatically drawn as well. Using the resources of the **AnnoManager** object, you set the location and usually the size of the **View** annotation relative to the viewport or the data coordinate space of its base plot. Then whenever the base plot changes size and/or location the annotation adjusts itself appropriately.

Adding annotations to a plot

There are two ways to add **View** objects as annotations to a plot object. You can make a group of **View** objects into annotations by setting the **PlotManager** resource *pmAnnoViews* with an array containing the ids of the **View** objects. Or you can add a single view to the existing set of annotations by calling the function **NhlAddAnnotation**. The view must belong to the same **Workstation** as the plot object to which it is added.

NhlAddAnnotation returns the id of a newly created **AnnoManager** whose resources you can set to control the view as an annotation. Alternatively, you can retrieve the ids of all current **AnnoManager** objects at once by getting the value of the array resource *pmAnnoManagers*. Each element of *pmAnnoManagers* contains the id of the **AnnoManager** object used to manage the view specified by the corresponding element of *pmAnnoViews*. Finally, you can also determine the **AnnoManager** id currently associated with any particular view by getting the value of the read-only **View** resource *vpAnnoManagerId*. If the view is not currently functioning as an annotation, the value of this resource will be `NullObjId` (0).

Setting *pmAnnoViews* removes any previously established annotations, although if a **View** object id in the array to be set matches an id in the existing array, the **PlotManager** does not destroy the corresponding **AnnoManager** object. It may, however, rearrange its position in the *pmAnnoManagers* array, based on the order of the new *pmAnnoViews* array. In contrast, when you call **NhlAddAnnotation** the ids of the **View** and its new **AnnoManager** are simply appended to the ends of the *pmAnnoViews* and *pmAnnoManagers* arrays.

You can add an annotation to a plot object even if the plot object is currently an overlay or an annotation itself. If the plot object is an overlay, the overlay's base plot assumes responsibility for its for its annotations. This means that the annotations belonging to the overlay are positioned with respect to the base plot just as if they belonged to the base plot itself. However, if the plot object is itself an annotation it acts as a subordinate base plot and manages any annotations added to it with respect to its own viewport.

Removing annotations from a plot

You may remove an annotation from a plot either by setting *pmAnnoViews* with a different set of **View** object ids, or one at a time by calling the function **NhlRemoveAnnotation**. When an annotation is removed, the plot object deletes the **View** object id from *pmAnnoViews* and the **AnnoManager** object id from *pmAnnoManagers*, and destroys the **AnnoManager** object. However, note that you are responsible for destroying the **View** object when you are finished with it. If you destroy a **View** object that is currently functioning as an annotation, it will remove itself from the plot object to which it belongs before it goes away.

Positioning annotations

There are two distinct ways to position annotation items: in NDC space or in data coordinate space. When the resource *amTrackData* is set False, the base plot sets the position in NDC space following the PlotManager Location Control Model. In this case, the base plot uses the resources *amZone*, *amSide*, *amParallelPosF*, *amOthogonalPosF* and *amJust*, in conjunction with its knowledge of its viewport and the location and size of all the other associated annotations to determine the location of the annotation. If *amTrackData* is True, the base plot uses the resources *amDataXF*, *amDataYF*, and *amJust*, along with its knowledge of the current data transformation in order to set the location. If the resource *amResizeNotify* is set True, the base plot also adjusts the view size of the annotation proportionally to changes in the size of its viewport.

Restrictions on use of View annotations

While a **View** object is functioning as an external annotation, you cannot draw it directly. It can only be drawn as a consequence of drawing the primary base plot of which it is a plot member.

Likewise, you cannot change the workstation of an external annotation directly. However, when you change the workstation of the primary base plot, the annotation's workstation changes along with it.

The annotation can belong only to one plot object. Attempts to call either **NhlAddAnnotation** or **NhlAddOverlay** using the id of any **View** object currently functioning as an external annotation will result in an error.

Support functions

The support functions used to manipulate **AnnoManager** objects are defined by the **Transform** object.

Status

See also

- NhlSetValues function
- NhlGetValues function
- NhlAddAnnotation function
- NhlRemoveAnnotation function
- PlotManager object

Copyright

Copyright 1987-1999 University Corporation for Atmospheric Research
The use of this Software is governed by a License Agreement.

NCAR Graphics is a registered trademark of the University Corporation for Atmospheric Research.

App class

The **App** class is used to manage resource databases.

Synopsis

```
Header file:             ncarg/hlu/App.h
Class name:              appClass
Class pointer:           NhlappClass
Fortran class function:  NHLFAPPCLASS
Superclass:              Base
Composite classes:       <None>
```

Note: If the hlu library is initialized with the **NhlOpen** function, then the default **AppClass** object will be named **hlu**.

Resources

Local resources

App Resource Set

NAME	CLASS	TYPE	DEFAULT	ACCESS
appUsrDir	AppUsrDir	NhlTString	./	RCG*
appSysDir	AppSysDir	NhlTString	<Dynamic>	RCG*
appFileSuffix	AppFileSuffix	NhlTString	.res	RCG*
appDefaultParent	AppDefaultParent	NhlTBoolean	<Dynamic>	RCSG
appResources	AppResources	NhlTStringGenArray	NULL	RCG

* These resources are settable from the Resource Database, however; they cannot be specified in the Application-specific resource files.

Superclass resources

There are no Superclass resources.

Description

All HLU applications must create at least one **App** class object. This is because each object in the HLU library needs to have an **App** object for its most top-level parent. It keeps track of the Resource Database for all objects that are created as decendents of it. The first **App** object that is created is somewhat special. It becomes the parent of the **Error** and **Workspace** object. Therefore, **Error** and **Workspace** resources can only be set in resource files used by the first **App** object in the application.

Initializing the HLU library

All HLU applications must initialize the HLU library in one of two ways:

First, the easiest method is to simply call the **NhlOpen** convienience function. If this method is used, it is not possible to interact with the **App** object directly. The Resource Database managed by this *internal* **App** object is made up from the $(NCARG_USRRESFILE) and the $(NCARG_SYSRESFILE), with the user's file taking precedence over the system file. This *internal* **App** object is named **hlu**. This is important to know for specifing resources in these two resource files if this **App** object is used as the parent for any other objects.

The other way of initializing the library is to call the **NhlInitialize** function, and then explicitly create an **App** object. The only reason to select this method is if the application programmer wants to provide an Application specific resource file. This would be a resource file, in the same format as the $(NCARG_SYSRESFILE), but it would only be used to load the Resource Database of the given application. This is especially useful for applications that are shared by a number of users. An application programmer could put a *system* Application specific resource file in a system directory like $(NCARG_SYSAPPRES). Then, the application programmer can provide appropriate defaults for the application for all users.

Additionally, the end user also has the ability to provide an Application specific resource file in this case. If the application explicitly creates the **App** object, then there are four resource files that get used, in an increasing order of precedance:

1. $(NCARG_SYSRESFILE)
2. System Application Specific Resource file.
3. $(NCARG_USRRESFILE)
4. Users Application Specific Resource file.

The Application Specific resource files are found by using the resources in the **App** class object. The name of the **App** class object indicates the base name of the Application resource files. The *appFileSuffix* resource indicates the suffix of the Application resource files.

The *appUsrDir* resource indicates the directory to look for the Users Application resource file. While the *appSysDir* resource is used to specify the directory the System Application Specific resource file can be found in.

The *appResources* resources is a very useful resource for Application developers. It allows you to defined your own resource names. This resource contains an array of string names. Each one of the names contained in this resource will become a valid resource to that **App** object. Resource names should be picked with some care, since not all names will work. For example, since the "." (period) is a special character for resource file syntax, a name containing a period would not be a good choice. Also, if you pick a name that is already a resource name for the **App** object, you will actually be interacting with the **App** object defined resource, and not the *Application* defined one. It is important to realize that all resource values for *appResources* defined resources are stored as strings. This means that all assigns are converted to strings, and all getvalues are converted from strings.

Support functions

The **App** object defines the following support functions:

NhlAppGetDefaultParentID
: This function is used to retrieve the ID of the **App** object that currently has it's *appDefaultParent* resource set to *True*.

NhlIsApp
: This function just determines if a given object id identifies an **App** class object.

See also

- Initialization functions
- NCAR Graphics Environment Variables
- Create function
- Resource Database

Copyright

Copyright 1987-1999 University Corporation for Atmospheric Research
The use of this Software is governed by a License Agreement.

NCAR Graphics is a registered trademark of the University Corporation for Atmospheric Research.

Base class

The **Base** class provides the base functionality used by all objects in the HLU library.

Synopsis

```
Header file:            ncarg/hlu/Base.h
Class name:             baseClass
```

```
Class pointer:           <Not referenceable>
Fortran class function: <Not referenceable>
Superclass:              <NONE>
Composite classes:       <NONE>
```

Class-defined types

```
Type name:               NhlTPointer
Definition:              typedef void *NhlPointer;

Type name:               NhlTVariable
Definition:              Any "scalar" value.

Type name:               NhlTString
Definition:              typedef char *NhlString;

Type name:               NhlTCharacter
Definition:              char

Type name:               NhlTByte
Definition:              typedef char NhlByte;

Type name:               NhlTShort
Definition:              short

Type name:               NhlTInteger
Definition:              int

Type name:               NhlTLong
Definition:              long

Type name:               NhlTFloat
Definition:              float

Type name:               NhlTDouble
Definition:              double

Type name:               NhlTBoolean
Definition:              typedef int NhlBoolean; /* 0 is False, non-0 is True */

Type name:               NhlTFont
Definition:              typedef int NhlFont; /* pseudo-enum */
                                 {0,     "pwritx"},
                                 {1,     "default"},
                                 {1,     "ugly"},
                                 {2,     "cartographic_roman"},
                                 {3,     "cartographic_greek"},
                                 {4,     "simplex_roman"},
                                 {5,     "simplex_greek"},
                                 {6,     "simplex_script"},
                                 {7,     "complex_roman"},
                                 {8,     "complex_greek"},
                                 {9,     "complex_script"},
                                 {10,    "complex_italic"},
                                 {11,    "complex_cyrillic"},
                                 {12,    "duplex_roman"},
                                 {13,    "triplex_roman"},
                                 {14,    "triplex_italic"},
                                 {15,    "gothic_german"},
                                 {16,    "gothic_english"},
```

```
{17,    "gothic_italian"},
{18,    "math_symbols"},
{19,    "symbol_set1"},
{20,    "symbol_set2"},
{21,    "helvetica"},
{22,    "helvetica-bold"},
{25,    "times-roman"},
{26,    "times-bold"},
{29,    "courier"},
{30,    "courier-bold"},
{33,    "greek"},
{34,    "math-symbols"},
{35,    "text-symbols"},
{36,    "weather1"},
{37,    "weather2"},
{121,    "o_helvetica"},
{122,    "o_helvetica-bold"},
{125,    "o_times-roman"},
{126,    "o_times-bold"},
{129,    "o_courier"},
{130,    "o_courier-bold"},
{133,    "o_greek"},
{134,    "o_math-symbols"},
{135,    "o_text-symbols"},
{136,    "o_weather1"},
{137,    "o_weather2"},
};
```

Type name: NhlTGenArray
Definition: Any array.

Type name: NhlTPointerGenArray
Definition: An array of NhlPointer elements.

Type name: NhlTStringGenArray
Definition: An array of NhlString elements.

Type name: NhlTByteGenArray
Definition: An array of NhlByte elements.

Type name: NhlTCharacterGenArray
Definition: An array of char elements.

Type name: NhlTShortGenArray
Definition: An array of short elements.

Type name: NhlTIntegerGenArray
Definition: An array of int elements.

Type name: NhlTLongGenArray
Definition: An array of long elements.

Type name: NhlTFloatGenArray
Definition: An array of float elements.

Type name: NhlTDoubleGenArray
Definition: An array of double elements.

Type name: NhlTBooleanGenArray
Definition: An array of NhlBoolean elements.

```
Type name:               NhlTObjId
Definition:              An object ID.

Type name:               NhlTObjIdGenArray
Definition:              An array of object ID elements.
```

Resources

The **Base** class does not define any public resources.

Description

This object is the base of all other objects in the HLU library. It provides much of the functionality that is available for all other objects in the library.

Support functions

The following support functions are defined for objects of type **Base**:

NhlCreate
: The **NhlCreate** function is used to create an instance of any HLU object.

NhlSetValues
: The **NhlSetValues** function is used to modify the current values of any HLU object.

NhlGetValues
: The **NhlGetValues** function is used to retrieve the current values of any HLU object.

NhlDraw
: The **NhlDraw** function is the way to tell an object to produce its graphical output. Calling the **Draw** Function on a Workstation object causes all of that Workstation object's children to draw.

NhlDestroy
: The **NhlDestroy** function is used to destroy any HLU object, freeing all memory and system resources used by the object.

See also

- NhlCreate
- NhlSetValues
- NhlGetValues
- NhlDraw
- NhlDestroy

Copyright

Copyright 1987-1999 University Corporation for Atmospheric Research
The use of this Software is governed by a License Agreement.

NCAR Graphics is a registered trademark of the University Corporation for Atmospheric Research.

ContourPlot class

The **ContourPlot** class draws line or filled contours with labels and annotations.

Synopsis

```
Header file:             ncarg/hlu/ContourPlot.h
Class name:              contourPlotClass
Class pointer:           NhlcontourPlotClass
Fortran class function: NHLFCONTOURPLOTCLASS
Superclass:              DataComm
Composite classes:       LogLinTransformation,IrregularTransformation,PlotManager

Data specific class
Class name:              contourPlotDataDepClass
Class pointer:           NhlcontourPlotDataDepClass
Fortran class function: NHLFCONTOURPLOTDATADEPCLASS
Superclass:              DataSpec
```

Class-defined types

```
Type name:               NhlTcnLevelUseMode
Definition:
typedef enum _NhlcnLevelUseMode {
        NhlNOLINE                 = 0,    /* "NoLine"              */
        NhlLINEONLY               = 1,    /* "LineOnly"            */
        NhlLABELONLY              = 2,    /* "LabelOnly"           */
        NhlLINEANDLABEL           = 3     /* "LineAndLabel"        */
} NhlcnLevelUseMode;

Type name:               NhlTcnLevelUseModeGenArray
Definition:              An array of NhlTcnLevelUseMode elements.


Type name:               NhlTcnLineLabelPlacementMode
Definition:
typedef enum _NhlcnLineLabelSpacingMode {
        NhlCONSTANT               = 0,    /* "Constant"            */
        NhlRANDOMIZED             = 1,    /* "Randomized"          */
        NhlCOMPUTED               = 2     /* "Computed"            */
```

```
} NhlcnLineLabelSpacingMode;

Type name:              NhlTcnHighLowLabelOverlapMode
Definition:
typedef enum _NhlcnHighLowLabelOverlapMode {
        NhlIGNOREOVERLAP        = 0,1   /* "IgnoreOverlap"      */
        NhlOMITOVERHL           = 2,3   /* "OmitOverHL"         */
        NhlOMITOVERVP           = 4,5   /* "OmitOverVP"         */
        NhlOMITOVERVPANDHL      = 6,7   /* "OmitOverVPAndHL"    */
        NhlADJUSTVP             = 8,9   /* "AdjustVP"           */
        NhlADJUSTVPOMITOVERHL   = 10,11 /* "AdjustVPOmitOverHL" */
} NhlcnHighLowLabelOverlapMode;
```

Resources

Local resources

```
+-----------------------------------------------------------------+
|               ContourPlot Resource Set                          |
|-----------------------------------------------------------------|
| NAME                           TYPE                    ACCESS   |
|       CLASS                            DEFAULT                  |
|=================================================================|
| cnScalarFieldData              NhlTInteger             RCSG     |
|       CnScalarFieldData                <None>                   |
|-----------------------------------------------------------------|
| cnLevelSelectionMode           NhlTLevelSelectionMode  RCSG     |
|       LevelSelectionMode               "AutomaticLevels"        |
|-----------------------------------------------------------------|
| cnLevelCount                   NhlTInteger                G     |
|       CnLevelCount                     <dynamic>                |
|-----------------------------------------------------------------|
| cnMaxLevelCount                NhlTInteger             RCSG     |
|       MaxLevelCount                    16                       |
|-----------------------------------------------------------------|
| cnLevelSpacingF                NhlTFloat               RCSG     |
|       LevelSpacingF                    5.0                      |
|-----------------------------------------------------------------|
| cnMinLevelValF                 NhlTFloat               RCSG     |
|       MinLevelValF                     <dynamic>                |
|-----------------------------------------------------------------|
| cnMaxLevelValF                 NhlTFloat               RCSG     |
|       MaxLevelValF                     <dynamic>                |
|-----------------------------------------------------------------|
| cnLevels                       NhlTFloatGenArray       RCSG     |
|       Levels                           <dynamic>                |
|-----------------------------------------------------------------|
| cnMonoLevelFlag                NhlTBoolean             RCSG     |
|       CnMonoLevelFlag                  False                    |
|-----------------------------------------------------------------|
| cnLevelFlag                    NhlTcnLevelUseMode      RCSG     |
|       CnLevelFlags                     "LineOnly"               |
|-----------------------------------------------------------------|
| cnLevelFlags                   NhlTcnLevelUseModeGenArray RCSG  |
|       CnLevelFlags                     <dynamic>                |
|-----------------------------------------------------------------|
| cnLineLabelInterval            NhlTFloat               RCSG     |
```

```
      CnLineLabelInterval             2
---------------------------------------------------------------
 cnLinesOn                     NhlTBoolean             RCSG
      CnLinesOn                       True
---------------------------------------------------------------
 cnLineDrawOrder               NhlTDrawOrder           RCSG
      CnLineDrawOrder                 "Draw"
---------------------------------------------------------------
 cnMonoLineColor               NhlTBoolean             RCSG
      CnMonoLineColor                 True
---------------------------------------------------------------
 cnLineColor                   NhlTColorIndex          RCSG
      LineColor                       "Foreground"
---------------------------------------------------------------
 cnLineColors                  NhlTColorIndexGenArray  RCSG
      CnLineColors                    <dynamic>
---------------------------------------------------------------
 cnMonoLineDashPattern         NhlTBoolean             RCSG
      CnMonoLineDashPattern           True
---------------------------------------------------------------
 cnLineDashPattern             NhlTDashIndex           RCSG
      LineDashPattern                 "SolidLine"
---------------------------------------------------------------
 cnLineDashPatterns            NhlTDashIndexGenArray   RCSG
      CnLineDashPatterns              <dynamic>
---------------------------------------------------------------
 cnMonoLineThickness           NhlTBoolean             RCSG
      CnMonoLineThickness             True
---------------------------------------------------------------
 cnLineThicknessF              NhlTFloat               RCSG
      LineThicknessF                  1.0
---------------------------------------------------------------
 cnLineThicknesses             NhlTFloatGenArray       RCSG
      CnLineThicknesses               1.0 for all elements
---------------------------------------------------------------
 cnLineDashSegLenF             NhlTFloat               RCSG
      LineDashSegLenF                 0.15
---------------------------------------------------------------
 cnRasterModeOn                NhlTBoolean             RCSG
      CnRasterModeOn                  False
---------------------------------------------------------------
 cnRasterSmoothingOn           NhlTBoolean             RCSG
      CnRasterSmoothingOn             False
---------------------------------------------------------------
 cnRasterMinCellSizeF          NhlTFloat               RCSG
      CnRasterMinCellSizeF            0.001
---------------------------------------------------------------
 cnRasterSampleFactorF         NhlTFloat               RCSG
      CnRasterSampleFactorF           False
---------------------------------------------------------------
 cnRasterCellSizeF             NhlTFloat               RCSG
      CnRasterCellSizeF               <dynamic>
---------------------------------------------------------------
 cnFillOn                      NhlTBoolean             RCSG
      CnFillOn                        False
---------------------------------------------------------------
 cnFillDrawOrder               NhlTDrawOrder           RCSG
      CnFillDrawOrder                 "Draw"
---------------------------------------------------------------
 cnFillBackgroundColor         NhlTColorIndex          RCSG
      FillBackgroundColor             "Transparent"
```

```
-----------------------------------------------------------------
 cnMonoFillColor              NhlTBoolean             RCSG
       CnMonoFillColor                False
-----------------------------------------------------------------
 cnFillColor                  NhlTColorIndex          RCSG
       FillColor                      "Foreground"
-----------------------------------------------------------------
 cnFillColors                 NhlTColorIndexGenArray  RCSG
       CnFillColors                   <dynamic>
-----------------------------------------------------------------
 cnMonoFillPattern            NhlTBoolean             RCSG
       CnMonoFillPattern              True
-----------------------------------------------------------------
 cnFillPattern                NhlTFillIndex           RCSG
       FillPattern                    "SolidFill"
-----------------------------------------------------------------
 cnFillPatterns               NhlTFillIndexGenArray   RCSG
       CnFillPatterns                 <dynamic>
-----------------------------------------------------------------
 cnMonoFillScale              NhlTBoolean             RCSG
       CnMonoFillScale                True
-----------------------------------------------------------------
 cnFillScaleF                 NhlTFloat               RCSG
       FillScaleF                     1.0
-----------------------------------------------------------------
 cnFillScales                 NhlTFloatGenArray       RCSG
       CnFillScales                   1.0 for all elements
-----------------------------------------------------------------
 cnLabelDrawOrder             NhlTDrawOrder           RCSG
       CnLabelDrawOrder               "Draw"
-----------------------------------------------------------------
 cnLabelMasking               NhlTBoolean             RCSG
       CnLabelMasking                 False
-----------------------------------------------------------------
 cnLowUseHighLabelRes         NhlTBoolean             RCSG
       CnLowUseHighLabelRes           False
-----------------------------------------------------------------
 cnHighUseLineLabelRes        NhlTBoolean             RCSG
       CnHighUseLineLabelRes          False
-----------------------------------------------------------------
 cnConstFUseInfoLabelRes      NhlTBoolean             RCSG
       CnConstFUseInfoLabelRes        False
-----------------------------------------------------------------
 cnHighLowLabelOverlapMode NhlTcnHighLowLabelOverlapMode RCSG
       CnHighLowLabelOverlapMode      "IgnoreOverlap"
-----------------------------------------------------------------
 cnLabelScalingMode           NhlTScalingMode         RCSG
       CnLabelScalingMode             "ScaleFactor"
-----------------------------------------------------------------
 cnLabelScaleValueF           NhlTFloat               RCSG
       CnLabelScaleValueF             1.0
-----------------------------------------------------------------
 cnLabelScaleFactorF          NhlTFloat                  G
       CnLabelScaleFactorF            <dynamic>
-----------------------------------------------------------------
 cnMaxDataValueFormat         NhlTString              RCSG
       CnMaxDataValueFormat           NULL
-----------------------------------------------------------------
 cnLineLabelsOn               NhlTBoolean             RCSG
       PlotLabelsOn                   True
-----------------------------------------------------------------
```

```
cnExplicitLineLabelsOn        NhlTBoolean             RCSG
      CnExplicitLineLabelsOn          False
---------------------------------------------------------------
cnLineLabelPlacementMode NhlTcnLineLabelPlacementMode    RCSG
      CnLineLabelPlacementMode        "Randomized"
---------------------------------------------------------------
cnLineLabelStrings            NhlTStringGenArray      RCSG
      CnLineLabelStrings              <dynamic>
---------------------------------------------------------------
cnMonoLineLabelFontColor      NhlTBoolean             RCSG
      CnMonoLineLabelFontColor        True
---------------------------------------------------------------
cnLineLabelFontColor          NhlTColorIndex          RCSG
      FontColor                       "Foreground"
---------------------------------------------------------------
cnLineLabelFontColors         NhlTColorIndexGenArray  RCSG
      CnLineLabelFontColors           <dynamic>
---------------------------------------------------------------
cnLineLabelFormat             NhlTString              RCSG
      NumberFormat                    "*+g"
---------------------------------------------------------------
cnLineLabelFontHeightF        NhlTFloat               RCSG
      FontHeightF                     <dynamic>
---------------------------------------------------------------
cnLineLabelFont               NhlTFont                RCSG
      Font                            "pwritx"
---------------------------------------------------------------
cnLineLabelFontAspectF        NhlTFloat               RCSG
      FontAspectF                     1.3125
---------------------------------------------------------------
cnLineLabelFontThicknessF     NhlTFloat               RCSG
      FontThicknessF                  1.0
---------------------------------------------------------------
cnLineLabelFontQuality        NhlTFontQuality         RCSG
      FontQuality                     "High"
---------------------------------------------------------------
cnLineLabelConstantSpacingF   NhlTFloat               RCSG
      TextConstantSpacingF            0.0
---------------------------------------------------------------
cnLineLabelAngleF             NhlTFloat               RCSG
      CnLineLabelAngleF               -1.0
---------------------------------------------------------------
cnLineLabelFuncCode           NhlTCharacter           RCSG
      TextFuncCode                    ':'
---------------------------------------------------------------
cnLineLabelBackgroundColor    NhlTColorIndex          RCSG
      FillBackgroundColor             "Background"
---------------------------------------------------------------
cnLineLabelPerimOn            NhlTInteger             RCSG
      EdgesOn                         True
---------------------------------------------------------------
cnLineLabelPerimSpaceF        NhlTFloat               RCSG
      EdgeBorderWidthF                0.33
---------------------------------------------------------------
cnLineLabelPerimColor         NhlTColorIndex          RCSG
      EdgeColor                       "Foreground"
---------------------------------------------------------------
cnLineLabelPerimThicknessF    NhlTFloat               RCSG
      EdgeThicknessF                  1.0
---------------------------------------------------------------
cnHighLabelsOn                NhlTBoolean             RCSG
```

	PlotLabelsOn		False	
cnHighLabelString	CnHighLabelString	NhlTString	NULL	RCSG
cnHighLabelFormat	NumberFormat	NhlTString	"*+g"	RCSG
cnHighLabelFontHeightF	FontHeightF	NhlTFloat	<dynamic>	RCSG
cnHighLabelFont	Font	NhlTFont	"pwritx"	RCSG
cnHighLabelFontColor	FontColor	NhlTColorIndex	"Foreground"	RCSG
cnHighLabelFontAspectF	FontAspectF	NhlTFloat	1.3125	RCSG
cnHighLabelFontThicknessF	FontThicknessF	NhlTFloat	1.0	RCSG
cnHighLabelFontQuality	FontQuality	NhlTFontQuality	"High"	RCSG
cnHighLabelConstantSpacingF	TextConstantSpacingF	NhlTFloat	0.0	RCSG
cnHighLabelAngleF	TextAngleF	NhlTFloat	0.0	RCSG
cnHighLabelFuncCode	TextFuncCode	NhlTCharacter	':'	RCSG
cnHighLabelBackgroundColor	FillBackgroundColor	NhlTColorIndex	"Background"	RCSG
cnHighLabelPerimOn	EdgesOn	NhlTInteger	True	RCSG
cnHighLabelPerimSpaceF	EdgeBorderWidthF	NhlTFloat	0.33	RCSG
cnHighLabelPerimColor	EdgeColor	NhlTColorIndex	"Foreground"	RCSG
cnHighLabelPerimThicknessF	EdgeThicknessF	NhlTFloat	1.0	RCSG
cnLowLabelsOn	PlotLabelsOn	NhlTBoolean	False	RCSG
cnLowLabelString	CnLowLabelString	NhlTString	NULL	RCSG
cnLowLabelFormat	NumberFormat	NhlTString	"*+g"	RCSG
cnLowLabelFontHeightF	FontHeightF	NhlTFloat	<dynamic>	RCSG

```
---------------------------------------------------------------
 cnLowLabelFont                 NhlTFont                RCSG
       Font                             "pwritx"
---------------------------------------------------------------
 cnLowLabelFontColor            NhlTColorIndex          RCSG
       FontColor                        "Foreground"
---------------------------------------------------------------
 cnLowLabelFontAspectF          NhlTFloat               RCSG
       FontAspectF                      1.3125
---------------------------------------------------------------
 cnLowLabelFontThicknessF       NhlTFloat               RCSG
       FontThicknessF                   1.0
---------------------------------------------------------------
 cnLowLabelFontQuality          NhlTFontQuality         RCSG
       FontQuality                      "High"
---------------------------------------------------------------
 cnLowLabelConstantSpacingF     NhlTFloat               RCSG
       TextConstantSpacingF             0.0
---------------------------------------------------------------
 cnLowLabelAngleF               NhlTFloat               RCSG
       TextAngleF                       0.0
---------------------------------------------------------------
 cnLowLabelFuncCode             NhlTCharacter           RCSG
       TextFuncCode                     ':'
---------------------------------------------------------------
 cnLowLabelBackgroundColor      NhlTColorIndex          RCSG
       FillBackgroundColor              "Background"
---------------------------------------------------------------
 cnLowLabelPerimOn              NhlTInteger             RCSG
       EdgesOn                          True
---------------------------------------------------------------
 cnLowLabelPerimSpaceF          NhlTFloat               RCSG
       EdgeBorderWidthF                 0.33
---------------------------------------------------------------
 cnLowLabelPerimColor           NhlTColorIndex          RCSG
       EdgeColor                        "Foreground"
---------------------------------------------------------------
 cnLowLabelPerimThicknessF      NhlTFloat               RCSG
       EdgeThicknessF                   1.0
---------------------------------------------------------------
 cnInfoLabelOn                  NhlTBoolean             RCSG
       AnnotationLabelsOn               True
---------------------------------------------------------------
 cnInfoLabelString              NhlTString              RCSG
       CnInfoLabelString                NULL
---------------------------------------------------------------
 cnInfoLabelFormat              NhlTString              RCSG
       NumberFormat                     "*+g"
---------------------------------------------------------------
 cnInfoLabelFontHeightF         NhlTFloat               RCSG
       FontHeightF                      <dynamic>
---------------------------------------------------------------
 cnInfoLabelTextDirection       NhlTTextDirection       RCSG
       TextDirection                    "Across"
---------------------------------------------------------------
 cnInfoLabelFont                NhlTFont                RCSG
       Font                             "pwritx"
---------------------------------------------------------------
 cnInfoLabelFontColor           NhlTColorIndex          RCSG
       FontColor                        "Foreground"
---------------------------------------------------------------
```

```
cnInfoLabelFontAspectF          NhlTFloat               RCSG
      FontAspectF                       1.3125
----------------------------------------------------------------
cnInfoLabelFontThicknessF       NhlTFloat               RCSG
      FontThicknessF                    1.0
----------------------------------------------------------------
cnInfoLabelFontQuality          NhlTFontQuality         RCSG
      FontQuality                       "High"
----------------------------------------------------------------
cnInfoLabelConstantSpacingF     NhlTFloat               RCSG
      TextConstantSpacingF              0.0
----------------------------------------------------------------
cnInfoLabelAngleF               NhlTFloat               RCSG
      TextAngleF                        0.0
----------------------------------------------------------------
cnInfoLabelFuncCode             NhlTCharacter           RCSG
      TextFuncCode                      ':'
----------------------------------------------------------------
cnInfoLabelBackgroundColor      NhlTColorIndex          RCSG
      FillBackgroundColor               "Background"
----------------------------------------------------------------
cnInfoLabelPerimOn              NhlTInteger             RCSG
      EdgesOn                           True
----------------------------------------------------------------
cnInfoLabelPerimSpaceF          NhlTFloat               RCSG
      EdgeBorderWidthF                  0.33
----------------------------------------------------------------
cnInfoLabelPerimColor           NhlTColorIndex          RCSG
      EdgeColor                         "Foreground"
----------------------------------------------------------------
cnInfoLabelPerimThicknessF      NhlTFloat               RCSG
      EdgeThicknessF                    1.0
----------------------------------------------------------------
cnInfoLabelZone                 NhlTInteger             RCSG
      CnInfoLabelZone                   3
----------------------------------------------------------------
cnInfoLabelSide                 NhlTPosition            RCSG
      CnInfoLabelSide                   "Bottom"
----------------------------------------------------------------
cnInfoLabelJust                 NhlTJustification       RCSG
      CnInfoLabelJust                   "TopRight"
----------------------------------------------------------------
cnInfoLabelParallelPosF         NhlTFloat               RCSG
      CnInfoLabelParallelPosF           1.0
----------------------------------------------------------------
cnInfoLabelOrthogonalPosF       NhlTFloat               RCSG
      CnInfoLabelOrthogonalPosF         0.02
----------------------------------------------------------------
cnNoDataLabelOn                 NhlTBoolean             RCSG
      AnnotationLabelsOn                True
----------------------------------------------------------------
cnNoDataLabelString             NhlTString              RCSG
      CnNoDataLabelString               NULL
----------------------------------------------------------------
cnConstFLabelOn                 NhlTBoolean             RCSG
      AnnotationLabelsOn                True
----------------------------------------------------------------
cnConstFLabelString             NhlTString              RCSG
      CnConstFLabelString               NULL
----------------------------------------------------------------
cnConstFLabelFormat             NhlTString              RCSG
```

NumberFormat	"*+g"	
cnConstFLabelFontHeightF	NhlTFloat	RCSG
FontHeightF	<dynamic>	
cnConstFLabelTextDirection	NhlTTextDirection	RCSG
TextDirection	"Across"	
cnConstFLabelFont	NhlTFont	RCSG
Font	"pwritx"	
cnConstFLabelFontColor	NhlTColorIndex	RCSG
FontColor	"Foreground"	
cnConstFLabelFontAspectF	NhlTFloat	RCSG
FontAspectF	1.3125	
cnConstFLabelFontThicknessF	NhlTFloat	RCSG
FontThicknessF	1.0	
cnConstFLabelFontQuality	NhlTFontQuality	RCSG
FontQuality	"High"	
cnConstFLabelConstantSpacingF	NhlTFloat	RCSG
TextConstantSpacingF	0.0	
cnConstFLabelAngleF	NhlTFloat	RCSG
TextAngleF	0.0	
cnConstFLabelFuncCode	NhlTCharacter	RCSG
TextFuncCode	':'	
cnConstFLabelBackgroundColor	NhlTColorIndex	RCSG
FillBackgroundColor	"Background"	
cnConstFLabelPerimOn	NhlTInteger	RCSG
EdgesOn	True	
cnConstFLabelPerimSpaceF	NhlTFloat	RCSG
EdgeBorderWidthF	0.33	
cnConstFLabelPerimColor	NhlTColorIndex	RCSG
EdgeColor	"Foreground"	
cnConstFLabelPerimThicknessF	NhlTFloat	RCSG
EdgeThicknessF	1.0	
cnConstFLabelZone	NhlTInteger	RCSG
CnConstFLabelZone	0	
cnConstFLabelSide	NhlTPosition	RCSG
CnConstFLabelSide	"Bottom"	
cnConstFLabelJust	NhlTJustification	RCSG
CnConstFLabelJust	"CenterCenter"	
cnConstFLabelParallelPosF	NhlTFloat	RCSG
CnConstFLabelParallelPosF	0.0	
cnConstFLabelOrthogonalPosF	NhlTFloat	RCSG
CnConstFLabelOrthogonalPosF	0.0	

cnMissingValPerimOn EdgesOn	NhlTInteger	False	RCSG
cnMissingValPerimThicknessF EdgeThicknessF	NhlTFloat	1.0	RCSG
cnMissingValPerimDashPattern EdgeDashPattern	NhlTDashIndex	"SolidLine"	RCSG
cnMissingValPerimColor EdgeColor	NhlTColorIndex	"Foreground"	RCSG
cnMissingValFillColor FillColor	NhlTColorIndex	"Background"	RCSG
cnMissingValFillPattern FillPattern	NhlTInteger	"HollowFill"	RCSG
cnMissingValFillScaleF FillScaleF	NhlTFloat	1.0	RCSG
cnGridBoundPerimOn EdgesOn	NhlTInteger	False	RCSG
cnGridBoundPerimThicknessF EdgeThicknessF	NhlTFloat	1.0	RCSG
cnGridBoundPerimDashPattern EdgeDashPattern	NhlTDashIndex	"SolidLine"	RCSG
cnGridBoundPerimColor EdgeColor	NhlTColorIndex	"Foreground"	RCSG
cnOutOfRangePerimOn EdgesOn	NhlTInteger	False	RCSG
cnOutOfRangePerimThicknessF EdgeThicknessF	NhlTFloat	1.0	RCSG
cnOutOfRangePerimDashPattern EdgeDashPattern	NhlTDashIndex	"SolidLine"	RCSG
cnOutOfRangePerimColor EdgeColor	NhlTColorIndex	"Foreground"	RCSG
cnSmoothingOn CnSmoothingOn	NhlTBoolean	False	RCSG
cnSmoothingTensionF CnSmoothingTensionF	NhlTFloat	-2.5	RCSG
cnSmoothingDistanceF CnSmoothingDistanceF	NhlTFloat	0.01	RCSG
cnMaxPointDistanceF CnMaxPointDistanceF	NhlTFloat	0.05	RCSG
cnExplicitLabelBarLabelsOn CnExplicitLabelBarLabelsOn	NhlTBoolean	False	RCSG

```
| cnLabelBarEndLabelsOn         NhlTBoolean             RCSG     |
|       CnLabelBarEndLabelsOn           False                    |
|----------------------------------------------------------------|
| cnExplicitLegendLabelsOn      NhlTBoolean             RCSG     |
|       CnExplicitLegendLabelsOn        False                    |
|----------------------------------------------------------------|
| cnLegendLevelFlags            NhlTcnLevelUseModeGenArray RCSG  |
|       CnLegendLevelFlags              NULL                     |
|----------------------------------------------------------------|
| cnConpackParams               NhlTStringGenArray      RCS      |
|       CnConpackParams                 NULL                     |
+----------------------------------------------------------------+
```

Data specific resources

The **ContourPlot** class does not currently use any data specific resources.

Composite resources

Transformation resources

Transformation class resources specify the extent and direction of the data coordinate system. As the superclass of both **LogLinTransformation** and **IrregularTransformation**, you can access all **Transformation** resources. However, note that **ContourPlot** intercepts these resources, as follows:

- trXMinF
  By default *trXMinF* is set to the minimum data coordinate value along the X Axis, as determined from the contents of the **ScalarField** object.
- trXMaxF
  By default *trXMaxF* is set to the maximum data coordinate value along the X Axis, as determined from the contents of the **ScalarField** object.
- trXReverse
  By default *trXReverse* is set based on the direction of the X Axis implied by the contents of the **ScalarField** object.
- trYMinF
  By default *trYMinF* is set to the minimum data coordinate value along the Y Axis, as determined from the contents of the **ScalarField** object.
- trYMaxF
  By default *trYMaxF* is set to the maximum data coordinate value along the Y Axis, as determined from the contents of the **ScalarField** object.
- trYReverse
  By default *trYReverse* is set based on the direction of the Y Axis implied by the contents of the **ScalarField** object.

LogLinTransformation resources

The **ContourPlot** class uses the **LogLinTransformation** to handle its transformations as long as neither of the **ScalarField** array resources, *sfXArray* or *sfYArray*, is set. **LogLinTransformation** has resources for selecting between linear and logarithmic coordinates for each axis.

You can access all **LogLinTransformation** resources. However, note that **ContourPlot** intercepts **LogLinTransformation** resources, as follows:

- trXLog
  ContourPlot issues a warning if *trXLog* is set True when the set value of *trXMinF* is less than or equal to 0.0. In this case it resets *trXLog* to False. If the **IrregularTransformation** resource *trXAxisType* is set, **ContourPlot** sets *trXLog* True if *trXAxisType* is set to `LogAxis`. If *trXAxisType* is set to any other value, it sets *trXLog* False.
- trYLog
  ContourPlot issues a warning if *trYLog* is set True when the set value of *trYMinF* is less than or equal to 0.0. In this case it resets *trYLog* to False. If the **IrregularTransformation** resource *trYAxisType* is set, **ContourPlot** sets *trYLog* True if *trYAxisType* is set to `LogAxis`. If *trYAxisType* is set to any other value, it sets *trYLog* False.

IrregularTransformation resources

ContourPlot automatically uses the **IrregularTransformation** instead of the **LogLinTransformation** to handle its transformations if either of the **ScalarField** arrays *sfXArray* or *sfYArray* is set, implying that one or both of the coordinate axes is irregularly spaced.

All **Transformation** superclass resources are accessible. However, **ContourPlot** blocks access to all resources specific to the **IrregularTransformation** except for:

- trXTensionF
- trYTensionF

and for the following intercepted resources:

- trXAxisType
  If the **ScalarField** resource *sfXArray* is non-`NULL` and *trXAxisType* is set to any value other than `IrregularAxis`, **ContourPlot** switches to a coordinate extent bounded by 0 and the length of the X-Axis dimension minus one. If *trXAxisType* is not set, but the **LogLinTransformation** resource *trXLog* is set, **ContourPlot** sets *trXAxisType* to `LogAxis` if *trXLog* is True; if *trXLog* is False, it changes *trXAxisType* to `LinearAxis` if it had been set to `LogAxis` and leaves it unchanged otherwise. *trXAxisType* can be set to `LogAxis` without error only when the X-Axis coordinate extent as passed from the **ScalarField** is entirely positive. If this is not the case, *trXAxisType* will default to `LinearAxis`.

  V4.1 Status Note 1

- trYAxisType
  If the **ScalarField** resource *sfYArray* is non-`NULL` and *trYAxisType* is set to any value other than `IrregularAxis`, **ContourPlot** switches to a coordinate extent bounded by 0 and the length of the Y-Axis dimension minus one. If *trYAxisType* is not set, but the **LogLinTransformation** resource *trYLog* is set, **ContourPlot** sets *trYAxisType* to `LogAxis` if *trYLog* is True; if *trYLog* is False, it changes *trYAxisType* to `LinearAxis` if it had been set to `LogAxis` and leaves it unchanged otherwise. *trYAxisType* can be set to `LogAxis` without error

only when the Y-Axis coordinate extent as passed from the **ScalarField** is entirely positive. If this is not the case, *trYAxisType* will default to `LinearAxis`.

V4.1 Status Note 1

PlotManager resources

If *tfPlotManagerOn* is True when a **ContourPlot** object is created, you can access all **PlotManager** resources. However, note that **ContourPlot** intercepts certain **PlotManager** resources, as follows:

- pmLegendDisplayMode
  The default value of *pmLegendDisplayMode* is set to `Never`.
- pmLabelBarDisplayMode
  The default value of *pmLabelBarDisplayMode* is set to `Never`.
- pmTickMarkDisplayMode
  The default value of *pmTickMarkDisplayMode* is set to `Conditional`.
- pmTitleDisplayMode
  The default value of *pmTitleDisplayMode* is set to `Conditional`.

You can also access resources for any of composite classes of the **PlotManager** class. However, the **PlotManager** class modifies the access and behavior of some of the resources belonging to these classes, as follows:

- TickMark resources as modified by PlotManager
- Title resources as modified by PlotManager
- LabelBar resources as modified by PlotManager
- Legend resources as modified by PlotManager

The **ContourPlot** class itself also modifies the access and behavior of a number of **LabelBar** and **Legend** resources.

Additional modifications to LabelBar resources

The **ContourPlot** class disables a number of **LabelBar** resources, since it sets them automatically based on the current values of certain relevant **ContourPlot** resources. The disabled resources include:

- lbBoxCount
- lbMonoFillColor
- lbFillColor
- lbFillColors
- lbMonoFillPattern
- lbFillPattern
- lbFillPatterns
- lbMonoFillScale
- lbFillScaleF
- lbFillScales

In addition, the **ContourPlot** class intercepts certain **LabelBar** resources, as follows:

- lbLabelStrings

  Any set value of this resource is ignored unless the **ContourPlot** resource *cnExplicitLabelBarLabelsOn* is set True.
- lbLabelFuncCode

  Any set value of this resource is ignored unless the **ContourPlot** resource *cnExplicitLabelBarLabelsOn* is set True.
- lbLabelAlignment

  Any set value of this resource is ignored unless the **ContourPlot** resource *cnExplicitLabelBarLabelsOn* is set True.

Additional modifications to Legend resources

The **ContourPlot** object disables a number of **Legend** resources, since it sets them automatically based on the current values of certain relevant **ContourPlot** resources. The disabled resources include:

- lgItemCount
- lgMonoItemType
- lgItemType
- lgItemTypes
- lgMonoDashIndex
- lgDashIndex
- lgDashIndexes
- lgMonoLineColor
- lgLineColor
- lgLineColors
- lgMonoLineThickness
- lgLineThicknessF
- lgLineThicknesses
- lgMonoLineDashSegLen
- lgLineDashSegLenF
- lgLineDashSegLens
- lgMonoMarkerIndex
- lgMarkerIndex
- lgMarkerIndexes
- lgMonoMarkerColor
- lgMarkerColor
- lgMarkerColors
- lgMonoMarkerThickness
- lgMarkerThicknessF
- lgMarkerThicknesses
- lgMonoMarkerSize
- lgMarkerSizeF
- lgMarkerSizes
- lgMonoLineLabelFontHeight
- lgLineLabelFontHeightF
- lgLineLabelFontHeights
- lgMonoLineLabelFontColor
- lgLineLabelFontColor

- lgLineLabelFontColors
- lgLineLabelStrings
- lgLineLabelFontAspectF
- lgLineLabelFontThicknessF
- lgLineLabelFontQuality
- lgLineLabelConstantSpacingF
- lgLineLabelFuncCode

In addition, the **ContourPlot** class intercepts two **Legend** resources, as follows:

- lgLabelStrings
  Any set value of this resource is ignored unless the **ContourPlot** resource *cnExplicitLabelBarLabelsOn* is set True.
- lgLabelFuncCode
  Any set value of this resource is ignored unless the **ContourPlot** resource *cnExplicitLabelBarLabelsOn* is set True.

Superclass resources

You can set all resources defined by the superclasses of the **ContourPlot** object class, including:

- **DataComm** resources
- **Transform** resources
- **View** resources

Description

The **ContourPlot** object draws a contour plot from two-dimensional data provided by the **ScalarField** object. The plot may consist only of lines, or may be filled using color and/or fill patterns to distinguish the various contour regions. You can control how the contour levels are selected. Selected contour levels, as well as regional high and low points, may be labeled. You can cause a number of annotational items to appear, including an informational label, tick marks, titles, a labelbar, and a line legend. These items automatically adjust themselves to the current state of the **ContourPlot** object. You may also differentiate certain special areas with unique lines or fill, and control to some extent the algorithms used to generate the contour lines.

The **ContourPlot** object provides what may seem like an overwhelming number of resources, but keep in mind that generally you will need to use only a few of these resources. In fact, there is only one resource you need to set explicitly in order to generate a meaningful contour plot. You need to focus only on resources that modify the features of the **ContourPlot** object that are important to you. All the others will default to a generally usable value. Moreover, resources that provide control over related attributes of the various **ContourPlot** elements have related names. As an example, all resources that specify the color of the text used for the various labels available in the **ContourPlot** object have names of the form *cn...LabelFontColor(s)*, where the plural form is used for resources that represent arrays.

Data input

The only resource you must set in order to generate an actual contour plot is *cnScalarFieldData*. This resource specifies the id of an existing **ScalarField** object. The **ScalarField** object accepts data in a variety of types and configurations and allows you to specify how it is to be interpreted. As received by the **ContourPlot** object, the data are accompanied by information specifying the extents of the data in the data coordinate system and the minimum and maximum data values. Also, if either or both of the data coordinate axes are irregularly spaced, the **ContourPlot** object receives information defining this spacing.

Coordinate transformations

ContourPlot intrinsically supports linear, logarithmic, and irregular rectangular gridded data coordinate spaces. It does *not yet*, on its own, support the transformation of an irregular data space into either a linear or a logarithmic space. However, such transformations are easily accomplished using the overlay mechanism.

ContourPlot instantiates child objects to manage transformations between the data coordinate system and NDC space. The **LogLinTransformation** manages linear and logarithmic transformations, and the **IrregularTransformation** manages the transformation if one or both axes are irregularly spaced. By default the data coordinate extents are based on the extents of the supplied **ScalarField** object data, but you may adjust them using resources belonging to the transformation objects.

Use of the LogLinTransformation object

ContourPlot uses a **LogLinTransformation** object as long as the **ScalarField** resources *sfXArray* and *sfYArray* are both `NULL`. The coordinate extents of a linear axis may arbitrarily intersect or encompass the data extent. If you set a logarithmic axis, then the coordinate extent of that axis must be entirely positive, but otherwise may intersect or extend beyond the data extent.

Use of the IrregularTransformation object

If either of the **ScalarField** coordinate array resources *sfXArray* and *sfYArray* are non-`NULL`, then **ContourPlot** uses an **IrregularTransformation** object. Note that **ContourPlot** treats an axis with an associated coordinate array as irregular even if the coordinate array actually has evenly spaced values. **ContourPlot** represents an irregular axis not by stretching and compressing various regions of the plot, but by displaying it with irregularly spaced tick marks.

In addition to a small performance penalty, there are some restrictions associated with use of the **IrregularTransformation** object. Although you may limit the coordinate extent to a subspace of the data coordinate extent of the **ScalarField** object data, you are not allowed to define a coordinate range that extends outside the range of the data coordinates of an irregular axis. Using the **IrregularTransformation** resources trXAxisType or trYAxisType, it is possible to set an irregular axis to `LinearAxis` or even, under certain conditions, to `LogAxis`, but the results are probably not what you want. Since **ContourPlot** does not intrinsically support a linearization transformation for irregularly spaced data, it can only switch to a linear system by replacing the data coordinates with array index coordinates, which are, in fact, linearly spaced. To properly transform irregularly spaced data into a linear or logarithmic coordinate system, you must use the overlay mechanism (V4.1 Status Note 1).

Overlays

By overlaying a **contourplot** on various types of base plot, it is possible to map contours of arbitrary **ScalarField** data into a variety of coordinate spaces. You can overlay a **contourplot** on a **mapplot** to transform the data into any of 10 different map projections. You can overlay contours of irregularly gridded data on a **loglinplot** to have the data appear in a linear or logarithmic coordinate space. You can also overlay any contoured data on an **irregularplot**, either to project regularly gridded data into an irregular coordinate space, or perhaps to overcome the data extent limitations of the **ContourPlot**'s intrinsic **IrregularTransformation** object. You can overlay a **contourplot** on any type of base plot in a one-step operation using the **NhlAddOverlay** function.

Controlling draw order

The **ContourPlot** object allows you to control the order in which its major elements are drawn. You may separately control the drawing order for lines, fill, and labels using the resources *cnLineDrawOrder*, *cnFillDrawOrder*, and *cnLabelDrawOrder*. There are three drawing phases: the *predraw* phase, the *draw* phase, and the *postdraw* phase. By default, **ContourPlot** draws all plot elements during the draw phase: first the fill, then the lines, and finally the labels. When a **ContourPlot** is drawn by itself, this is usually a reasonable order that results in the important features of the plot being visible. However, when a **ContourPlot** object is overlaid over a base plot, particularly when masking is involved, you often need to adjust the drawing order to ensure that the features you want to see remain visible.

Note that annotations associated with the **ContourPlot**, including the informational and constant field labels, are always drawn during the *postdraw* phase.

Controlling annotations

The **ContourPlot** object supports three kinds of annotations: those that it creates on its own, the informational and constant field embedded annotations; those that are created by its composite **PlotManager** object, including **Title**, **Tickmark**, **Labelbar**, and **Legend** intrinsic annotations; and finally arbitrary user-defined external annotations. The **Transform** class resource *tfPlotManagerOn* must be set True (its default value) when the **ContourPlot** object is created in order for the intrinsic or external annotations to be available. The informational label (and the constant field label under the appropriate conditions) appear by default. Assuming *tfPlotManagerOn* is set True, tick marks also appear by default. Titles appear if you set the appropriate title string resource to any non-NULL string value. In order for a labelbar or legend to appear, you must set the **PlotManager** resources *pmLabelBarDisplayMode* or *pmLegendDisplayMode* to the values `Conditional` or `Always`.

Except for the tick marks and the titles, all these annotations are positioned using resources that follow the PlotManager Location Control Model. The **ContourPlot** object provides resources for controlling attributes of the informational and the constant field labels. You can control the resources of the objects that render the intrinsic annotations directly, subject to some access and behavioral modifications imposed by the **PlotManager** object and by the **ContourPlot** object itself. These objects are the **TickMark** object, the **Title** object, the **LabelBar** object, and the **Legend** object.

Selecting levels

By appropriately setting the *cnLevelSelectionMode* resource, you can choose from four level-selection modes to set up the levels used in the contour plot. The default mode, `AutomaticLevels`, is easy to use. It selects a "nice" spacing starting from the smallest relatively round number greater than the minimum data value, such that the number of contour levels is as close as possible to, but less than, the value of the resource *cnMaxLevelCount*. `EqualSpacedLevels` mode defines exactly *cnMaxLevelCount* levels spaced evenly from the data minimum value to the data maximum value. In `ManualLevels` mode, you set the maximum and minimum contour levels using the resources *cnMinLevelValF* and *cnMaxLevelValF* with a spacing defined by the resource *cnLevelSpacingF*. Finally, `Explicit` mode allows you to define the level values yourself by setting the array resource *cnLevels*. The `ManualLevels` and `ExplicitLevels` modes have the advantage that they are independent of the maximum and minimum values of the particular dataset you are contouring, and therefore can be used to enforce consistency in the contouring of a series of related datasets. `ExplicitLevels` is the only mode that allows you to establish variably spaced contour levels.

Once you have established the contour levels, you can retrieve the number of levels actually used by getting the value of the read-only resource *cnLevelCount*. Based on this value, you can set the *cnLevelFlags* enumerated array resource to specify whether you want lines only, labels only, lines and labels, or nothing at each individual contour level. This resource also has a scalar equivalent, *cnLevelFlag*, which you can set instead if you want all levels to appear the same way. If you set neither of these resources, then line labels automatically appear at contour levels spaced apart by the value of the resource *cnLineLabelInterval*, set by default to the value 2.

Contour lines

You can control whether contour lines appear at all by setting the boolean resource *cnLinesOn*. By default it is set True, meaning that contour lines will appear. If set False, no contour lines will appear regardless of the setting of *cnLevelFlags* or *cnLevelFlag*. You can control the color, the dash pattern and the thickness of contour lines individually or uniformly as a group. Individual control is through the array resources *cnLineColors*, *cnLineDashPatterns*, and *cnLineThicknesses* while control as a group is through the scalar resources *cnLineColor*, *cnLineDashPattern*, and *cnLineThicknessF*. You choose the scalar resources by setting the boolean resources *cnMonoLineColor*, *cnMonoLineDashPattern*, and *cnMonoLineThickness* True; you choose the array resources by setting them False. There is also a scalar resource, *cnLineDashSegLenF* that sets the segment length for all dash patterns used.

Contour fill

ContourPlot supports two methods for filling contour regions: area and raster. Turn on area fill for contour regions by setting the boolean resource *cnFillOn* True. Turn on raster fill by setting the boolean resource *cnRasterModeOn* True. The default for both these resources is False.

Area fill

For non-raster fill, you can control the color, the fill pattern, and the scale of the fill pattern for all filled contour regions either individually or uniformly as a group. Individual control is through the array resources, *cnFillColors*, *cnFillPatterns*, and *cnFillScales* while control as a group is through the scalar resources *cnFillColor*, *cnFillPattern*, and *cnFillScaleF*. You choose the scalar resources by setting the boolean resources *cnMonoFillColor*, *cnMonoFillPattern*, and *cnMonoFillScale* True; you choose the array resources by setting them False.

Raster fill

ContourPlot performs raster fill by assigning appropriate colors to each cell of a two-dimensional array of equally sized rectangular cells superimposed on the area occupied by the data grid. If *cnRasterSmoothingOn* is set True, the level (and hence the color) assigned to each cell is determined by interpolating the values of neighboring points in the data grid. If *cnRasterSmoothingOn* is False (the default), **ContourPlot** creates a discrete raster plot: any raster cell whose center lies within the rectangular area bounded by lines halfway between each grid point (in data space) is given the color assigned to the level representing the grid point datum.

By default, **ContourPlot** dynamically tries to achieve an optimal size for the cell array based on the resolution of the workstation, the density of the data grid, the data transformation in effect, and whether smoothing is turned on. However, resources are provided for controlling the density of the raster cell grid. You can either set the cell size explicitly, using the resource *cnRasterCellSizeF*, or you can adjust the dynamically determined density, using the resource *cnRasterSampleFactorF*. The *cnRasterMinCellSizeF* resource sets a lower bound for the size of dynamically sized raster cells and consequently an upper limit on the raster cell density. Its purpose is to prevent the cell array from becoming unreasonably large for high-resolution workstations.

As with area fill, raster fill colors are specified using the *cnFillColors* resource. The fill pattern and fill scale resources are ignored when raster fill is enabled.

Labels

Almost two thirds of the **ContourPlot** object's resources involve attributes of various kinds of labels. This section describes the label types and classifies the label attributes.

ContourPlot supports five label types: line labels, high labels, low labels, an informational label, and a constant field label. The constant field label also has the ability to act as a label indicating that no data have been supplied. Each type has its own set of resources allowing individual control of the particular label type. The resources controlling label attributes have a generally uniform behavior for all label types. The resource names follow a consistent naming scheme for distinguishing the label types and their attributes. There are broadly three categories of label attributes that apply to all labels: those that control the content and format of the text string; those that control the appearance of the text, called *font attributes*; and those that control the appearance of the label background and perimeter. The informational and constant field labels support a fourth category of label attributes: those that control the location of the label. In addition to these general categories, there are several resources that apply globally to all label types, as well as a few that only apply to a single label type.

To simplify the management of label resources, there are several resources that allow you tie many of the resources of different label types together. *cnLowUseHighLabelRes*, when set True, causes all low label resources except *cnLowLabelString* to take on the value of the corresponding high label resource. Similarly, *cnHighUseLineLabelRes*, when set True, causes all high label resources except *cnHighLabelString* and *cnHighLabelAngleF* to take on the value of the corresponding line label resource. And finally, *cnConstFUseInfoLabelRes*, when set True, causes all constant field label resources except *cnConstFLabelString* to take on the value of the corresponding informational label resource.

Note that because of constraints imposed by the low level utilities, some attributes of high and low labels are tied together regardless of the setting of *cnLowUseHighLabelRes*. Also, the line label resources differ because several of them are array resources, allowing individual control of the labels that appear with each contour line.

The following sections discuss each label attribute category as it applies to all label types. The ellipsis marks (...) in the resource names below stand for the names of the label types:

- *Line* for line labels
- *High* for high labels
- *Low* for low labels
- *Info* for the informational label
- *ConstF* for the constant field label

String content and formatting attributes

These resources control the content and format of all the label types:

cn...Label(s)On
: Turns the label on and off. Line, high, and low labels use the plural form. The informational and constant field labels use the singular form.

cn...LabelString(s)
: Specifies the text of the label. Line labels use the plural form to indicate an array of strings.

cn...LabelFuncCode
: Sets the function code character used for the label text.

cn...LabelFormat
: Specifies the format string used for numeric substitutions within the label text.

Since the information in the labels provided by the **ContourPlot** object is closely tied to the values in the dataset being contoured, there are several resources that apply for all label types to numbers generated from the dataset. Using the resources *cnLabelScalingMode* and *cnLabelScaleValueF*, you can choose from a number of methods of scaling the values of the dataset. The point may be to make the label numbers fit within some standard range or simply to make them easier to read. Once you have set these resources, you can get the value of the read-only resource *cnLabelScaleFactorF* to determine the scale factor used to achieve the desired results. Multiplying the numbers in the labels by the value of *cnLabelScaleFactorF* gives the actual data values the numbers represent.

You specify the format string for numerical substitutions according to the HLU Floating Point Format Specification scheme. This scheme allows certain aspects of a number's format to be determined dynamically. The **ContourPlot** object uses this facility to allow you to base the formatting of all labels--in particular the assumed rightmost significant digit--on the dataset element with the maximum absolute value. The resource *cnMaxDataValueFormat* specifies a format for the dataset element with the maximum absolute value. The value and the format used together determine the specification of each dynamically specified format option for all label types.

The **ContourPlot** object recognizes a number of special substitution strings that are replaced on output with certain key values related to the dataset and the contour levels. How or whether a particular substitution string is recognized depends on the label type as follows:

CIU
: The contour interval used (value of *cnLevelSpacingF*); informational label only.

CMN
: The minimum contour level (first element of *cnLevels*); informational label only.

CMX
: The maximum contour level (last element of *cnLevels*); informational label only.

SFU
: The scale factor used (value of *cnLabelScaleFactorF*); informational label only.

ZMN
: The minimum data value (value of **ScalarField** resource *sfDataMinV*); informational label only.

ZMX
: The maximum data value (value of **ScalarField** resource *sfDataMaxV*); informational label only.

ZDV
: The current low, high, and constant field value for low, high and constant field labels, respectively.

Note that line labels do not accept any substitution strings; by default the *cnLineLabelStrings* array is filled with strings representing the value of each contour level.

Font attributes

These resources set font attributes for all the label types:

cn...LabelFontHeightF
: Sets the height of the label text.

cn...LabelFont
: Sets the font used for the label text.

cn...LabelFontColor(s)
: Sets the color of the label text. Line labels use both the plural and singular form for array and scalar versions of the resource.

cn...LabelFontAspectF
: Sets the height-to-width ratio of the characters used for the label text.

cn...LabelFontThicknessF
: Sets the thickness of the lines used to draw fonts that are stroked rather than filled for the label text.

cn...LabelFontQuality
: Sets one of three qualities of fonts to use for the label text.

cn...LabelConstantSpacingF
: Determines whether the characters used for label text should have constant spacing, and if so, determines the spacing to use.

cn...LabelAngleF
: Sets the angle at which the label text is drawn.

Note that text height and angle for low labels cannot be set differently from high label text height and angle. Currently the resources *cnLowLabelFontHeightF* and *cnLowLabelAngleF* are ignored.

If the resource *cnMonoLineLabelFontColor* is set True, **ContourPlot** uses the scalar resource *cnLineLabelFontColor* for all line labels; otherwise it uses the array resource, *cnLineLabelFontColors*. Also the resource *cnLineLabelAngleF* is ignored if *cnLineLabelPlacementMode* is set to `Constant`.

Backgrounds and perimeter outline attributes

These resources set background and perimeter outline attributes for all the label types:

cn...LabelBackgroundColor
    Sets the background color for the area inside the label perimeter.
cn...LabelPerim
    Turns the label perimeter outline on and off.
cn...LabelPerimSpaceF
    Determines the amount of space between the edges of the label text and the label perimeter.
cn...LabelPerimColor
    Sets the color used for the label perimeter line.
cn...LabelPerimThicknessF
    Sets the thickness of the label perimeter line.

Note that low label perimeter space and background color cannot be set differently from high label perimeter space and background color. Currently the resources *cnLowLabelPerimSpaceF* and *cnLowLabelBackgroundColor* are ignored. Also note that all background and perimeter resources for line labels are ignored when *cnLineLabelPlacementMode* is set to `Constant`.

Location and density attributes

You can position the informational and constant field labels using resources that follow the PlotManager Location Control Model. These are:

cn...LabelZone
    Sets the overlay zone of the label.
cn...LabelSide
    Sets the side on which the label will be located in the plot viewport.
cn...LabelJust
    Sets the justification mode used to position the label.
cn...LabelParallelPosF
    Sets the offset from the zone origin parallel to the viewport side.
cn...LabelOrthogonalPosF
    Sets the offset from the zone origin orthogonal to the viewport side.

There are three ways to position line labels using the resource *cnLineLabelPlacementMode*. The `Constant` mode is the most basic: the line label is rendered as a part of the dash pattern. This implies that the label density can be controlled by setting the *cnLineDashSegLenF* resource, which controls the length of the dash pattern.

For the other two methods, `Randomized` and `Computed`, the labels are drawn separately from the lines. As yet, no native `ContourPlot` resources have been implemented for controlling label density for these modes. However, you can gain this capability by setting low level control parameters using the *cnConpackParams* resource.

Also you can control in a limited way the location of high and low labels with the resource *cnHighLowLabelOverlapMode*. Again there are no native resources for adjusting the label density, but control is possible using *cnConpackParams*.

Special areas and boundaries

There are several boundaries within the viewport of a **ContourPlot** that separate regions where contours appear from regions where they do not: the boundaries of missing value areas, the data grid boundaries, and the boundaries of regions where the data coordinates map to an out-of-range value. Missing value areas lie in those portions of the data grid whose elements are set to the special signal missing value as set by the **ScalarField** object's *sfMissingValueV* resource. You can outline the boundaries of missing value areas and also fill the interior of these areas. On the other hand, you can only draw boundary lines of the data grid and the out-of-range regions.

You control the line attributes of the boundaries with the following resources. The ellipsis marks (...) in the resource names below stand for the names of the boundary types: *MissingValue*, *Grid*, or *OutOfRange*.

cn...BoundPerim
: Turns the perimeter line on and off.

cn...BoundPerimThicknessF
: Sets the thickness of the line used to draw the perimeter.

cn...BoundPerimDashPattern
: Sets the dash pattern of the line used to draw the perimeter.

cn...BoundPerimColor
: Sets the color of the line used to draw the perimeter.

The **ContourPlot** object sets the dash pattern segment length for these special lines with the same resource used for the regular contour lines: *cnLineDashSegLenF*.

Missing values areas have in addition the resources *cnMissingValFillColor*, *cnMissingValFillPattern*, and *cnMissingValFillScaleF* for controlling the fill attributes of the missing value areas. If the raster mode of contour filling is enabled, only the resource *cnMissingValFillColor* has any effect.

Contour line rendering control

There are several resources you can use to control the way **ContourPlot** draws the contour lines. The resource *cnSmoothingOn* controls whether the contour lines are smoothed (using cubic splines under tension). *cnSmoothingTensionF* and *cnSmoothingDistanceF* control parameters of the smoothing algorithm. The resource *cnMaxPointDistanceF* controls the accuracy with which contour lines near discontinuous points in the mapping are rendered.

Support functions

The **ContourPlot** object does not define any support functions, but inherits all the support functions available to its superclass.

Status notes for Release 4.1

1. The support for irregular transformations is at a transitional stage. Eventually, **ContourPlot** will be able to perform transformations from irregular coordinates to linear and logarithmic coordinates without using the overlay mechanism. This will eliminate the need for a switch to the index coordinate system.

3. Under certain conditions when part of the data space is outside the viewport, the Computed line label placement mode seems to fail, and the plot appears without any line labels at all. This problem requires further investigation.

4. Large values of *cnSmoothingTensionF* above 10.0 or so often cause the **ContourPlot** object to improperly project lines out of the plotter frame. This needs to be examined more closely. Also note that when lines cross, contour fill often 'bleeds' out of its boundaries.

5. Since the **ContourPlot** object creates the informational label using an Annotation object managed by the PlotManager object, Conpack has no way of knowing the position of the Informational label. Therefore, there is no way to support the option of omitting high and low labels that overlap the informational label. For the time being at least, the type *NhlTcnHighLowLabelOverlapMode* maps directly to the Conpack internal parameter, 'HLO'. Therefore, numerically speaking, this type effectively has two values for each functionally unique enumerated value.

See also

- NhlCreate function
- NhlDestroy function
- NhlSetValues function
- NhlGetValues function
- NhlAddData function
- ScalarField object
- PlotManager object
- Title object
- TickMark object
- LabelBar object
- Legend object
- AnnoManager object
- DataComm object
- Transform object
- View object
- Base object

Copyright

Copyright 1987-1999 University Corporation for Atmospheric Research
The use of this Software is governed by a License Agreement.

NCAR Graphics is a registered trademark of the University Corporation for Atmospheric Research.

CoordArrTable class

The **CoordArrTable** class provides a way to describe X/Y coordinate data to the HLU library.

Synopsis

```
Header file:             ncarg/hlu/CoordArrTable.h
Class name:              coordArrTableClass
Class pointer:           NhlcoordArrTableClass
Fortran class function:  <Not referenceable>
Superclass:              DataItem
Composite classes:       <None>
```

Note: This object is only available from "C". Fortran programmers should use the **CoordArrays** object.

Resources

Local resources

CoordArrTable resource set

NAME / CLASS	TYPE / DEFAULT	ACCESS
ctCopyTables diCopyData	NhlTBoolean True	RCSG
ctXTable CtXTable	NhlTPointerGenArray NULL	RCSG
ctXTableLengths CtXTableLengths	NhlTIntegerGenArray NULL	RCSG
ctXTableType CtXTableType	NhlTString NULL	RCSG
ctXElementSize CtXElementSize	NhlTInteger <dynamic>	RCSG

ctXMissingV	diMissingValue	NhlTVariable	NULL	RCSG
ctXMaxV	CtXMaxV	NhlTVariable	<dynamic>	RCSG
ctXMinV	CtXMinV	NhlTVariable	<dynamic>	RCSG
ctYTable	CtYTable	NhlTPointerGenArray	NULL	RCSG
ctYTableLengths	CtYTableLengths	NhlTIntegerGenArray	NULL	RCSG
ctYTableType	CtYTableType	NhlTString	NULL	RCSG
ctYElementSize	CtYElementSize	NhlTInteger	<dynamic>	RCSG
ctYMissingV	diMissingValue	NhlTVariable	NULL	RCSG
ctYMaxV	CtYMaxV	NhlTVariable	<dynamic>	RCSG
ctYMinV	CtYMinV	NhlTVariable	<dynamic>	RCSG

Superclass resources

There are no Superclass resources.

Description

The **CoordArrTable** object is used to describe X/Y coordinate data to the HLU library. It is usually used in combination with the **XyPlot** object.

The **CoordArrTable** object uses two "tables" to do this: the ctXTable and the ctYTable. These "table" resources are actually arrays of arrays. The main array points to something of `type **var` where `type` is specified by the *ctXTableType* and *ctYTableType* resources. The easiest way to understand this is to think about the table as an array of vectors. Therefore, the "type" for the *ctXTable* and *ctYTable* NhlGenArray is NhlPointer, because each element of the NhlGenArray array is actually a pointer to a vector. Each element of each vector is of type *ctXTableType*, and *ctYTableType* respectively. The *ctXTableType* and *ctYTableType* resources must be set for their respecive "Table" resource.

The Min and Max resources are computed if they are not set explicitly. If the *ctXTable* or *ctYTable* resources are reset to new values, the Min and Max values are recomputed unless the Min and Max values have been explicitly set by the user. If these resources are ever set, the programmer/user is taking responsibility for updating them if they need to be updated.

Support functions

There are no support functions for the **CoordArrTable** object, although all the support functions that are available for the **DataItem** class are available. Additionally, some of the **DataComm** functions are used with **DataItem** objects.

See also

- NhlAddData function
- NhlRemoveData function
- DataComm object
- DataItem object

Copyright

Copyright 1987-1999 University Corporation for Atmospheric Research
The use of this Software is governed by a License Agreement.

NCAR Graphics is a registered trademark of the University Corporation for Atmospheric Research.

CoordArrays class

The **CoordArrays** class provides a way to describe X/Y coordinate data to the HLU library.

Synopsis

```
Header file:             ncarg/hlu/CoordArrays.h
Class name:              coordArraysClass
Class pointer:           NhlcoordArraysClass
Fortran class function: NHLFCOORDARRAYSCLASS
Superclass:              DataItem
Composite classes:       <None>
```

Class-defined types

```
Type name:               NhlTcaCastMode
Definition:
typedef enum _NhlcaCastMode {
        NhlSINGLEVECTOR         = 1,
        NhlMULTIPLEVECTORS      = 2,
        NhlSPLITVECTORS         = 3
```

```
} NhlcaCastMode;
```

Resources

Local resources

CoordArrays resource set

NAME	CLASS	TYPE	DEFAULT	ACCESS
caCopyArrays	diCopyData	NhlTBoolean	True	RCSG
caXArray	CaXArray	NhlTGenArray	NULL	RCSG
caXCast	CaCast	NhlTcaCastMode	<Dynamic>	RCSG
caXMissingV	diMissingValue	NhlTVariable	NULL	RCSG
caXMaxV	CaXMaxV	NhlTVariable	NULL	RCSG
caXMinV	CaXMinV	NhlTVariable	NULL	RCSG
caYArray	CaYArray	NhlTGenArray	NULL	RCSG
caYCast	CaCast	NhlTcaCastMode	<Dynamic>	RCSG
caYMissingV	diMissingValue	NhlTVariable	NULL	RCSG
caYMaxV	CaYMaxV	NhlTVariable	NULL	RCSG
caYMinV	CaYMinV	NhlTVariable	NULL	RCSG

Superclass resources

There are no Superclass resources.

Description

The **CoordArrays** object is used to describe X/Y coordinate data to the HLU library. It is usually used in combination with the **XyPlot** object.

The **CoordArrays** object uses two arrays to do this: the *caXArray* and the *caYArray*. The *caXCast* and *caYCast* resources are used to determine which dimension of the *caXArray* and *caYArray* should be recognized as vectors, and which dimension should be element of the vectors.

The Min and Max resources are computed if they are not set explicitly. If the *caXArray* or *caYArray* resources are reset to new values, the Min and Max values are recomputed unless the Min and Max values have been explicitly set by the user. If these resources are ever set, the programmer/user is taking responsibility for updating them if they need to be updated.

Support functions

There are no support functions for the **CoordArrays** object, although all the support functions that are available for the **DataItem** class are available. Additionally, some of the **DataComm** functions are used with **DataItem** objects.

See also

- NhlAddData function
- NhlRemoveData function
- DataComm object
- DataItem object

Copyright

Copyright 1987-1999 University Corporation for Atmospheric Research
The use of this Software is governed by a License Agreement.

NCAR Graphics is a registered trademark of the University Corporation for Atmospheric Research.

DataComm class

The **DataComm** class provides the functionality for its subclasses to communicate with **DataItem** objects.

Synopsis

```
Header file:             ncarg/hlu/DataComm.h
Class name:              dataCommClass
Class pointer:           <Not referenceable>
Fortran class function:  <Not referenceable>
Superclass:              Transform
Composite classes:       <None>
```

Class-defined types

```
Type name:               NhlTLevelSelectionMode
Definition:
typedef enum _NhlLevelSelectionMode {
        NhlAUTOMATICLEVELS      = 0,    /* "AutomaticLevels"    */
        NhlMANUALLEVELS         = 1,    /* "ManualLevels"       */
        NhlEXPLICITLEVELS       = 2,    /* "ExplicitLevels"     */
        NhlEQUALSPACEDLEVELS    = 3     /* "EqualSpacedLevels"  */
} NhlLevelSelectionMode;

Type name:               NhlTScalingMode
Definition:
typedef enum _NhlScalingMode {
        NhlSCALEFACTOR          = 0,    /* "ScaleFactor"        */
        NhlCONFINETORANGE       = 1,    /* "ConfineToRange"     */
        NhlTRIMZEROS            = 2,    /* "TrimZeros"          */
        NhlMAXSIGDIGITSLEFT     = 3,    /* "MaxSigDigitsLeft"   */
        NhlALLINTEGERS          = 4     /* "AllIntegers"        */
} NhlScalingMode;
```

Resources

Local resources

```
+---------------------------------------------------------------+
|                        DataComm resource set                  |
|---------------------------------------------------------------|
| NAME                           TYPE                   ACCESS  |
|       CLASS                            DEFAULT               |
|===============================================================|
| dcDelayCompute                 NhlTBoolean            RCSG    |
|       DcDelayCompute                   False                 |
+---------------------------------------------------------------+
```

Superclass resources

All the **Transform** object resources can be set for **DataComm** objects.

Description

The **DataComm** class is used as a building block for all the graphical objects that visualize data. It provides the **NhlAddData**, **NhlRemoveData**, and **NhlUpdateData** functions.

The *dcDelayCompute* resource is used to indicate when the object should update itself. By default, if a user modifies a **DataItem** object that has already been associated with a *data resource* in a **DataComm** subclass, then the **DataComm** subclass is notified immediately, and actually updates its data extents at the time the **DataItem** is modified. If the *dcDelayCompute* resource is True, then the **DataComm** object just notes that the **DataItem** has been changed; it doesn't update its data extents. The **DataComm** class will automatically update its data extents based on changes when the **Draw** function is called on it, or if the user calls the **UpdateData** function on it.

Support functions

The following support functions are defined for objects of type **DataComm**:

NhlAddData
: The **NhlAddData** function is used to associate a **DataItem** with a data resource.

NhlRemoveData
: The **NhlRemoveData** function is used to unassociate a **DataItem** with a data resource.

NhlUpdateData
: The **NhlUpdateData** function is used to tell a **DataComm** object to update its internal state with the changes of any **DataItem** objects that are associated with its data resources.

NhlIsDataComm
: This function just determines if a given object id identifies a **DataComm** class object.

See also

- NhlAddData function
- NhlRemoveData function
- NhlUpdateData function
- DataItem object

Copyright

Copyright 1987-1999 University Corporation for Atmospheric Research
The use of this Software is governed by a License Agreement.

NCAR Graphics is a registered trademark of the University Corporation for Atmospheric Research.

DataItem class

The **DataItem** class provides a way to describe data to the HLU library.

Synopsis

```
Header file:             ncarg/hlu/DataItem.h
Class name:              dataItemClass
Class pointer:           <Not referenceable>
Fortran class function: <Not referenceable>
Superclass:              Base
Composite classes:       <None>
```

Resources

Local resources

The **DataItem** class doesn't define any public resources.

Superclass resources

There are no Superclass resources.

Description

The **DataItem** class is used as a building block for all the objects that describe data to the HLU library. There are no actual data description resources or anything in this class accessible to the application programmer. This class is the basis for all the classes that do describe data. It understands how to communicate with the **DataComm** class so that the actual data description objects can share that functionality.

Support functions

Some of the support functions for the **DataComm** class are intended to be used with **DataItem** class objects. Additionally, the **DataItem** class defines the following function:

NhlIsDataItem
   This function just determines if a given object id identifies a **DataItem** class object.

See also

- NhlAddData function

- NhlRemoveDatafunction
- DataComm object

Copyright

Copyright 1987-1999 University Corporation for Atmospheric Research
The use of this Software is governed by a License Agreement.

NCAR Graphics is a registered trademark of the University Corporation for Atmospheric Research.

DataSpec class

The **DataSpec** class provides a way to describe data- specific plot attributes.

Synopsis

```
Header file:             <None>
Class name:              dataSpecClass
Class pointer:           <Not referenceable>
Fortran class function:  <Not referenceable>
Superclass:              Base
Composite classes:       <None>
```

Resources

Local resources

The **DataSpec** class doesn't define any public resources.

Superclass resources

There are no Superclass resources.

Description

The **DataSpec** class is used as a building block for all the *data-specific* classes. Currently, **XyDataSpec** is the only **DataSpec** subclass that is used.

Quite often, there are plot attributes that are specific to the data being visualized. For example, when you use **XyPlot** to visualize coordinate data, the line labels that are drawn with each line are specified using a *data-specific* resource. Since it is possible to add multiple **DataItem**s to the *xyCoordData* resource, it is necessary to be able to set these *data-specific* resources for each **DataItem** that gets added to the *xyCoordData* resource.

Support functions

NhlIsDataSpec
This function just determines if a given object id identifies a **DataSpec** class object.

See also

- DataItem object
- DataComm object
- XyPlot object

Copyright

Copyright 1987-1999 University Corporation for Atmospheric Research
The use of this Software is governed by a License Agreement.

NCAR Graphics is a registered trademark of the University Corporation for Atmospheric Research.

Error class

The **Error** class provides configuration options for the error reporting module of the HLU library.

Synopsis

```
Header file:             ncarg/hlu/hlu.h
Class name:              errorClass
Class pointer:           <Not referenceable>
Fortran class function:  <Not referenceable>
Superclass:              Base
Composite classes:       <None>

Name:                    Error
```

Note: Only one error object is created for an hlu program, and it is always named *Error*.

Class Defined Types

```
Type name:              NhlTErrorTypes
Definition:
typedef enum _NhlErrType{
        NhlFATAL        = -4,   /* "FATAL"      */
        NhlWARNING      = -3,   /* "WARNING"    */
        NhlINFO         = -2,   /* "INFO"       */
        NhlNOERROR      = -1    /* "NOERROR"    */
} NhlErrorTypes;
```

Resources

Local resources

```
+---------------------------------------------------------------+
|                       Error resource set                      |
|---------------------------------------------------------------|
| NAME                          TYPE                    ACCESS  |
|       CLASS                           DEFAULT                 |
|===============================================================|
| errBuffer                     NhlTBoolean             RSG     |
|       ErrBuffer                       False                   |
|---------------------------------------------------------------|
| errLevel                      NhlTErrorTypes          RSG     |
|       ErrLevel                        NhlWARNING              |
|---------------------------------------------------------------|
| errPrint                      NhlTBoolean             RSG     |
|       ErrPrint                        True                    |
|---------------------------------------------------------------|
| errFileName                   NhlTString              RSG     |
|       ErrFileName                     "stderr"                |
|---------------------------------------------------------------|
| *errFilePtr                   NhlTPointer             SG      |
|       ErrFilePtr                      NULL                    |
|---------------------------------------------------------------|
| **errUnitNumber               NhlTInteger             RSG     |
|       ErrUnitNumber                   74                      |
+---------------------------------------------------------------+
```

* *errFilePtr* is only valid in "C" mode.
** *errUnitNumber* is only valid in "Fortran" mode.

Description

This object is used to configure the error-reporting module of the HLU library. It works in conjunction with the Error functions.

The Error object works in two modes: C mode and Fortran mode. Most of the resources work the same in both modes. The *errBuffer*, *errLevel*, and *errPrint* resources are not affected by the language mode.

The language mode that the **Error** object uses is determined by how the HLU library is initialized. If the library is initialized using the Fortran initialization function **NHLFINITIALIZE**, or **NHLFOPEN**, then the **Error** object uses Fortran mode; otherwise it uses C mode.

C mode

When in C mode, the **Error** object disables the *errUnitNumber* resource and uses the *errFilePtr* resource. If the *errFilePtr* resource is used, the *errFileName* resource is set to a NULL string.

Fortran mode

When in Fortran mode, the **Error** object disables the *errFilePtr* resource, and uses the *errUnitNumber* resource. If the *errUnitNumber* resource is set to a currently open unit, then the *errFileName* resource is set to NULL. If the unit is not currently open, then the **Error** object will attempt to open it with the name specified by the *errFileName* resource. If the *errUnitNumber* is not set, and the *errFileName* is set to either "stdout" or "stderr", then the appropriate unit numbers are used in place of the default. These are usually unit 6 and unit 0, respectively.

Support functions

The error-reporting support functions are described in the Error functions reference page.

See also

- NhlSetValues
- Error functions

Copyright

Copyright 1987-1999 University Corporation for Atmospheric Research
The use of this Software is governed by a License Agreement.

NCAR Graphics is a registered trademark of the University Corporation for Atmospheric Research.

GraphicStyle class

The **GraphicStyle** class provides grouped attributes for drawing primitives.

Synopsis

Header file:	ncarg/hlu/GraphicStyle.h
Class name:	graphicStyleClass
Class pointer:	NhlgraphicStyleClass
Fortran class function:	NHLFGRAPHICSTYLECLASS
Superclass:	Style
Composite classes:	<*None*>

Resources

Local resources

GraphicStyle Resource Set

NAME	CLASS	TYPE	DEFAULT	ACCESS
gsClipOn	GsClipOn	NhlTBoolean	True	RCSG
gsLineDashPattern	LineDashPattern	NhlTDashIndex	0	RCSG
gsLineDashSegLenF	LineDashSegLenF	NhlTFloat	0.15	RCSG
gsLineColor	LineColor	NhlTColorIndex	"Foreground"	RCSG
gsLineThicknessF	LineThicknessF	NhlTFloat	1.0	RCSG
gsLineLabelString	LineLabelString	NhlTString	<dynamic>	RCSG
gsLineLabelFont	Font	NhlTFont	"pwritx"	RCSG
gsLineLabelFontColor	FontColor	NhlTColorIndex	"Foreground"	RCSG
gsLineLabelFontHeightF	FontHeightF	NhlTFloat	0.0125	RCSG
gsLineLabelFontAspectF	FontAspectF	NhlTFloat	1.3125	RCSG
gsLineLabelFontThicknessF	FontThicknessF	NhlTFloat	1.0	RCSG
gsLineLabelFontQuality	FontQuality	NhlTFontQuality	"High"	RCSG

gsLineLabelConstantSpacingF	NhlTFloat	RCSG
TextConstantSpacingF	0.0	
gsLineLabelFuncCode	NhlTCharacter	RCSG
TextFuncCode	:	
gsFillIndex	NhlTFillIndex	RCSG
FillIndex	0	
gsFillColor	NhlTColorIndex	RCSG
FillColor	"Foreground"	
gsFillBackgroundColor	NhlTColorIndex	RCSG
FillBackgroundColor	"Transparent"	
gsFillScaleF	NhlTFloat	RCSG
FillScaleF	1.0	
gsFillLineThicknessF	NhlTFloat	RCSG
FillLineThicknessF	1.0	
gsEdgesOn	NhlTBoolean	RCSG
EdgesOn	False	
gsEdgeDashPattern	NhlTDashIndex	RCSG
EdgeDashPattern	0	
gsEdgeThicknessF	NhlTFloat	RCSG
EdgeThicknessF	1.0	
gsEdgeDashSegLenF	NhlTFloat	RCSG
EdgeDashSegLenF	.15	
gsEdgeColor	NhlTColorIndex	RCSG
EdgeColor	"Foreground"	
gsMarkerIndex	NhlTMarkerIndex	RCSG
MarkerIndex	"asterisk"	
gsMarkerColor	NhlTColorIndex	RCSG
MarkerColor	"Foreground"	
gsMarkerSizeF	NhlTFloat	RCSG
MarkerSizeF	0.007	
gsMarkerThicknessF	NhlTFloat	RCSG
MarkerThicknessF	1.0	
gsTextAngleF	NhlTFloat	RCSG
TextAngleF	0.0	
gsFont	NhlTFont	RCSG
Font	"pwritx"	
gsTextJustification	NhlTJustification	RCSG
TextJustification	"CenterCenter"	
gsFontQuality	NhlTFontQuality	RCSG
FontQuality	"High"	
gsFontColor	NhlTColorIndex	RCSG

FontColor	"Foreground"	
gsFontHeightF FontHeightF	NhlTFloat 0.015	RCSG
gsFontAspectF FontAspectF	NhlTFloat 1.3125	RCSG
gsFontThicknessF FontThicknessF	NhlTFloat 1.0	RCSG
gsTextConstantSpacingF TextConstantSpacingF	NhlTFloat 0.0	RCSG
gsTextDirection TextDirection	NhlTTextDirection "Across"	RCSG
gsTextFuncCode TextFuncCode	NhlTCharacter :	RCSG

Composite resources

The **GraphicStyle** object class has no composite class objects.

Superclass resources

The superclasses of the **GraphicStyle** class do not define any resources.

Description

The **GraphicStyle** class defines a set of resources for specifying the attributes of line, fill, marker, and text drawing primitives. Currently, **GraphicStyle** objects are used to control the appearance of graphics primitives drawn using the following immediate mode drawing functions:

- NhlDataPolygon
- NhlDataPolyline
- NhlDataPolymarker
- NhlNDCPolygon
- NhlNDCPolyline
- NhlNDCPolymarker

In the future, use of **GraphicStyle** and related classes may become more prevalent throughout the HLU library. Note that at present the **GraphicStyle** text attribute resources have no function.

All the immediate mode drawing functions take an instance of the **GraphicStyle** class as an argument. You have several options for specifying this argument. You may create a **graphicstyle** just as you would any other HLU object, set its resources as desired, and then use its id in the argument list to any of these functions.

As an alternative, you may specify the argument using the default **graphicstyle** that each **workstation** creates automatically when it is initialized. The easiest way to do this is simply to use `NhlNULLOBJID` (0) as the argument. This value is recognized by the **workstation** as a shorthand signal telling it to use its default **graphicstyle**.

On the other hand, you may retrieve the actual id of the **workstation**'s default **graphicstyle** by getting the value of the read-only **Workstation** resource *wkDefGraphicStyleId*. Once you have retrieved this id, you may set the resources of the default **graphicstyle** just as if you had created it yourself.

Note that setting **GraphicStyle** *class* resources in a resource file affects the **Workstation** default **graphicstyle** the same as it affects **graphicstyle**s you create on your own.

Support functions

The **GraphicStyle** object does not define any support functions, but inherits all the support functions available to its superclass.

Status

See also

- NhlCreate function
- NhlDestroy function
- NhlSetValues function
- NhlGetValues function
- Style object
- Base object

Copyright

Copyright 1987-1999 University Corporation for Atmospheric Research
The use of this Software is governed by a License Agreement.

NCAR Graphics is a registered trademark of the University Corporation for Atmospheric Research.

IrregularPlot class

The **IrregularPlot** class draws immediate mode lines and may be used to map overlay plots into an irregular rectangular coordinate space.

Synopsis

```
Header file:            ncarg/hlu/IrregularPlot.h
Class name:             irregularPlotClass
Class pointer:          NhlirregularPlotClass
Fortran class function: NHLFIRREGULARPLOTCLASS
Superclass:             Transform
Composite classes:      IrregularTransformation,PlotManager
```

Resources

Local resources

The **IrregularPlot** object has no local resources.

Composite resources

Transformation resources

Transformation class resources specify the extent and direction of the data coordinate system. As the superclass of **IrregularTransformation**, you can access all **Transformation** resources.

IrregularTransformation resources

You can access all resources specific to the **IrregularTransformation** subclass. These resources specify the type of transformation for each axis, the coordinate arrays if required, and other associated parameters.

PlotManager resources

Assuming *tfPlotManagerOn* is True when an **IrregularPlot** object is created, you can access all **PlotManager** resources. You can also access resources for any of the **PlotManager's** composite class members. However, the **PlotManager** object modifies the access and behavior of some of the resources belonging to these objects, as follows:

- LabelBar resources as modified by PlotManager
- Legend resources as modified by PlotManager
- TickMark resources as modified by PlotManager
- Title resources as modified by PlotManager

Superclass resources

You can access all resources defined by the superclasses of the **IrregularPlot** object class, including:

- **Transform** resources
- **View** resources

Description

Unlike most **Transform** subclasses, **IrregularPlot** has no specialized plot elements and no **Draw** method of its own. Its primary function is to act as a base plot over which other plot objects may be overlaid in order to transform their data space into an irregular coordinate space. Like other members of the **Transform** class, **IrregularPlot** can also function as a "canvas" for drawing graphics primitives, either in data space or in NDC space.

You can transform a plot object into an irregularly gridded coordinate space by overlaying it on an **irregularplot**. The irregularly gridded space may encompass the data space of any number of plot objects with partially or completely disjoint data coordinate extents. You define the irregular mapping, as well as the coordinate axis extents and directions, using the resources of the **IrregularTransformation** object belonging to the **IrregularPlot**. Note that **IrregularPlot** is the only class that allows unrestricted access to all the resources of its **IrregularTransformation** object.

Assuming *tfPlotManagerOn* is True when you create an **IrregularPlot** object, the **IrregularPlot** object has the same plot management capabilities as any other plot object. The **IrregularPlot** object does not intercept any resources belonging to its **PlotManager** or the **PlotManager's** composite class members. Therefore, you access these resources just as described for the **PlotManager** object itself.

Support functions

The **IrregularPlot** object does not define any support functions, but inherits all the support functions available to its superclasses.

Status

1. The status of the immediate mode drawing functions (DataPolyline) is not yet resolved.

2. The resources of the **IrregularTransformation** used to set the irregular coordinate transformation need to be updated to use GenArrays.

See also

- NhlCreate function
- NhlDestroy function
- NhlSetValues function

- NhlGetValues function
- IrregularTransformation object
- PlotManager object
- Transform object
- View object
- Base object

Copyright

Copyright 1987-1999 University Corporation for Atmospheric Research
The use of this Software is governed by a License Agreement.

NCAR Graphics is a registered trademark of the University Corporation for Atmospheric Research.

IrregularTransformation class

The **IrregularTransformation** class manages forward and reverse transformations in an irregular rectangular coordinate space.

Synopsis

```
Header file:             ncarg/hlu/IrregularTransformation.h
Class name:              irregularTransformationClass
Class pointer:           <Not referenceable>
Fortran class function:  <Not referenceable>
Superclass:              Transformation
Composite classes:       <None>
```

Class-defined types

```
Type name:               NhlTAxisType
Definition:
typedef enum _NhlAxisType {
        NhlIRREGULARAXIS        = 0,    /* "IrregularAxis"      */
        NhlLINEARAXIS           = 1,    /* "LinearAxis"         */
        NhlLOGAXIS              = 2     /* "LogAxis"            */
} NhlAxisType;
```

Resources

Local resources

IrregularTransformation resources are accessible only through **Transform** class objects that use the **IrregularTransformation** to manage transformations to and from data space. These include:

- **ContourPlot** object
- **LogLinPlot** object
- **StreamlinePlot** object
- **VectorPlot** object
- **XyPlot** object

IrregularTransformation Resource Set

NAME	CLASS	TYPE	DEFAULT	ACCESS
trXAxisType	TrXAxisType	NhlTAxisType	"LinearAxis"	RCSG
trXCoordPoints	TrXCoordPoints	NhlTFloatGenArray	NULL	RCSG
trXInterPoints	TrXInterPoints	NhlTFloatGenArray	NULL	RCSG
trXTensionF	TrXTensionF	NhlTFloat	2.0	RCSG
trXSamples	TrXSamples	NhlTInteger	9	RCSG
trYAxisType	TrYAxisType	NhlTAxisType	"LinearAxis"	RCSG
trYCoordPoints	TrYCoordPoints	NhlTFloatGenArray	NULL	RCSG
trYInterPoints	TrYInterPoints	NhlTFloatGenArray	NULL	RCSG
trYTensionF	TrYTensionF	NhlTFloat	2.0	RCSG
trYSamples	TrYSamples	NhlTInteger	9	RCSG

Composite resources

The **IrregularTransformation** class has no composite class objects.

Superclass resources

You can set all resources defined by the superclasses of the **IrregularTransformation** object class, including:

- **Transformation** resources

Description

The **IrregularTransformation** provides transformations to and from a rectangular irregularly gridded coordinate space for **Transform** class objects. The transformations are defined using continuous spline approximations of externally supplied arrays containing monotonically increasing or decreasing coordinate values. Either axis may also be specified as linear or logarithmic. Therefore, **IrregularTransformation** provides a superset of the functionality available to the **LogLinTransformation** class.

You do not create objects of the **IrregularTransformation** class directly. **Transform** objects that use a **IrregularTransformation** child, however, generally provide controlled access to some or all of its resources. Its **Transformation** superclass provides the basic resources for specifying the coordinate extent and direction. If you specify an axis as irregular and supply a monotonic coordinate array, **IrregularTransformation** restricts the values allowed for the extent resources to the range implied by the first and last elements of the array.

Although the **IrregularTransformation** provides resources for setting the coordinate array values (*trXCoordPoints* and *trYCoordPoints*), most **Transform** objects do not allow direct access to them. Usually these arrays are set automatically based on values contained in a currently associated data object. For example, **ContourPlot** sets them using the **ScalarField** resources *sfXArray* and *sfYArray*. **VectorPlot** and **StreamlinePlot** use the corresponding **VectorField** resources *vfXArray* and *vfYArray*. **XyPlot** has explicit array resources of its own: *XIrregularPoints* and *YIrregularPoints*. On the other hand, **IrregularPlot**, which has no associated data objects, allows unrestricted access to all resources belonging to the **IrregularTransformation** class.

Support functions

There are no special support functions defined for the **IrregularTransformation** class or its superclass.

Status

2. Irregular transformations may fail if the values of *trXCoordPoints* or *trYCoordPoints* represent samples of a transformation that is "too" irregular. What constitutes a "too" irregular transformation requires further investigation.

See also

- IrregularPlot object

- PlotManager object
- ContourPlot object
- StreamlinePlot object
- VectorPlot object
- XyPlot object
- Transformation object
- Obj object

Copyright

Copyright 1987-1999 University Corporation for Atmospheric Research
The use of this Software is governed by a License Agreement.

NCAR Graphics is a registered trademark of the University Corporation for Atmospheric Research.

LabelBar class

The **LabelBar** class draws an annotation for fill styles with an optional title and/or border.

Synopsis

```
Header file:             ncarg/hlu/LabelBar.h
Class name:              labelBarClass
Class pointer:           NhllabelBarClass
Fortran class function:  NHLFLABELBARCLASS
Superclass:              View
Composite classes:       <none>
```

Class-defined types

```
Type name:               NhlTlbLabelAlignmentMode
Definition:
typedef enum _NhllbLabelAlignmentMode {
        NhlBOXCENTERS           = 0,    /* "BoxCenters"    */
        NhlINTERIOREDGES        = 1,    /* "InteriorEdges" */
        NhlEXTERNALEDGES        = 2     /* "ExternalEdges" */
} NhllbLabelAlignmentMode;

Type name:               NhlTlbBoxSizingMode
Definition:
typedef enum _NhllbBoxSizingMode {
        NhlUNIFORMSIZING        = 0,    /* "UniformSizing" */
        NhlEXPLICITSIZING       = 1     /* "ExplicitSizing" */
} NhllbBoxSizingMode;
```

Resources

Local resources

LabelBar Resource Set

NAME	CLASS	TYPE	DEFAULT	ACCESS
lbLabelBarOn	lbLabelBarOn	NhlTBoolean	True	RCSG
lbOrientation	LbOrientation	NhlTOrientation	"Vertical"	RCSG
lbJustification	LbJustification	NhlTJustification	"BottomLeft"	RCSG
lbBoxMajorExtentF	LbBoxMajorExtentF	NhlTFloat	1.0	RCSG
lbBoxMinorExtentF	LbBoxMinorExtentF	NhlTFloat	0.33	RCSG
lbBoxCount	LbBoxCount	NhlTInteger	16	RCSG
lbBoxSizing	LbBoxSizing	NhlTlbBoxSizingMode	"UniformSizing"	RCSG
lbAutoManage	LbAutoManage	NhlTBoolean	True	RCSG
lbLabelOffsetF	LbLabelOffsetF	NhlTFloat	0.1	RCSG
lbTitleOffsetF	LbTitleOffsetF	NhlTFloat	0.03	RCSG
lbLeftMarginF	LbLeftMarginF	NhlTFloat	0.05	RCSG
lbRightMarginF	LbRightMarginF	NhlTFloat	0.05	RCSG
lbBottomMarginF	LbBottomMarginF	NhlTFloat	0.05	RCSG
lbTopMarginF	LbTopMarginF	NhlTFloat	0.05	RCSG
lbMonoFillColor	LbMonoFillColor	NhlTBoolean	False	RCSG
lbFillColor	FillColor	NhlTColorIndex	"Foreground"	RCSG
lbFillColors		NhlTColorIndexGenArray		RCSG

LbFillColors	<dynamic>	
lbMonoFillPattern	NhlTBoolean	RCSG
LbMonoFillColor	False	
lbFillPattern	NhlTFillIndex	RCSG
FillPattern	"SolidFill"	
lbFillPatterns	NhlTFillIndexGenArray	RCSG
LbFillPatterns	<dynamic>	
lbMonoFillScale	NhlTBoolean	RCSG
LbMonoFillScale	True	
lbFillScaleF	NhlTFloat	RCSG
FillScaleF	1.0	
lbFillScales	NhlTFloatGenArray	RCSG
LbFillScales	NULL	
lbLabelStrings	NhlTStringGenArray	RCSG
LbLabelStrings	NULL	
lbBoxFractions	NhlTFloatGenArray	RCSG
LbBoxFractions	NULL	
lbLabelsOn	NhlTBoolean	RCSG
lbLabelsOns	True	
lbLabelPosition	NhlTPosition	RCSG
LbLabelPosition	"Right"	
lbLabelAngleF	NhlTFloat	RCSG
TextAngleF	0.0	
lbLabelAlignment	NhlTlbLabelAlignmentMode	RCSG
LbLabelAlignment	"BoxCenters"	
lbLabelDirection	NhlTTextDirection	RCSG
TextDirection	"Across"	
lbLabelJust	NhlTJustification	RCSG
TextJustification	"CenterCenter"	
lbLabelFont	NhlTFont	RCSG
Font	"pwritx"	
lbLabelFontColor	NhlTInteger	RCSG
FontColor	"Foreground"	
lbLabelFontHeightF	NhlTFloat	RCSG
FontHeightF	0.02	
lbLabelFontAspectF	NhlTFloat	RCSG
FontAspectF	1.0	
lbLabelFontThicknessF	NhlTFloat	RCSG
FontThicknessF	1.0	
lbLabelFontQuality	NhlTFontQuality	RCSG
FontQuality	"High"	

lbLabelConstantSpacingF TextConstantSpacingF	NhlTFloat 0.0		RCSG
lbLabelFuncCode TextFuncCode	NhlTCharacter ':'		RCSG
lbLabelStride LbLabelStride	NhlTInteger 1		RCSG
lbTitleString LbTitleString	NhlTString <dynamic>		RCSG
lbTitleOn lbTitleOn	NhlTBoolean True		RCSG
lbTitlePosition LbTitlePosition	NhlTPosition "Top"		RCSG
lbTitleExtentF LbTitleExtentF	NhlTFloat 0.15		RCSG
lbTitleAngleF TextAngleF	NhlTFloat 0.0		RCSG
lbTitleDirection TextDirection	NhlTTextDirection "Across"		RCSG
lbTitleFont Font	NhlTFont "pwritx"		RCSG
lbTitleFontColor FontColor	NhlTInteger "Foreground"		RCSG
lbTitleJust TextJustification	NhlTJustification "CenterCenter"		RCSG
lbTitleFontHeightF FontHeightF	NhlTFloat 0.025		RCSG
lbTitleFontAspectF FontAspectF	NhlTFloat 1.0		RCSG
lbTitleFontThicknessF FontThicknessF	NhlTFloat 1.0		RCSG
lbTitleFontQuality FontQuality	NhlTFontQuality "High"		RCSG
lbTitleConstantSpacingF TextConstantSpacingF	NhlTFloat 0.0		RCSG
lbTitleFuncCode TextFuncCode	NhlTCharacter ':'		RCSG
lbBoxLinesOn lbBoxLinesOn	NhlTBoolean True		RCSG
lbBoxLineColor LineColor	NhlTInteger "Foreground"		RCSG

lbBoxLineThicknessF LineThicknessF	NhlTFloat 1.0	RCSG
lbBoxLineDashPattern LineDashPattern	NhlTInteger 0	RCSG
lbBoxLineDashSegLenF LineDashSegLenF	NhlTFloat 0.15	RCSG
lbPerimOn EdgesOn	NhlTBoolean True	RCSG
lbPerimColor EdgeColor	NhlTInteger "Foreground"	RCSG
lbPerimFill FillPattern	NhlTInteger "HollowFill"	RCSG
lbPerimFillColor FillColor	NhlTInteger "Foreground"	RCSG
lbPerimThicknessF EdgeThicknessF	NhlTFloat 1.0	RCSG
lbPerimDashPattern EdgeDashPattern	NhlTInteger 0	RCSG
lbPerimDashSegLenF EdgeDashSegLenF	NhlTFloat 0.15	RCSG
lbFillBackground FillBackgroundColor	NhlTInteger "Transparent"	RCSG
lbFillLineThicknessF FillLineThicknessF	NhlTFloat 1.0	RCSG

Composite resources

The **LabelBar** class has no composite class objects.

Superclass resources

You can set all **View** class resources for the **LabelBar** object.

Description

The **LabelBar** object formats a series of solid and/or pattern-filled boxes along with accompanying explanatory labels into a horizontal row or vertical column. It is designed to serve as the key for an associated plot, such as a contour plot, that employs pattern- or solid-fill areas. The labels, as well as an optional title, may be sized, angled, and positioned in a variety of ways. You may create a perimeter outlining or solid-filling the background of the area occupied by the **LabelBar**. When using pattern-filled boxes, you may solid-fill the background of the fill area. You can set the attributes of each **LabelBar** box individually using a number of array resources.

LabelBar elements

The **LabelBar** object provides resources to manage the following elements:

- Position, size, and shape
- Attributes of the filled boxes
- Attributes of the labels accompanying each box
- Attributes of the title
- Attributes of the perimeter line and the background area

Managing the LabelBar position, size, and shape

Direct management

If you create a **LabelBar** directly, you set its basic location and size using the resources of its **View** superclass, *vpXF*, *vpYF*, *vpWidthF*, and *vpHeightF*. However, the **LabelBar** may adjust its size (*vpWidthF* and *vpHeightF*) to accommodate the constraints imposed by some of its own resources. The NhlTJustification resource *lbJustification* specifies the point of the **LabelBar** object that is to remain fixed when size adjustments are made. If *lbJustification* has a value other than `TopLeft`, the position (*vpXF* and *vpYF*) may also change when size adjustments occur.

The NhlTOrientation resource *lbOrientation* determines whether **LabelBar** arranges the boxes horizontally or vertically. Based on the value of this resource, **LabelBar** defines its *major axis* to be the axis parallel to the direction of orientation and the *minor axis* as the axis orthogonal to the direction of orientation.

If the boolean resource *lbAutoManage* is set True, the **LabelBar** controls the size of its elements such that it can remain as close as possible to the size specified by its **View** resources. The length of the minor axis will remain as specified by the appropriate **View** resource value (either *vpWidthF* or *vpHeightF* depending on the orientation). The major axis grows or shrinks only if the angle of the **LabelBar** labels is modified.

When *lbAutoManage* is True, you have no direct control over the size of text used in the **LabelBar**. If you are working interactively, you may find it helpful to create a basic **LabelBar** layout close to the desired size with the *lbAutoManage* mode on, then switch it off to tune the text size precisely to your taste.

You can cause the **LabelBar** as a whole to appear or disappear from the view surface by manipulating the value of the boolean resource *lbLabelBarOn*.

Management as an annotation

More typically, **LabelBar** objects are created as annotations for a plot object such as **ContourPlot**. If not disabled by the plot object class, you can automatically instantiate a **LabelBar** object simply by setting the **PlotManager** resource *pmLabelBarDisplayMode* to any value other than `NoCreate`. The **ContourPlot** object does this by default.

As an intrinsic annotation of the **PlotManager**, the **LabelBar** object's **View** class resources can no longer be accessed directly. You now must control the size and location using resources that mostly belong to the **PlotManager** class. You set the location using resources that follow the PlotManager Location Control Model, as follows:

- *pmLabelBarZone*
- *pmLabelBarSide*
- *pmLabelBarParallelPosF*
- *pmLabelBarOrthogonalPosF*
- *lbJustification*

Note that the justification resource belongs to the **LabelBar** itself, while the remaining resources belong to the **PlotManager** class. You control the size using the **PlotManager** resources *pmLabelBarWidthF* and *pmLabelBarHeightF*. These resources behave the same as *vpWidthF* and *vpHeightF* do when you are controlling the **LabelBar** directly.

The **PlotManager** class modifies the behavior of certain **LabelBar** resources. The **ContourPlot** class modifies the behavior of others. Otherwise, you can set and retrieve **LabelBar** resources instantiated with a **ContourPlot** just as you would if you created the **LabelBar** directly.

Layout of the LabelBar interior

Within the perimeter of the **LabelBar**, there is an optional title, the **LabelBar** boxes themselves, and next to each box an optional label. Surrounding these elements you can specify a margin on each side using the resources *lbLeftMarginF*, *lbRightMarginF*, *lbBottomMarginF*, *lbTopMarginF*. If there is a title, it may be placed inside the margin along any of the edges, parallel to either axis. The amount of room allotted to the title is specified using the resource *lbTitleExtentF*. You can specify a space between the title extent and the other **LabelBar** elements using the resource *lbTitleOffsetF*. Between the boxes and the labels you can also specify a space using the resource *lbLabelOffsetF*.

LabelBar boxes

You set the number of boxes using the resource, *lbBoxCount*. All the array resources belonging to the **LabelBar** are sized in relation to the number of boxes.

When *lbAutoManage* is True, the resource *lbBoxMinorExtentF* determines what fraction of the distance across the minor axis (after subtracting the margins) is occupied by the boxes. If *lbAutoManage* is False, it still specifies this same fraction initially, but may no longer after the **LabelBar** size is adjusted based on the text heights of the labels and the title.

The boxes, placed end to end, occupy the extent of the major axis that remains after subtracting:

- the margins,
- the space occupied by the title if it is enabled and placed along one of the minor axis sides,
- any space added to accommodate angled labels, and
- an amount equal to the font height of the labels if *lbLabelAlignment* is set to `ExternalEdges`.

The resource *lbBoxMajorExtentF* should be a value greater than 0.0 and less than or equal to 1.0 that specifies what fraction of the space alloted to each box the box actually occupies. If *lbBoxMajorExtentF* is 0.5, for example, each box is separated from its neighboring boxes by a space equal to the space it occupies.

If the NhlTlbBoxSizingMode resource *lbBoxSizing* is set to `ExplicitSizing`, the array resource *lbBoxFractions* sets the beginning and end of each box as a fraction of the distance along the length of the major axis reserved for the boxes. In this case the *lbBoxMajorExtentF* fractional value is applied to each box based on the size available to it.

You can control the color, the fill pattern, and the scale of the fill pattern of the boxes either individually or uniformly as a group. If the boxes are to be differentiated, you would not want to control all three resources as a group. Individual control is through the array resources, *lbFillColors* , *lbFillPatterns*, and *lbFillScales* while control as a group is through the scalar resources *lbFillColor*, *lbFillPattern*, and *lbFillScaleF*. You choose the scalar resources by setting the boolean resources *lbMonoFillColor*, *lbMonoFillPattern*, and *lbMonoFillScale* True; the array resources by setting them False.

The resource *lbFillBackground* specifies a background color for the fill of the boxes. It only has an effect when the fill pattern is not solid or hollow and the fill color is not transparent. Note that *lbFillBackground* also applies to the background of the fill pattern used to fill the perimeter area around the boxes and behind the title and the labels.

You can also control the rendering of the lines that define the perimeters of the individual boxes. You turn these lines on and off using the boolean resource *lbBoxLinesOn*. You can control the color, thickness, dash pattern, and the dash segment length of these lines using the resources *lbBoxLineColor*, *lbBoxLineThicknessF*, *lbBoxLineDashPattern*, and *lbBoxLineDashSegLenF*.

LabelBar labels

You choose whether or not to label the boxes by setting the boolean resource *lbLabelsOn* True or False. The actual label strings are set using the array resource *lbLabelStrings*.

You can place the labels on either side of the boxes or center them over the boxes using the NhlTPosition resource *lbLabelPosition*. If you set *lbLabelPosition* position to a value inappropriate for the orientation (e.g. `Bottom` when *lbOrientation* is `Vertical`), it is automatically coerced to a valid value. Using the NhlTlbLabelAlignmentMode resource *lbLabelAlignment*, you can align the labels to the centers of the boxes, to the interior edges of the boxes, or to all edges including both endpoints of the boxes. If the **LabelBar** has too many boxes for there to be a reasonably sized label for each box, you can set the *lbLabelStride* resource in order to label only every *n*th box.

The labels have a complete set of text attribute resources. When *lbAutoManage* is True, however, the resources *lbLabelFontHeightF* or *lbLabelJust* are set automatically. In this case, **LabelBar** adjusts the font height and justification such that the labels fit the available space and remain properly aligned with their boxes. If the *lbLabelAngleF* resource is varied, **LabelBar** ensures that the labels do not overlap one another. If *lbAutoManage* is set False, the **LabelBar** viewport expands if necessary to accommodate the full label text entent. However, in this mode, there are no checks to prevent the labels from overlapping, and the justification must be set manually in order for the labels to appear properly aligned if the *lbLabelAngleF* resource is set to certain ranges of values.

The LabelBar title

You turn the **LabelBar** title on and off using the boolean resource, *lbTitleOn*. You set the text of the title using the resource, *lbTitleString*. Any time you set *lbTitleString* and do not explicitly set *lbTitleOn* at the same time, *lbTitleOn* is automatically set True. You set the title's position with the NhlTPosition resource *lbTitlePosition*. The only restriction on the title's position is that it cannot be set to `Center`.

Like the labels, the title has a full set of text attribute resources. If *lbAutoManage* is True, the title is automatically sized to fit the available space as determined by the **LabelBar** viewport and the *lbTitleExtentF* resource. Attempts to set the *lbTitleFontHeightF* resource are ignored. You can influence the title's size indirectly, however. When *lbAutoManage* is set False, you can set *lbTitleFontHeightF* directly. If necessary, the **LabelBar** instance will increase the size of its viewport to accommodate the full extent of the title string.

The LabelBar perimeter and background

You can draw an outline around the perimeter of the **LabelBar** by setting the boolean resource *lbPerimOn* True. You can control all attributes of the perimeter, including line color, line thickness, line dash pattern, and the dash segment length, using the resources *lbPerimColor*, *lbPerimThicknessF*, *lbPerimDashPattern*, and *lbPerimDashSegLenF*.

You can fill the interior area defined by the perimeter (in other words, the area around the boxes and behind the title and labels), using the resources *lbPerimFill* and *lbPerimFillColor*. The resource *lbFillBackground* specifies a background color for the fill. It only has an effect when *lbPerimFill* specifies a fill pattern other than `Solid` or `HollowFill`, and the *lbPerimFillColor* is not `Transparent`. Note that *lbFillBackground* also applies to the background of the fill patterns used within the **LabelBar** boxes.

Support functions

The **LabelBar** object does not define any support functions, but it inherits all support functions available to the **View** class.

Status

The LabelBar may display incorrectly if its View width or height is modified while the LabelBar is turned off (lbLabelBarOn set False). This is likely to happen if you add a ContourPlot as an overlay to another plot while pmLabelBarDisplayMode is set to "never", then subsequently change pmLabelBarDisplayMode to "conditional" or "always". If you are going to be using the LabelBar, turn it on before changing its size; set pmLabelBarDisplayMode to "conditional" or "always" before adding a ContourPlot as an overlay.

See also

- NhlCreate function
- NhlSetValues function
- NhlGetValues function
- PlotManager object

Copyright

Copyright 1987-1999 University Corporation for Atmospheric Research
The use of this Software is governed by a License Agreement.

NCAR Graphics is a registered trademark of the University Corporation for Atmospheric Research.

Legend class

The **Legend** class draws an annotation for line and/or marker styles with an optional title and/or border.

Synopsis

```
Header file:             ncarg/hlu/Legend.h
Class name:              legendClass
Class pointer:           NhllegendClass
Fortran class function: NHLFLEGENDCLASS
Superclass:              View
Composite classes:       <none>
```

Class-defined types

```
Type name:               NhlTlgLabelAlignmentMode
Definition:
typedef enum _NhllgLabelAlignmentMode {
        NhlITEMCENTERS          = 0,     /* "ItemCenters" */
        NhlABOVEITEMS           = 1,     /* "AboveItems"  */
        NhlBELOWITEMS           = 2      /* "BelowItems"  */
} NhllgLabelAlignmentMode;

Type name:               NhlTlgItemPlacementMode
Definition:
typedef enum _NhllgItemPlacementMode {
        NhlUNIFORMPLACEMENT     = 0,     /* "UniformPlacement" */
        NhlEXPLICITPLACEMENT    = 1      /* "ExplicitPlacement" */
} NhllgItemPlacementMode;
```

Resources

Local resources

Legend Resource Set

NAME	CLASS	TYPE	DEFAULT	ACCESS
lgLegendOn	LgLegendOn	NhlTBoolean	True	RCSG
lgOrientation	LgOrientation	NhlTOrientation	"Vertical"	RCSG
lgJustification	LgJustification	NhlTJustification	"BottomLeft"	RCSG
lgBoxMajorExtentF	LgBoxMajorExtentF	NhlTFloat	0.5	RCSG
lgBoxMinorExtentF	LgBoxMinorExtentF	NhlTFloat	0.6	RCSG
lgItemCount	LgItemCount	NhlTInteger	16	RCSG
lgItemPlacement	LgItemPlacement	NhlTlgItemPlacementMode	"UniformPlacement"	RCSG
lgBoxBackground	FillBackgroundColor	NhlTColorIndex	"Transparent"	RCSG
lgAutoManage	LgAutoManage	NhlTBoolean	True	RCSG
lgLabelOffsetF	LgLabelOffsetF	NhlTFloat	0.02	RCSG
lgTitleOffsetF	LgTitleOffsetF	NhlTFloat	0.03	RCSG
lgLeftMarginF	LgLeftMarginF	NhlTFloat	0.05	RCSG
lgRightMarginF	LgRightMarginF	NhlTFloat	0.05	RCSG
lgBottomMarginF	LgBottomMarginF	NhlTFloat	0.05	RCSG
lgTopMarginF	LgTopMarginF	NhlTFloat	0.05	RCSG
lgMonoItemType	LgMonoItemType	NhlTBoolean	True	RCSG
lgItemType		NhlTMarkLineMode		RCSG

	LgItemType		"Lines"	
lgItemTypes	LgItemTypes	NhlTMarkLineModeGenArray	<dynamic>	RCSG
lgMonoDashIndex	LgMonoDashIndex	NhlTBoolean	True	RCSG
lgDashIndex	LineDashPattern	NhlTDashIndex	"SolidLine"	RCSG
lgDashIndexes	LgDashIndexes	NhlTDashIndexGenArray	<dynamic>	RCSG
lgMonoMarkerIndex	LgMonoMarkerIndex	NhlTBoolean	True	RCSG
lgMarkerIndex	MarkerIndex	NhlTMarkerIndex	"default"	RCSG
lgMarkerIndexes	LgMarkerIndexes	NhlTMarkerIndexGenArray	<dynamic>	RCSG
lgLineLabelStrings	LgLineLabelStrings	NhlTStringGenArray	<dynamic>	RCSG
lgMonoLineColor	LgMonoLineColor	NhlTBoolean	False	RCSG
lgLineColor	LineColor	NhlTColorIndex	"Foreground"	RCSG
lgLineColors	LgLineColors	NhlTColorIndexGenArray	<dynamic>	RCSG
lgMonoMarkerColor	LgMonoMarkerColor	NhlTBoolean	False	RCSG
lgMarkerColor	MarkerColor	NhlTColorIndex	"Foreground"	RCSG
lgMarkerColors	LgMarkerColors	NhlTColorIndexGenArray	<dynamic>	RCSG
lgMonoLineDashSegLen	LgMonoLineDashSegLen	NhlTBoolean	True	RCSG
lgLineDashSegLenF	LineDashSegLenF	NhlTFloat	0.15	RCSG
lgLineDashSegLens	LgLineDashSegLens	NhlTFloatGenArray	NULL	RCSG
lgMonoLineThickness	LgMonoLineThickness	NhlTBoolean	True	RCSG
lgLineThicknessF	LineThicknessF	NhlTFloat	1.0	RCSG
lgLineThicknesses	LgLineThicknesses	NhlTFloatGenArray	<dynamic>	RCSG

lgMonoMarkerThickness LgMonoMarkerThickness	NhlTBoolean True		RCSG
lgMarkerThicknessF MarkerThicknessF	NhlTFloat 1.0		RCSG
lgMarkerThicknesses LgMarkerThicknesses	NhlTFloatGenArray <dynamic>		RCSG
lgMonoLineLabelFontHeight LgMonoLineLabelFontHeight	NhlTBoolean True		RCSG
lgLineLabelFontHeightF FontHeightF	NhlTFloat 0.01		RCSG
lgLineLabelFontHeights LgLineLabelFontHeights	NhlTFloatGenArray <dynamic>		RCSG
lgMonoMarkerSize LgMonoMarkerSize	NhlTBoolean True		RCSG
lgMarkerSizeF MarkerSizeF	NhlTFloat 0.01		RCSG
lgMarkerSizes LgMarkerSizes	NhlTFloatGenArray <dynamic>		RCSG
lgLabelStrings LgLabelStrings	NhlTStringGenArray <dynamic>		RCSG
lgItemPositions LgItemPositions	NhlTFloatGenArray NULL		RCSG
lgMonoLineLabelFontColor LgMonoLineLabelFontColor	NhlTBoolean False		RCSG
lgLineLabelFontColor FontColor	NhlTColorIndex "Foreground"		RCSG
lgLineLabelFontColors LgLineLabelFontColors	NhlTColorIndexGenArray <dynamic>		RCSG
lgLineLabelsOn LgLineLabelsOn	NhlTBoolean True		RCSG
lgLineLabelFont Font	NhlTFont "pwritx"		RCSG
lgLineLabelFontAspectF FontAspectF	NhlTFloat 1.0		RCSG
lgLineLabelFontThicknessF FontThicknessF	NhlTFloat 1.0		RCSG
lgLineLabelFontQuality FontQuality	NhlTFontQuality "High"		RCSG
lgLineLabelConstantSpacingF TextConstantSpacingF	NhlTFloat 0.0		RCSG

```
lgLineLabelFuncCode             NhlTCharacter            RCSG
      TextFuncCode                      ':'
----------------------------------------------------------------
lgLabelsOn                      NhlTBoolean              RCSG
      LgLabelsOn                        True
----------------------------------------------------------------
lgLabelPosition                 NhlTPosition             RCSG
      LgLabelPosition                   "Right"
----------------------------------------------------------------
lgLabelAngleF                   NhlTFloat                RCSG
      TextAngleF                        0.0
----------------------------------------------------------------
lgLabelAlignment                NhlTlgLabelAlignmentModeRCSG
      LgLabelAlignment                  "ItemCenters"
----------------------------------------------------------------
lgLabelDirection                NhlTTextDirection        RCSG
      TextDirection                     "Across"
----------------------------------------------------------------
lgLabelJust                     NhlTJustification        RCSG
      TextJustification                 "CenterCenter"
----------------------------------------------------------------
lgLabelFont                     NhlTFont                 RCSG
      Font                              "pwritx"
----------------------------------------------------------------
lgLabelFontColor                NhlTColorIndex           RCSG
      FontColor                         "Foreground"
----------------------------------------------------------------
lgLabelFontHeightF              NhlTFloat                RCSG
      FontHeightF                       0.02
----------------------------------------------------------------
lgLabelFontAspectF              NhlTFloat                RCSG
      FontAspectF                       1.0
----------------------------------------------------------------
lgLabelFontThicknessF           NhlTFloat                RCSG
      FontThicknessF                    1.0
----------------------------------------------------------------
lgLabelFontQuality              NhlTFontQuality          RCSG
      FontQuality                       "High"
----------------------------------------------------------------
lgLabelConstantSpacingF         NhlTFloat                RCSG
      TextConstantSpacingF              0.0
----------------------------------------------------------------
lgLabelFuncCode                 NhlTCharacter            RCSG
      TextFuncCode                      ':'
----------------------------------------------------------------
lgLabelStride                   NhlTInteger              RCSG
      LgLabelStride                     1
----------------------------------------------------------------
lgTitleString                   NhlTString               RCSG
      LgTitleString                     <dynamic>
----------------------------------------------------------------
lgTitleOn                       NhlTBoolean              RCSG
      LgTitleOn                         True
----------------------------------------------------------------
lgTitlePosition                 NhlTPosition             RCSG
      LgTitlePosition                   "Top"
----------------------------------------------------------------
lgTitleExtentF                  NhlTFloat                RCSG
      LgTitleExtentF                    0.15
----------------------------------------------------------------
lgTitleAngleF                   NhlTFloat                RCSG
```

	TextAngleF		0.0	
lgTitleDirection	TextDirection	NhlTTextDirection	"Across"	RCSG
lgTitleFont	Font	NhlTFont	"pwritx"	RCSG
lgTitleFontColor	FontColor	NhlTColorIndex	"Foreground"	RCSG
lgTitleJust	TextJustification	NhlTJustification	"CenterCenter"	RCSG
lgTitleFontHeightF	FontHeightF	NhlTFloat	0.025	RCSG
lgTitleFontAspectF	FontAspectF	NhlTFloat	1.0	RCSG
lgTitleFontThicknessF	FontThicknessF	NhlTFloat	1.0	RCSG
lgTitleFontQuality	FontQuality	NhlTFontQuality	"High"	RCSG
lgTitleConstantSpacingF	TextConstantSpacingF	NhlTFloat	0.0	RCSG
lgTitleFuncCode	TextFuncCode	NhlTCharacter	':'	RCSG
lgBoxLinesOn	LgBoxLinesOn	NhlTBoolean	False	RCSG
lgBoxLineColor	LineColor	NhlTColorIndex	"Foreground"	RCSG
lgBoxLineThicknessF	LineThicknessF	NhlTFloat	1.0	RCSG
lgBoxLineDashPattern	LineDashPattern	NhlTDashIndex	"SolidLine"	RCSG
lgBoxLineDashSegLenF	LineDashSegLenF	NhlTFloat	0.15	RCSG
lgPerimOn	EdgesOn	NhlTBoolean	False	RCSG
lgPerimColor	EdgeColor	NhlTColorIndex	"Foreground"	RCSG
lgPerimFill	FillPattern	NhlTFillIndex	"HollowFill"	RCSG
lgPerimFillColor	FillColor	NhlTColorIndex	"Background"	RCSG
lgPerimThicknessF	EdgeThicknessF	NhlTFloat	1.0	RCSG

```
|-----------------------------------------------------------------|
| lgPerimDashPattern          NhlTDashIndex          RCSG          |
|       EdgeDashPattern               "SolidLine"                 |
|-----------------------------------------------------------------|
| lgPerimDashSegLenF          NhlTFloat              RCSG          |
|       EdgeDashSegLenF               0.15                        |
+-----------------------------------------------------------------+
```

Composite resources

The **Legend** class has no composite class objects.

Superclass resources

You can set all **View** class resources for the **Legend** object.

Description

The **Legend** object formats a series of lines and/or markers of varying styles (**Legend** 'items') along with accompanying explanatory labels into a horizontal row or vertical column. It is designed to serve as a "key" for an associated plot object. The labels, as well as an optional title, may be sized, angled, and positioned in a variety of ways. You may create a rectangular perimeter outlining of the area occupied by the Legend. The area inside this perimeter may be filled providing a background for the **Legend** object. Solid-filled or hollow boxes may surround each Legend item. You can set a number of the attributes of the **Legend** items individually using array resources.

Legend elements

The **Legend** object provides resources to manage the following elements:

- Position, size, and shape
- Attributes of the **Legend** items
- Attributes of the labels accompanying the items
- Attributes of the title
- Attributes of the perimeter line and the background area

Managing the legend position, size, and shape

Direct management

If you create a **Legend** directly, you set its basic location and size using the resources of its **View** superclass, *vpXF*, *vpYF*, *vpWidthF*, and *vpHeightF*. However, the **Legend** may adjust its size (*vpWidthF* and *vpHeightF*) to accommodate the constraints imposed by some of its own resources. The NhlTJustification resource *lgJustification* specifies the point of the **Legend** object that is to remain fixed when size adjustments are made. If *lgJustification* has a value other than `TopLeft`, the position (*vpXF* and *vpYF*) may also change when size adjustments occur.

The NhlTOrientation resource *lgOrientation* determines whether the **Legend** object arranges the Legend items in a vertical column or in a horizontal row. Note that *lgOrientation* relates to the arrangement of the group of objects, rather than the orientation of the individual items in the group. When *lgOrientation* is `Vertical`, **Legend** items that are lines are oriented *horizontally*. Based on value of this *lgOrientation* **Legend** defines its *major axis* to be the axis parallel to the direction of orientation and the *minor axis* as the axis orthogonal to the direction of orientation.

If the boolean resource, *lgAutoManage* is set True, the **Legend** controls the size of its elements such that it can remain as close as possible to the size specified by its **View** resources. The length of the minor axis will remain as specified by the appropriate **View** resource value (either *vpWidthF* or *vpHeightF* depending on the orientation). The major axis grows or shrinks only when if the angle of the **Legend** labels is modified.

When *lgAutoManage* is True, you have no direct control over the size of text used in the **Legend**. If you are working interactively, you may find it helpful to create a basic **Legend** layout close to the desired size with the *lgAutoManage* mode on, then switch it off to tune the text size precisely to your taste. Even when *lgAutoManage* is False, text size scales when you change the size of the **Legend** object, unless you explicitly set the text size at the same time.

You can cause the **Legend** as a whole to appear or disappear from the view surface, by manipulating the value of the boolean resource *lgLegendOn*.

Management as an annotation

More typically, **Legend** objects are created as annotations for plot objects such as **ContourPlot** or **XyPlot**. If not disabled by the plot object class, you can automatically instantiate a **Legend** object simply by setting the **PlotManager** resource *pmLegendDisplayMode* to any value other than `NoCreate`. **ContourPlot** and **XyPlot** create a **Legend** objects by default.

As an intrinsic annotation of the **PlotManager**, the **Legend** object's **View** class resources can no longer be accessed directly. You now must control the size and location using resources that mostly belong to the **PlotManager** class. You set the location using resources that follow the PlotManager Location Control Model, as follows:

- *pmLegendZone*
- *pmLegendSide*
- *pmLegendParallelPosF*
- *pmLegendOrthogonalPosF*
- *lgJustification*

Note that most of these resources belong to the **PlotManager** class, but the justification resource belongs to the **Legend**. Instead of using *vpWidthF* and *vpHeightF* to adjust the size, you now use the **PlotManager** resources *pmLegendWidthF* and *pmLegendHeightF*.

The **PlotManager** class modifies the behavior of certain **Legend** resources. The **ContourPlot** class modifies the behavior of others. The **XyPlot** class modifies still others. Otherwise, you can set and retrieve **Legend** resources just as you would if you instantiated the **Legend** directly.

Layout of the legend interior

Within the perimeter of the **Legend**, there is an optional title, the **Legend** items (lines or markers), and next to each item an optional label. Surrounding these elements you can specify a margin on each side using the resources *lgLeftMarginF*, *lgRightMarginF*, *lgBottomMarginF*, *lgTopMarginF*. If there is a title, it may be placed inside the margin along any of the edges, parallel to either axis. The amount of room allotted to the title is specified using the resource *lgTitleExtentF*. You can specify a space between the title extent and the other **Legend** elements using the resource *lgTitleOffsetF*. Between the items and the labels you can also specify a space using the resource *lgLabelOffsetF*.

The legend items

A **Legend** may consist either of *marker items* or *line items* or a combination of both. You set the number of items using the resource, *lgItemCount*. All the array resources belonging to the **Legend** are sized relative to the number of items.

Associated with each **Legend** item is a rectangular space called the *item box*. All **Legend** items are located in the center of their item box. Line items extend from one end of the item box to the other along the minor axis. Marker items, if made large enough, may extend outside the borders of their item box. You can make the outline of the item box visible by setting the resource *lgBoxLinesOn* True.

When *lgAutoManage* is True, the resource *lgBoxMinorExtentF* determines what fraction of the distance across the minor axis (after subtracting the margins) is occupied by the legend item box. If *lgAutoManage* is False, it still specifies this same fraction initially, but may no longer after the **Legend** size is adjusted based on the text heights of the labels and the title.

The item boxes placed in a column or row occupy all the extent of the major axis that remains after subtracting:

- the margins,
- the space occupied by the title if it is enabled and placed along one of the minor axis sides,
- any space added to accommodate angled labels,
- and the equivalent of the font height of the labels if *lgLabelAlignment* is set to `ExternalEdges`.

The resource *lgBoxMajorExtentF*, which should be set to a value greater than 0.0 and less than or equal to 1.0, specifies what fraction of the space alloted to each item box the box actually occupies. If *lgBoxMajorExtentF* is 0.5, for example, each item box is separated from its neighboring boxes by a space equal to its size. Note that adjustments to *lgBoxMajorExtentF* have no effect on the **Legend** items themselves, whether markers or lines.

If the NhlTlgItemPlacementMode resource *lgItemPlacement* is set to `ExplicitPlacement`, the array resource *lgItemPositions* sets the position of the center of each items as a fraction of the distance along the portion of the major axis extent reserved for **Legend** items. Using this resource you can create separate groupings of related items within a single **Legend** object.

The **Legend** object provides a number of resources for controlling the attributes of the items. Some attributes have both a scalar resource and an associated array resource. With the array resource you can control the attribute value for each item individually, while with the scalar resource you can set the attribute for all items to a uniform value. By convention, the name of the array resource is the plural form of the name of the scalar resource. A boolean resource, whose name is formed by adding the prefix *Mono* to the scalar resource name, specifies which of the two resources is to be used. (If the scalar name contains the suffix "F", indicating a float type resource, it is discarded before forming the plural or adding the *Mono* prefix.)

Here are the resources that allow individual control of item attributes:

lgItemType, *lgItemTypes*, *lgMonoItemType*
: This NhlTMarkLineMode resource sets the type of a **Legend** item, either to line, marker, or both.

lgDashIndex, *lgDashIndexes*, *lgMonoDashIndex*
: Sets the dash pattern index for line item types.

lgLineColor, *lgLineColors*, *lgMonoLineColor*
: Sets the HLU color index of line items.

lgLineThicknessF, *lgLineThicknesses*, *lgMonoLineThickness*
: Sets the thickness of the line for line items.

lgMarkerIndex, *lgMarkerIndexes*, *lgMonoMarkerIndex*
: Sets the marker index for marker item types.

lgMarkerColor, *lgMarkerColors*, *lgMonoMarkerColor*
: Sets the HLU color index of marker items.

lgMarkerThicknessF, *lgMarkerThicknesses*, *lgMonoMarkerThickness*
: Sets the thickness of the line used for marker symbols (if the symbol is not taken from one of the filled fonts).

lgMarkerSizeF, *lgMarkerSizes*, *lgMonoMarkerSize*
: Specifies the height used for the marker symbol for marker items.

lgLineLabelStrings
: Sets the internal line label text for line items.
: (Do not confuse with *lgLabelStrings*, used for the label that describes each **Legend** item.)

lgLineLabelFontHeightF, *lgLineLabelFontHeights*, *lgMonoLineLabelFontHeight*
: Specifies the font height used for the internal line labels.

lgLineLabelFontColor, *lgLineLabelFontColors*, *lgMonoLineLabelFontColor*
: Specifies the color used for the internal line labels for line items.

In addition, **Legend** supports a number of scalar-only resources for controlling other attributes of the **Legend** items. These resources apply uniformly to all items of the correct type. *lgLineDashSegLenF* controls the segment length used for line item dash patterns. Note that the dash segment length for line items does *not* scale when the size of the **Legend** object changes. This is because the appearance of the **Legend** lines needs to be dependent on the plot object using the **Legend** as an annotation, and not upon the current size of the **Legend** itself.

lgLineLabelsOn is a boolean flag that specifies whether line items should be drawn with internal labels. You may set all the remaining text attributes of the internal line labels using the resources *lgLineLabelFont*, *lgLineLabelFontAspectF*, *lgLineLabelFontThicknessF*, *lgLineLabelFontQuality*, *lgLineLabelConstantSpacingF*, and *lgLineLabelFuncCode*.

If you set the boolean resource *lgBoxLinesOn* True, you can draw the perimeter outline of each **Legend** item box. You can control the color, thickness, dash pattern, and the dash segment length of these perimeter lines using the resources *lgBoxLineColor*, *lgBoxLineThicknessF*, *lgBoxLineDashPattern*, and *lgBoxLineDashSegLenF*. You can also use the resource *lgBoxBackground* to specify a solid fill color for the background of these boxes.

Legend labels

You choose whether or not to provide an explanatory label to accompany the **Legend** items by setting the boolean resource *lgLabelsOn* True or False. The actual label strings are set using the array resource *lgLabelStrings*.

You can place the labels on either side of the items or center them on items using the NhlTPosition resource *lgLabelPosition*. If you set *lgLabelPosition* position to a value inappropriate for the orientation (e.g. `Bottom` when *lgOrientation* is `Vertical`), it is automatically coerced to a valid value. Using the NhlTlgLabelAlignmentMode resource *lgLabelAlignment*, you can align the labels to the centers of the items, or above or below the items. If you set *lgLabelPosition* to `Center`, you should set *lgLabelAlignment* to `AboveItems` or `BelowItems` unless you want the labels to appear on top of the **Legend** items.

If the **Legend** has too many items for there to be a reasonably sized label for each item, you can set the *lgLabelStride* resource in order to label only every n'th box.

The labels have a complete set of text attribute resources. When *lgAutoManage* is True, however, the resources *lgLabelFontHeightF* or *lgLabelJust* are set automatically. In this case, **Legend** adjusts the font height and justification such that the labels fit the available space and remain properly aligned with their boxes. If the *lgLabelAngleF* resource is varied, **Legend** ensures that the labels do not overlap one another. If *lgAutoManage* is set False, the **Legend** viewport expands if necessary to accommodate the full label text entent. However, in this mode, there are no checks to prevent the labels from overlapping, and the justification must be set manually in order for the labels to appear properly aligned if the *lgLabelAngleF* resource is set to certain ranges of values.

The legend title

You turn the **Legend** title on and off using the boolean resource, *lgTitleOn*. You set the text of the title using the resource, *lgTitleString*. Any time you set *lgTitleString* and do not explicitly set *lgTitleOn* at the same time, *lgTitleOn* is automatically set True. You set the title's position with the NhlTPosition resource *lgTitlePosition*. The only restriction on the title's position is that it cannot be set to `Center`.

Like the labels, the title has a full set of text attribute resources. If *lgAutoManage* is True, the title is automatically sized to fit the available space as determined by the **Legend** viewport and the *lgTitleExtentF* resource. Attempts to set the *lgTitleFontHeightF* resource are ignored. You can influence the title's size indirectly, however. When *lgAutoManage* is set False, you can set *lgTitleFontHeightF* directly. If necessary, the **Legend** instance will increase the size of its viewport to accommodate the full extent of the title string.

The legend perimeter

You can draw an outline around the perimeter of the **Legend** by setting the boolean resource *lgPerimOn*

True. You can control all attributes of the perimeter, including line color, line thickness, line dash pattern, and the dash segment length, using the resources *lgPerimColor*, *lgPerimThicknessF*, *lgPerimDashPattern*, and *lgPerimDashSegLenF*.

You can fill the interior area defined by the perimeter (in other words, the area around the boxes and behind the title and labels), using the resources *lgPerimFill* and *lgPerimFillColor*.

Support functions

The **Legend** object does not define any support functions, but it inherits all support functions available to the **View** class.

Status

See also

- NhlCreate function
- NhlSetValues function
- NhlGetValues function
- PlotManager object

Copyright

Copyright 1987-1999 University Corporation for Atmospheric Research
The use of this Software is governed by a License Agreement.

NCAR Graphics is a registered trademark of the University Corporation for Atmospheric Research.

LogLinPlot class

The **LogLinPlot** class draws immediate mode lines and may be used to map overlay plots into a log or linear coordinate space.

Synopsis

```
Header file:            ncarg/hlu/LogLinPlot.h
Class name:             logLinPlotClass
```

```
Class pointer:          NhllogLinPlotClass
Fortran class pointer:  NHLFLOGLINPLOTCLASS
Superclass:             Transform
Composite classes:      LogLinTransformation, PlotManager
```

Resources

Local resources

The LogLinPlot object has no local resources.

Composite resources

LogLinTransformation resources

You can access all **LogLinTransformation** resources. These resources define the extent and direction of the data coordinates, and whether each axis is to use a logarithmic or linear coordinate system.

PlotManager resources

Assuming *tfPlotManagerOn* is True when a **LogLinPlot** object is created, you can access all **PlotManager** resources. You can also access resources for any of the **PlotManager's** composite class members. However, the **PlotManager** object modifies the access and behavior of some of the resources belonging to these objects, as follows:

- LabelBar resources as modified by PlotManager
- Legend resources as modified by PlotManager
- TickMark resources as modified by PlotManager
- Title resources as modified by PlotManager

Superclass resources

You can access all resources defined by the superclasses of the **LogLinPlot** object class, including:

- **Transform** resources
- **View** resources

Description

Unlike most **Transform** subclasses, **LogLinPlot** has no specialized plot elements and no **Draw** method of its own. Its primary function is to act as a base plot over which other plot objects may be overlaid in order to transform their data space into a log/linear coordinate space. Like other members of the **Transform** class, **LogLinPlot** can also function as a "canvas" for drawing graphics primitives, either in data space or in NDC space.

You can transform a plot object into log/linear coordinate space by overlaying it on a **loglinplot**. The log/linear coordinate space may encompass the data space of any number of overlaid plot objects with partially or completely disjoint data coordinate extents. You set the extent and direction of the data coordinate space, as well as whether one or both axes should use a logarithmic coordinate system, using the resources of the **LogLinTransformation** object belonging to the **LogLinPlot**. **LogLinPlot** allows unrestricted access to all the resources of its **LogLinTransformation** object.

Assuming *tfPlotManagerOn* is True when you create a **LogLinPlot** object, the **LogLinPlot** object has the same plot management capabilities as any other plot object. The **LogLinPlot** object does not intercept any resources belonging to the **PlotManager** object or the composite class members of the **PlotManager**. Therefore, you access these resources just as described for the **PlotManager** object itself.

Support functions

The **LogLinPlot** object does not define any support functions, but inherits all the support functions available to its superclasses.

Status

See also

- NhlCreate function
- NhlDestroy function
- NhlSetValues function
- NhlGetValues function
- LogLinTransformation object
- PlotManager object
- Transform object
- View object
- Base object

Copyright

Copyright 1987-1999 University Corporation for Atmospheric Research
The use of this Software is governed by a License Agreement.

NCAR Graphics is a registered trademark of the University Corporation for Atmospheric Research.

LogLinTransformation class

The **LogLinTransformation** class manages forward and reverse transformations in log or linear coordinate space.

Synopsis

```
Header file:            ncarg/hlu/LogLinTransformation.h
Class name:             logLinTransformationClass
Class pointer:          <Not referenceable>
Fortran class function: <Not referenceable>
Superclass:             Transformation
Composite classes:      <None>
```

Resources

Local resources

LogLinTransformation resources are accessible only through **Transform** class objects that use the **LogLinTransformation** to manage transformations to and from data space. These include:

- **ContourPlot** object
- **LogLinPlot** object
- **StreamlinePlot** object
- **VectorPlot** object
- **VectorPlot** object

```
+---------------------------------------------------------------+
|              LogLinTransformation Resource Set                |
|---------------------------------------------------------------|
| NAME                         TYPE                   ACCESS    |
|       CLASS                          DEFAULT                  |
|===============================================================|
| trXLog                       NhlTInteger            RCSG      |
|       TrXLog                         0                        |
|---------------------------------------------------------------|
| trYLog                       NhlTInteger            RCSG      |
|       TrYLog                         0                        |
+---------------------------------------------------------------+
```

Composite resources

The **LogLinTransformation** class has no composite class objects.

Superclass resources

You can set all resources defined by the superclasses of the **LogLinTransformation** object class, including:

- **Transformation** resources

Description

The **LogLinTransformation** provides transformations in logarithmic or linear coordinate space for **Transform** class objects. Its **Transformation** superclass provides the basic resources for specifying the coordinate extent and direction. **LogLinTransformation** adds resources that specify for each each axis whether it is to be linear or logarithmic.

You do not create objects of the **LogLinTransformation** class directly. **Transform** objects that use a **LogLinTransformation** child, however, generally provide controlled access to some or all of its resources. Ordinarily they set the default extent and direction resources based on values contained in a currently associated data object. By manipulating the **Transformation** superclass resources, however, you can define coordinate spaces that arbitrarily intersect or encompass the data space.

Support functions

There are no special support functions defined for the **LogLinTransformation** class or its superclasses.

Status

See also

- LogLinPlot object
- PlotManager object
- ContourPlot object
- StreamlinePlot object
- VectorPlot object
- XyPlot object
- Transformation object
- Obj object

Copyright

Copyright 1987-1999 University Corporation for Atmospheric Research
The use of this Software is governed by a License Agreement.

NCAR Graphics is a registered trademark of the University Corporation for Atmospheric Research.

MapPlot class

The **MapPlot** class draws outlined and/or filled maps of arbitrary regions of the globe and may be used to map overlay plots into a number of standard projections.

Synopsis

```
Header file:            ncarg/hlu/MapPlot.h
Class name:             mapPlotClass
Class pointer:          NhlmapPlotClass
Fortran class function: NHLFMAPPLOTCLASS
Superclass:             Transform
Composite classes:      MapTransformation,PlotManager
```

Class-defined types

```
Type name:              NhlTMapDataBaseVersion
Definition:
typedef enum _NhlMapDataBaseVersion {
        NhlNCARG4_0      = 0, /* "Ncarg4_0" */
        NhlNCARG4_1      = 1  /* "Ncarg4_1" */
} NhlMapDataBaseVersion;

Type name:              NhlTMapBoundarySets
Definition:
typedef enum _NhlMapBoundarySets {
        NhlNOBOUNDARIES             = 0, /* "NoBoundaries"            */
        NhlGEOPHYSICAL              = 1, /* "Geophysical"             */
        NhlNATIONAL                 = 2, /* "National"                */
        NhlUSSTATES                 = 3, /* "USStates"                */
        NhlGEOPHYSICALANDUSSTATES   = 4, /* "GeophysicalAndUSStates" */
        NhlALLBOUNDARIES            = 5  /* "AllBoundaries"           */
} NhlMapBoundarySets;

Type name:              NhlTSpecifiedFillPriority
Definition:
typedef enum _NhlSpecifiedFillPriority {
        NhlGEOPHYSICALPRIORITY      = 0, /* "GeophysicalPriority" */
        NhlPOLITICALPRIORITY        = 1  /* "PoliticalPriority"   */
} NhlSpecifiedFillPriority;

Type name:              NhlTMapGridMaskMode
Definition:
typedef enum _NhlMapGridMaskMode {
        NhlMASKNONE                 = 0, /* "MaskNone"     */
        NhlMASKOCEAN                = 1, /* "MaskOcean"    */
        NhlMASKNOTOCEAN             = 2, /* "MaskNotOcean" */
        NhlMASKLAND                 = 3, /* "MaskLand"     */
```

```
        NhlMASKNOTLAND              = 4, /* "MaskNotLand"  */
        NhlMASKFILLAREA             = 5, /* "MaskFillArea" */
        NhlMASKMASKAREA             = 6  /* "MaskMaskArea" */
} NhlMapGridMaskMode;

Type name:              NhlTMapShapeMode
Definition:
typedef enum _NhlMapShapeMode {
        NhlFREEASPECT               = 0, /* "FreeAspect"         */
        NhlFIXEDASPECTFITBB         = 1, /* "FixedAspectFitBB"   */
        NhlFIXEDASPECTNOFITBB       = 2  /* "FixedAspectNoFitBB" */
} NhlMapShapeMode;
```

Resources

Local resources

```
+---------------------------------------------------------------+
|                       MapPlot Resource Set                     |
|---------------------------------------------------------------|
| NAME                         TYPE                    ACCESS   |
|       CLASS                          DEFAULT                  |
|===============================================================|
| mpDataBaseVersion            NhlTMapDataBaseVersion  RCSG     |
|       MpDataBaseVersion              "Ncarg4_0"               |
|---------------------------------------------------------------|
| mpShapeMode                  NhlTMapShapeMode        RCSG     |
|       MpShapeMode                    "FixedAspectFitBB"       |
|---------------------------------------------------------------|
| mpAreaNames                  NhlTStringGenArray      RCSG     |
|       MpAreaNames                    <array>                  |
|---------------------------------------------------------------|
| mpAreaTypes                  NhlTIntegerGenArray     G        |
|       MpAreaTypes                    <array>                  |
|---------------------------------------------------------------|
| mpFixedAreaGroups            NhlTIntegerGenArray     G        |
|       MpFixedAreaGroups              <array>                  |
|---------------------------------------------------------------|
| mpDynamicAreaGroups          NhlTIntegerGenArray     RCSG     |
|       MpDynamicAreaGroups            <array>                  |
|---------------------------------------------------------------|
| mpAreaGroupCount             NhlTInteger             RCSG     |
|       MpAreaGroupCount               10                       |
|---------------------------------------------------------------|
| mpOutlineOn                  NhlTBoolean             RCSG     |
|       MpOutlineOn                    True                     |
|---------------------------------------------------------------|
| mpOutlineDrawOrder           NhlTDrawOrder           RCSG     |
|       MpOutlineDrawOrder             "PostDraw"               |
|---------------------------------------------------------------|
| mpOutlineBoundarySets        NhlTMapBoundarySets     RCSG     |
|       MpOutlineBoundarySets          "Geophysical"            |
|---------------------------------------------------------------|
| mpOutlineSpecifiers          NhlTStringGenArray      RCSG     |
|       MpOutlineSpecifiers            NULL                     |
|---------------------------------------------------------------|
```

mpAreaMaskingOn	MpAreaMaskingOn	NhlTBoolean	False	RCSG
mpMaskAreaSpecifiers	MpMaskAreaSpecifiers	NhlTStringGenArray	NULL	RCSG
mpFillOn	MpFillOn	NhlTBoolean	False	RCSG
mpFillDrawOrder	MpFillDrawOrder	NhlTDrawOrder	"Draw"	RCSG
mpFillBoundarySets	MpFillBoundarySets	NhlTMapBoundarySets	"Geophysical"	RCSG
mpFillPatternBackground	FillBackgroundColor	NhlTColorIndex	"Background"	RCSG
mpMonoFillColor	MpMonoFillColor	NhlTBoolean	False	RCSG
mpFillColor	FillColor	NhlTColorIndex	"Foreground"	RCSG
mpFillColors	MpFillColors	NhlTColorIndexGenArray	<array>	RCSG
mpMonoFillPattern	MpMonoFillPattern	NhlTBoolean	True	RCSG
mpFillPattern	FillPattern	NhlTFillIndex	"SolidFill"	RCSG
mpFillPatterns	MpFillPatterns	NhlTFillIndexGenArray	<array>	RCSG
mpMonoFillScale	MpMonoFillScale	NhlTBoolean	True	RCSG
mpFillScaleF	FillScaleF	NhlTFloat	1.0	RCSG
mpFillScales	MpFillScales	NhlTFloatGenArray	<array>	RCSG
mpFillAreaSpecifiers	MpFillAreaSpecifiers	NhlTStringGenArray	NULL	RCSG
mpSpecifiedFillDirectIndexing	MpSpecifiedFillDirectIndexing	NhlTBoolean	False	RCSG
mpSpecifiedFillPriority	MpSpecifiedFillPriority	NhlTSpecifiedFillPriority	"GeophysicalPriority"	RCSG
mpSpecifiedFillColors	MpSpecifiedFillColors	NhlTColorIndexGenArray	<dynamic array>	RCSG
mpSpecifiedFillPatterns	MpSpecifiedFillPatterns	NhlTFillIndexGenArray	<dynamic array>	RCSG
mpSpecifiedFillScales		NhlTFloatGenArray		RCSG

MpSpecifiedFillScales	<dynamic array>	
mpDefaultFillColor	NhlTColorIndex	RCSG
FillColor	16	
mpDefaultFillPattern	NhlTFillIndex	RCSG
FillPattern	"SolidFill"	
mpDefaultFillScaleF	NhlTFloat	RCSG
FillScaleF	1.0	
mpOceanFillColor	NhlTColorIndex	RCSG
FillColor	10	
mpOceanFillPattern	NhlTFillIndex	RCSG
FillPattern	"SolidFill"	
mpOceanFillScaleF	NhlTFloat	RCSG
FillScaleF	1.0	
mpLandFillColor	NhlTColorIndex	RCSG
FillColor	8	
mpLandFillPattern	NhlTFillIndex	RCSG
FillPattern	"SolidFill"	
mpLandFillScaleF	NhlTFloat	RCSG
FillScaleF	1.0	
mpInlandWaterFillColor	NhlTColorIndex	RCSG
FillColor	10	
mpInlandWaterFillPattern	NhlTFillIndex	RCSG
FillPattern	"SolidFill"	
mpInlandWaterFillScaleF	NhlTFloat	RCSG
FillScaleF	1.0	
mpGeophysicalLineColor	NhlTColorIndex	RCSG
LineColor	"Foreground	
mpGeophysicalLineDashPattern	NhlTDashIndex	RCSG
LineDashPattern	"SolidLine"	
mpGeophysicalLineDashSegLenF	NhlTFloat	RCSG
LineDashSegLenF	<dynamic>	
mpGeophysicalLineThicknessF	NhlTFloat	RCSG
LineThicknessF	1.0	
mpUSStateLineColor	NhlTColorIndex	RCSG
LineColor	"Foreground"	
mpUSStateLineDashPattern	NhlTDashIndex	RCSG
LineDashPattern	"SolidLine"	
mpUSStateLineDashSegLenF	NhlTFloat	RCSG
LineDashSegLenF	<dynamic>	
mpUSStateLineThicknessF	NhlTFloat	RCSG
LineThicknessF	1.0	

mpNationalLineColor LineColor	NhlTColorIndex "Foreground"	RCSG
mpNationalLineDashPattern LineDashPattern	NhlTDashIndex "SolidLine"	RCSG
mpNationalLineDashSegLenF LineDashSegLenF	NhlTFloat <dynamic>	RCSG
mpNationalLineThicknessF LineThicknessF	NhlTFloat 1.0	RCSG
mpGridAndLimbOn MpGridAndLimbOn	NhlTBoolean True	RCSG
mpGridAndLimbDrawOrder MpGridAndLimbDrawOrder	NhlTDrawOrder "PostDraw"	RCSG
mpGridMaskMode MpGridMaskMode	NhlTMapGridMaskMode "MaskNone"	RCSG
mpGridSpacingF MpGridSpacingF	NhlTFloat 15.0	RCSG
mpGridLatSpacingF MpGridLatSpacingF	NhlTFloat 15.0	RCSG
mpGridLonSpacingF MpGridLonSpacingF	NhlTFloat 15.0	RCSG
mpGridMaxLatF MpGridMaxLatF	NhlTFloat 15.0	RCSG
mpGridPolarLonSpacingF MpGridPolarLonSpacingF	NhlTFloat 15.0	RCSG
mpGridLineColor LineColor	NhlTColorIndex "Foreground"	RCSG
mpGridLineDashPattern LineDashPattern	NhlTDashIndex "SolidLine"	RCSG
mpGridLineDashSegLenF DashSegLenF	NhlTFloat <dynamic>	RCSG
mpGridLineThicknessF LineThicknessF	NhlTFloat 1.0	RCSG
mpLimbLineColor LineColor	NhlTColorIndex "Foreground"	RCSG
mpLimbLineDashPattern LineDashPattern	NhlTDashIndex "SolidLine"	RCSG
mpLimbLineDashSegLenF LineDashSegLenF	NhlTFloat <dynamic>	RCSG
mpLimbLineThicknessF LineThicknessF	NhlTFloat 1.0	RCSG

mpPerimOn EdgesOn	NhlTBoolean False	RCSG
mpPerimDrawOrder MpPerimDrawOrder	NhlTDrawOrder "Draw"	RCSG
mpPerimLineColor LineColor	NhlTColorIndex "Foreground"	RCSG
mpPerimLineDashPattern LineDashPattern	NhlTDashIndex "SolidLine"	RCSG
mpPerimLineDashSegLenF LineDashSegLenF	NhlTFloat <dynamic>	RCSG
mpPerimLineThicknessF LineThicknessF	NhlTFloat 1.0	RCSG
mpLabelsOn PlotLabelsOn	NhlTBoolean False	RCSG
mpLabelDrawOrder MpLabelDrawOrder	NhlTDrawOrder "PostDraw"	RCSG
mpLabelFontHeightF FontHeightF	NhlTFloat <dynamic>	RCSG
mpLabelFontColor FontColor	NhlTColorIndex True	RCSG

Composite resources

MapTransformation resources

You can access all **MapTransformation** resources. The **MapTransformation** resources define the projection and the portion of the world map visible within the NDC viewport.

PlotManager resources

If *tfPlotManagerOn* is True when a **MapPlot** object is created, you can access all **PlotManager** resources except that if you attempt to set *pmTickMarkDisplayMode* to a value other than `NoCreate` you will receive a warning message. Tick marks have not yet been implemented for map projections. Except for **TickMark** resources, you can also access resources for the **PlotManager** composite annotation objects. However, the **PlotManager** object modifies the access and behavior of some of the resources belonging to these objects, as follows:

- LabelBar resources as modified by PlotManager
- Legend resources as modified by PlotManager
- Title resources as modified by PlotManager

Superclass resources

You can access all resources defined by the superclasses of the **MapPlot** object class, including:

- **Transform** resources
- **View** resources

Description

The **MapPlot** object renders maps of the world in any of 10 different projections. It allows you to limit the projected view to a portion of the globe, as well as to limit the rendering to entities individually selected by name from either of the MapPlot databases. You can outline areas, you can fill areas, or you can *mask* areas, allowing previously rendered plot elements to remain visible.

MapPlot allows access to two different databases containing points defining geophysical and political features of the globe. The original database, defined at a generally lower resolution, is out out of date with respect to national borders. However, because of its lower resolution, rendering is much faster for maps that span a major portion of the globe. Therefore the low resolution database remains the default. The new database defines twice as many named areas, has much higher resolution, and the political boundaries are up to date as of mid-year 1998. Because of the density of its data, it is best suited for plots of limited areas of the globe where the greater detail can be appreciated.

You can use **MapPlot** as a base plot on which one or more plot objects, such as a **ContourPlot** object, are overlaid, thereby transforming its elements into the current map projection. You can also control the order of drawing of **MapPlot** elements relative to each other and to the elements of any overlaid plot objects. Used in conjunction with area masking, these facilities allow you to limit overlay plots to the geographical boundaries for which their data have meaning. For instance, you can confine a contour plot of ocean temperatures to areas representing only the ocean. Note that unlike most plot objects, you cannot add the **MapPlot** as an overlay of another base plot. It can, however, be used as an annotation plot.

MapPlot supports grids representing latitude and longitude. You can mask the grids independently of area masking. Also supported are labels for some key features of the world map, including the poles, the equator, the Greenwich Meridian and the International Dateline. Finally, you can draw a perimeter, either rectangular or elliptical, around the projected map area.

MapPlot elements

There are six basic elements provided by **MapPlot**:

Outlines
: Map boundary outlines are turned on by default. Turn them off by setting *mpOutlineOn* to False. The drawing order is specified using *mpOutlineDrawOrder*.

Fill
: Map area fill is turned off by default. Turn it on by setting *mpFillOn* True. The drawing order is specified using *mpFillDrawOrder*.

Masking
: Masking is turned off by default. Turn it on by setting *mpAreaMaskingOn*. When masking is turned on, areas specified by name in the *mpMaskAreaSpecifiers* array resource remain empty

when fill drawing occurs.

Grid and limb lines

Grid and limb lines are turned on by default. Turn them off by setting *mpGridAndLimbOn* to False. The drawing order is specified using *mpGridAndLimbDrawOrder*.

Perimeter line

The perimeter line is turned off by default. Turn it on by setting *mpPerimOn* to True. The drawing order is specified using *mpPerimDrawOrder*.

Labels

Labels are turned off by default. Turn them on by setting *mpLabelsOn* to True. The drawing order is specified using *mpLabelDrawOrder*.

Controlling the MapPlot transformation

The **MapPlot** object itself contains the resources that control the rendering of plot features such as area outlines and fill, but you control the projection, its center, and how much of the globe is to appear using resources belonging to the **MapPlot's** composite MapTransformation object. Note that it is possible to overlay another plot object on a **MapPlot** object using one of the projections provided by the **MapTransformation**, and have *no* **MapPlot** elements appear in the plot. You would simply turn off each **MapPlot** element that is drawn by default. That is, you would set the resources *mpOutlineOn*, and *mpGridAndLimbOn* False.

The map projections provided by the **MapTransformation** must retain their intrinsic shape in order to be considered "correct" projections. However, in some situations you may not care whether this shape is preserved. The **MapPlot** object provides a resource, *mpShapeMode*, that allows you to decide whether or not to preserve the shape intrinsic to the projection. Moreover, if shape is to be preserved, you can decide whether you want to shrink the **MapPlot** viewport to fit around the projected area, which is centered in the originally specified viewport; or whether you want the viewport to remain as originally specified, even if the projected area cannot completely fill the space. In this case, you can get the values of the **MapTransformation** resources *mpLeftMapPosF*, *mpRightMapPosF*, *mpBottomMapPosF*, and *mpTopMapPosF* in order to find the actual boundaries of the projected area.

The MapPlot databases

The original **MapPlot** Ncarg4_0 database contains 583 areas uniquely identified by name, while the Ncarg4_1 database contains 1085 areas, almost twice as many. The Ncarg4_1 database uses a hierarchical model that is much more extensible. However, because the Ncarg4_0 database has a lower resolution that can be rendered more efficiently when a global-scale map is drawn, it remains the default. You can choose between the two databases by setting the enumerated resource *mpDataBaseVersion*.

Area names

You can retrieve a list of all the area names in the **MapPlot** database being used by getting the value of the resource *mpAreaNames*. The list is stored in a fixed order. By setting the *mpAreaNames* resource you *can* change an area's name at runtime, but the boundary it describes remains fixed and always has the same element number within the *mpAreaNames* array. In addition to a name, each area has a type. Each area also belongs to two area groups that are used to determine its fill attributes.

Area types

You can retrieve an area's type using the read-only integer array resource *mpAreaTypes*. The number and meaning of the area type categories vary depending on the **MapPlot** database in use. The Ncarg4_0 database has eight categories while the Ncarg4_1 database currently has four.

When using the Ncarg4_0 database:

The elements of the *mpAreaTypes* array have the same ordering as the *mpAreaNames* array. There are 8 types, numbered from 0, as follows:

- (0) mpOcean
- (1) mpContinent
- (2) mpLargeIsland
- (3) mpSmallIsland
- (4) mpInlandWater
- (5) mpNational
- (6) mpUSStateLand
- (7) mpUSStateWater

The division between large islands and small islands is arbitrary; its purpose is to make it easy to create small scale maps with a less cluttered appearance and some savings of processing time. Each area has only one type, and its geophysical properties are considered first. In other words, areas are classified as *mpNational* only if they form a proper subset of a geophysical entity. For instance, Bolivia is of type *mpNational* because it is part, but not the whole, of the geophysical area North-and-South-America. On the other hand, Australia is both a geophysical entity (a continent) and a country, therefore its type is *mpContinent*. You do not use the area types directly to control the operation of **MapPlot**; however knowledge of an area's type is useful for understanding the **MapPlot** object's operation.

When using the Ncarg4_1 database:

The elements of the *mpAreaTypes* array have the same ordering as the *mpAreaNames* array. There are 4 types currently, numbered from 1, as follows:

- (1) mpGeophysical
- (2) mpContinental
- (3) mpNational
- (4) mpState

Pre-defined string constants

When using the Ncarg4_0 database:

Somewhat related to the area types are the pre-defined string constants that specify areas by category. You use these strings when setting area specifier string array resources. Here are the constant strings (which **MapPlot** handles in case-insensitive manner):

- "NullArea"
- "Land"
- "Water"
- "InlandWater"
- "Oceans"
- "Continents"

- "Islands"
- "LargeIslands"
- "SmallIslands"
- "AllUSStates"
- "USStatesLand"
- "USStatesWater"
- "AllNational"
- "AllGeophysical"

Unlike area types, the area categories described by these string constants can overlap. The category "Islands," for instance, includes all the members of "LargeIslands" and "SmallIslands." On the other hand, note that all members of the set "LargeIslands" are of type *mpLargeIsland*, so knowing an area's type can help you figure out whether it is a member of a particular constant string category. The string "NullArea" specifies no area at all; it is useful for eliminating an element of one of the area specifier arrays without resizing the array or rearranging the remaining elements.

When using the Ncarg4_1 database:

Only the "NullArea" string constant is supported. The string "NullArea" specifies no area at all; it is useful for eliminating an element of one of the area specifier arrays without resizing the array or rearranging the remaining elements.

Boundary sets and area specifiers

The **MapPlot** object has two types of resources for specifying areas. The first type, supported for outlines and fill, uses resources of type NhlTMapBoundarySets to specify very generally the area sets to be used. The second type, supported for outlines, fill and area masking, allows you to specify groups of areas by category and individual areas by name using string array type resources. For outlines and fill, it is often convenient to use both resource types to complete the specification of the areas desired.

The string array resources allow individual specification of areas and area categories. To the basic boundary specification (which might well be `NoBoundaries`), you add other areas you want to outline or fill and subtract areas you want to mask. You do this using string array specifier resources, of which there are three: *mpOutlineSpecifiers*, *mpFillAreaSpecifiers*, and *mpMaskAreaSpecifiers*.

When using the Ncarg4_0 database:

The specifier strings you use for these resources may be any of the **MapPlot** pre-defined string constants, explicit names from the **MapPlot** Ncarg4_0 database, or matching substrings. All specifier strings are treated in a case-insensitive manner.

When using the Ncarg4_1 database:

The specifier strings you use for these resources may be names from the **MapPlot** Ncarg4_1 database. The name may be a fully qualified name or contain any part of the basic name's parental hierarchy to the left of the basic name. The hierarchical parts may be separated either using a colon character (':') or a period character ('.'). The specifier strings are treated in a case-insensitive manner.

Substring matching

When using the Ncarg4_0 database:

You form a matching substring by placing the wildcard character '*' at either or both ends of the character group you want to match. For instance, the substring "canada*" matches all entries in the database beginning with the name "Canada," and "*lake*" would match all entries with the letters "lake" (any case) anywhere in the name.

When using the Ncarg4_1 database:
Substring matching is not supported.

Outlines

By default **MapPlot** draws area outlines, but you can turn them off by setting *mpOutlineOn* to False. You establish the basic set of outlines to use by choosing a value for the NhlTMapBoundarySets type resource *mpOutlineBoundarySets*. If you wish, you can then add to the basic outline set by specifying other areas using the resource *mpOutlineSpecifiers*. Unlike fill, the current version of **MapPlot** does not allow individual control of the lines defining the boundary of a particular area. However, you can set outline attributes for three general boundary categories, depending on which of the three basic datasets the line belongs to: geophysical, national, or U.S. States. **MapPlot** supports line color, line dash pattern, line dash segment length, and line thickness for each category.

Area fill

Since area fill is more compute intensive and may not be appropriate for many plotting tasks, **MapPlot** turns it off by default. Set *mpFillOn* True to turn it on. **MapPlot** fill has more options than outlines, and thus is a bit more complicated to explain. Like outlines, you establish a basic set of fill areas using the NhlTMapBoundarySets type resource *mpFillBoundarySets*. To this set you can add other fill areas using the string array resource, *mpFillAreaSpecifiers*.

Normally **MapPlot** determines the fill attributes for each area based on its group memberships within the fixed and dynamic area group arrays. However, for areas specified explicitly in the *mpFillAreaSpecifiers* array, you can override the usual fill attribute choices and pick the fill attributes explicitly. This will be explained further below.

Area groups

MapPlot defines a set of area groups that it uses to determine the "normal" fill attributes assigned to each filled area. By default there are 10 area groups, arranged as follows:

Group 0: *Default group*
Determines the fill attributes for areas that fall within the map projection boundaries, but otherwise are neither specified as fill areas nor as mask areas.
Group 1: *Ocean group*
Determines the fill attributes for the world oceans.
Group 2: *Land group*
Determines the fill attributes for continents and islands.
Group 3: *Inland water group*
Determines the fill attributes for lakes, inland seas, and all other bodies of water except the oceans.
Groups 4 through 9 (default) through 255 (maximum): *Dynamic groups*
By default, **MapPlot** assigns areas to the dynamic groups in such as way as to ensure that

adjoining land areas never belong to the same group, and therefore have distinct fill attributes. However, the user may add more dynamic groups and reassign memberships in order to achieve any desired distribution of fill attributes.

MapPlot maintains two arrays of these area groups, accessible through the integer array resources *mpFixedAreaGroups* and *mpDynamicAreaGroups*. Both arrays are keyed to the *mpAreaNames* array, and within each array, the value of each element represents the area group assigned to an area. Most areas are assigned to different groups in the two area group areas, but in a number of cases (all areas representing water bodies, for instance), **MapPlot** assigns an area to the same group in both arrays.

The *mpFixedAreaGroups* array is sometimes known as the *geophysical* group array, because it provides the indexes that allow you to distinguish land areas from water areas. As its name implies, you cannot modify the *mpFixedAreaGroups* resource. Within this array, based on area type, **MapPlot** assigns each area in the database either to the Ocean group, to the Land group, or to the InlandWater group (i.e. to groups 1, 2, or 3).

The *mpDynamicAreaGroups* array is sometimes called the *political* group array, since the default assignment of dynamic area group numbers guarantees that adjoining politically distinct areas belong to different area groups. However, you are free to assign area groups within the dynamic area group array in any way you please. You may use all the available area groups from 1 through the current value of *mpAreaGroupCount*.

Setting fill attributes based on area groups

The elements of the array resources used to set fill color, fill pattern, and fill scale are indexed through the area group numbers. In other words, the first element of each of these arrays defines a fill attribute for group 0, the second an attribute for group 1, and so forth. For convenience, you can also access the fill attribute resources of groups 0 through 4 using a set of resources with names corresponding to the area group name. These resources have names with the following prefixes: *mpDefault...*, *mpOcean...*, *mpLand...*, and *mpInlandWater...*. You can think of these resources as aliases for the corresponding array resource element. When an array resource and a named alias resource accessing one of its elements are set in the same **SetValues** call, the named alias resource overrides.

Each of the three basic fill attributes supported by **MapPlot**, fill color, fill pattern, and fill pattern scale factor, provides both a scalar resource that you can use if you want to set all area groups to the same value, and an array resource that you use if you want to set each area group individually. The scalar fill attribute resources are *mpFillColor*, *mpFillPattern*, and *mpFillScaleF*. The array resources names, formed by pluralizing the scalar resource names (after removing the 'F' suffix, if present) are *mpFillColors*, *mpFillPatterns*, and *mpFillScales*. The boolean resources, *mpMonoFillColor*, *mpMonoFillPattern*, and *mpMonoFillScale*, specify use of the scalar resources when set True, and the array resources when set False. Note that if a *Mono* resource is set True, the named alias resources applying to that attribute are ignored, since they are, after all, only aliases for particular elements of the array resource.

For each attribute that has the *Mono* resource set False, **MapPlot** finds a group number for each area either from the fixed area group array or the dynamic area group array and uses that number as an index into the appropriate array resource. How does **MapPlot** decide when to index based on the fixed area group number and when to index based on the dynamic area group number? For fill areas specified using only the *mpFillBoundarySets* resource, **MapPlot** chooses between the area group arrays as follows:

`NoBoundaries`
: Neither area group array is used. Each area is assigned fill attributes using the *default* group number (0) as an index.

`Geophysical`
: Each area is assigned fill attributes using its *mpFixedAreaGroups* group numbers as an index.

`National`
: Each area is assigned fill attributes using its *mpDynamicAreaGroups* group number as an index.

`USStates`
: If the area belongs within the USStates dataset, it is assigned fill attributes using its *mpDynamicAreaGroups* group number as an index. Otherwise it is assigned fill attributes using the *default* group number (0) as an index.

`GeophysicalAndUSStates`
: If the area belongs within the USStates dataset it is assigned fill attributes using its *mpDynamicAreaGroups* group number as an index. Otherwise it is assigned fill attributes using its *mpFixedAreaGroups* group number as an index.

`AllBoundaries`
: Each area is assigned fill attributes using its *mpDynamicAreaGroups* group number as an index.

For areas specified in the *mpFillAreaSpecifiers* array, you may explicitly set the fill attributes individually as explained in the next section. However, whenever a fill attribute is not explicitly set, **MapPlot** uses a group number either from the fixed or the dynamic area group arrays as an index into the fill attribute array resources.

When using the Ncarg4_0 database:

In this case, the choice is based on the setting of the *mpSpecifiedFillPriority* resource. If *mpSpecifiedFillPriority* has the value `GeophysicalPriority`, **MapPlot** picks an index from the *mpFixedAreaGroups* array to determine fill attributes. Otherwise, it picks an index from the *mpDynamicAreaGroups* array. You may reverse the fill priority for individual areas within the *mpFillAreaSpecifiers* array by prepending the exclamation mark character ('!') to the area's specifier string. For instance, if *mpSpecifiedFillPriority* is set to `PoliticalPriority` but you want Australia to be displayed using its geophysical color, you would set the appropriate *mpFillAreaSpecifiers* element to the value "!australia".

When using the Ncarg4_1 database:

MapPlot picks an index from the *mpFixedAreaGroups* if the area's type is 1 (mpGeophysical) or if its type is 2 (mpContinental) and its parent is Water. Otherwise it picks an index from the *mpDynamicAreaGroups*.

Explicitly setting fill attributes

You can explicitly set the fill attributes of areas specified in the *mpFillAreaSpecifiers* array using the array resources *mpSpecifiedFillColors*, *mpSpecifiedFillPatterns*, and *mpSpecifiedFillScales*. Each element of the specified fill attribute arrays provide an attribute value for the corresponding element of the *mpFillAreaSpecifiers* array. Explicit specification of fill attributes overrides fill attribute selection based on area group memberships. Explicit specification also overrides the *Mono* flag. Therefore, you can easily highlight an individual area by explicitly setting, for example, the appropriate element of *mpSpecifiedFillPatterns* to a unique pattern while the remaining areas are all set perhaps to `SolidFill`, as determined by the value of the *mpFillPattern* resource.

If you want to explicitly specify the attributes of only certain of the areas specified in the *mpFillAreaSpecifiers*, while others base their fill attributes on area group memberships as discussed in the previous section, there are special values you can assign to indicate an "unset" value. For *mpSpecifiedFillColors* use the special color index `UnspecifiedColor` (-2); for *mpSpecifiedFillPatterns* the special fill pattern index `UnspecifiedFill` (-2); and for *mpSpecifiedFillScales* use the otherwise invalid value, 0.0.

By default, the specified fill attribute arrays directly set the attribute. In other words, the value of an element of *mpSpecifiedFillColors* represents an HLU color index, the value of an element of *mpSpecifiedFillPatterns* represents an HLU pattern index, and the value of an element of *mpSpecifiedFillScales* is a floating point number representing a fill scaling factor. However, if you set the resource *mpSpecifiedFillDirectIndexing* False, the values of these array elements are converted to integers and treated as group numbers. This facility allows you to use the specified fill attribute arrays to specify the fill attributes of an area indirectly by temporarily changing its group membership.

Area masking

MapPlot allows you to specify areas or groups of areas that are to remain unfilled or *masked*. You enable and disable area masking using the resource, *mpAreaMaskingOn*. You specify individual areas or area categories that you want to mask using the string array resource, *mpMaskAreaSpecifiers*.

Masking and fill precedence

Since there is a potential for fill and masking specifications to conflict with each other, **MapPlot** has established an order of precedence for these specifiers. Explicitly named areas take precedence over area groups specified using *mpFillBoundarySets* or any of the pre-defined constants. Small areas take precedence over enclosing larger areas. Otherwise masking takes precedence over filling. Note that if the *mpFillBoundarySets* resource is set to the value `Noboundaries`, it is possible that areas within the map projection may remain unspecified either as filled or masked. **MapPlot** *fills* these areas using the *default* fill attributes.

Grid and limb line

By default **MapPlot** turns on the map grid and, where appropriate, the limb line; you can turn it off by setting the resource *mpGridAndLimbOn* False. Set the resource *mpGridSpacingF* to specify a uniform distance in degrees between grid lines both for longitude and for latitude. If you want different grid spacings for the latitude and the longitude, set the *mpGridLonSpacingF* and *mpGridLatSpacingF* resources instead. You can also exercise detailed control over the appearance of the grid in the polar regions using the *mpGridMaxLatF* and *mpGridPolarLonSpacingF* resources.

You can turn on grid masking and choose how to select the areas where grid lines do not appear with the NhlTMapGridMaskMode resource *mpGridMaskMode*. Note that grid masking operates independently of area masking, although it is possible to mask the grid over masked areas if you set *mpGridMaskMode* to `MaskMaskArea`.

You can set the line attributes for the grid and limb lines independently. Grid attributes are controlled using the resources *mpGridLineColor*, *mpGridLineDashPattern*, *mpGridLineDashSegLenF*, and *mpGridLineThicknessF*. You set limb line attributes using *mpLimbLineColor*, *mpLimbLineDashPattern*, *mpLimbLineDashSegLenF*, and *mpLimbLineThicknessF*.

Labels

MapPlot draws labels by default, but you can turn them off by setting the *mpLabelsOn* resource False. When labels are turned on, the text strings "NP", "SP", "EQ", "GM", and "ID" are written at the positions of the North Pole, the South Pole, the Equator, the Greenwich Meridian, and the International Dateline. You can control the size of labels using the resources *mpLabelFontHeightF* and their color using *mpLabelFontColor*. Currently, no other resources are available for labels.

Perimeter

By default **MapPlot** draws a rectangular perimeter around the projected map area, but you can turn it off by setting the boolean resource *mpPerimOn* False. If you set the **MapTransformation** resource *mpEllipticalBoundary* True, the perimeter will be elliptical instead of rectangular. You can set line attributes for the perimeter using the resources *mpPerimLineColor*, *mpPerimLineDashPattern*, *mpPerimLineDashSegLenF*, and *mpPerimLineThicknessF*.

Support functions

The **MapPlot** object does not define any support functions, but inherits all the support functions available to its **Transform** superclass.

Status

1. **MapPlot** does not yet support tickmarks since the `Geographic` style has not yet been implemented for the **TickMark** class.

2. **MapPlot** does not support substring matching when using the new `Ncarg4_1` map database. Due to the hierarchical nature of the new database, this capability is not as necessary as with the old database. Expect that if and when new matching facilities are developed, they will not work the same way as they do for the old database.

3. Eventually you should be able to add groups of points that define your own areas, give them arbitrary names and add these areas to the area names array.

See also

- NhlCreate function

- NhlSetValues function
- NhlGetValues function
- MapTransformation object
- PlotManager object
- Title object
- LabelBar object
- Legend object
- Transform object
- View object

Copyright

Copyright 1987-1999 University Corporation for Atmospheric Research
The use of this Software is governed by a License Agreement.

NCAR Graphics is a registered trademark of the University Corporation for Atmospheric Research.

MapTransformation class

The **MapTransformation** manages forward and reverse transformations in the coordinate space defined by various map projections.

Synopsis

```
Header file:             ncarg/hlu/MapTransformation.h
Class name:              mapTransformationClass
Class pointer:           <Not referenceable>
Fortran class function:  <Not referenceable>
Supeclass:               Transformation
Composite classes:       <None>
```

Class-defined types

```
Type name:               NhlTMapLimitMode
Definition:
typedef enum _NhlMapLimitMode {
        NhlMAXIMALAREA  = 0,    /* "MaximalArea" */
        NhlLATLON       = 1,    /* "LatLon"      */
        NhlANGLES       = 2,    /* "Angles"      */
        NhlNPC          = 3,    /* "NPC"         */
        NhlNDC          = 4,    /* "NDC"         */
        NhlCORNERS      = 5,    /* "Corners"     */
        NhlPOINTS       = 6,    /* "Points"      */
        NhlWINDOW       = 7     /* "Window"      */
} NhlMapLimitMode;
```

```
Type name:              NhlTProjection
Definition:
typedef enum _NhlProjection {
        NhlORTHOGRAPHIC           = 0, /* "Orthographic"           */
        NhlSTEREOGRAPHIC          = 1, /* "Stereographic"          */
        NhlLAMBERTEQUALAREA       = 2, /* "LambertEqualArea"       */
        NhlGNOMONIC               = 3, /* "Gnomonic"               */
        NhlAZIMUTHALEQUIDISTANT   = 4, /* "AzimuthalEquidistant"   */
        NhlSATELLITE              = 5, /* "Satellite"              */
        NhlMOLLWEIDE              = 6, /* "Mollweide"              */
        NhlMERCATOR               = 7, /* "Mercator"               */
        NhlCYLINDRICALEQUIDISTANT = 8, /* "CylindricalEquidistant" */
        NhlLAMBERTCONFORMAL       = 9  /* "LambertConformal"       */
} NhlProjection;
```

Resources

Local resources

You may access **MapTransformation** resources only through objects that instantiate a **MapTransformation** child object. These include:

- **MapPlot** object

```
+---------------------------------------------------------------+
|                     MapTransformation Resource Set            |
|---------------------------------------------------------------|
| NAME                         TYPE                    ACCESS   |
|       CLASS                          DEFAULT                  |
|===============================================================|
| mpProjection                 NhlTProjection          RCSG     |
|       MpProjection                 "CylindricalEquidistant"   |
|---------------------------------------------------------------|
| mpLeftMapPosF                NhlTFloat               G        |
|       MpLeftMapPosF                  <dynamic>                |
|---------------------------------------------------------------|
| mpRightMapPosF               NhlTFloat               G        |
|       MpRightMapPosF                 <dynamic>                |
|---------------------------------------------------------------|
| mpBottomMapPosF              NhlTFloat               G        |
|       MpBottomMapPosF                <dynamic>                |
|---------------------------------------------------------------|
| mpTopMapPosF                 NhlTFloat               G        |
|       MpTopMapPosF                   <dynamic>                |
|---------------------------------------------------------------|
| mpCenterLatF                 NhlTFloat               RCSG     |
|       MpCenterLatF                   0.0                      |
|---------------------------------------------------------------|
| mpCenterLonF                 NhlTFloat               RCSG     |
|       MpCenterLonF                   0.0                      |
|---------------------------------------------------------------|
| mpCenterRotF                 NhlTFloat               RCSG     |
|       MpCenterRotF                   0.0                      |
|---------------------------------------------------------------|
| mpLimitMode                  NhlTMapLimitMode        RCSG     |
```

```
      MpLimitMode                       "MaximalArea"
---------------------------------------------------------------
mpMinLatF                     NhlTFloat                RCSG
      MpMinLatF                         -90.0
---------------------------------------------------------------
mpMaxLatF                     NhlTFloat                RCSG
      MpMaxLatF                         90.0
---------------------------------------------------------------
mpMinLonF                     NhlTFloat                RCSG
      MpMinLonF                         -180.0
---------------------------------------------------------------
mpMaxLonF                     NhlTFloat                RCSG
      MpMaxLonF                         180.0
---------------------------------------------------------------
mpRelativeCenterLat           NhlTBoolean              RCSG
      MpRelativeCenterLat               False
---------------------------------------------------------------
mpRelativeCenterLon           NhlTBoolean              RCSG
      MpRelativeCenterLon               False
---------------------------------------------------------------
mpLeftAngleF                  NhlTFloat                RCSG
      MpLeftAngleF                      80.0
---------------------------------------------------------------
mpRightAngleF                 NhlTFloat                RCSG
      MpRightAngleF                     80.0
---------------------------------------------------------------
mpBottomAngleF                NhlTFloat                RCSG
      MpBottomAngleF                    80.0
---------------------------------------------------------------
mpTopAngleF                   NhlTFloat                RCSG
      MpTopAngleF                       80.0
---------------------------------------------------------------
mpLeftNPCF                    NhlTFloat                RCSG
      MpLeftNPCF                        0.0
---------------------------------------------------------------
mpRightNPCF                   NhlTFloat                RCSG
      MpRightNPCF                       1.0
---------------------------------------------------------------
mpBottomNPCF                  NhlTFloat                RCSG
      MpBottomNPCF                      0.0
---------------------------------------------------------------
mpTopNPCF                     NhlTFloat                RCSG
      MpTopNPCF                         1.0
---------------------------------------------------------------
mpLeftNDCF                    NhlTFloat                RCSG
      MpLeftNDCF                        <dynamic>
---------------------------------------------------------------
mpRightNDCF                   NhlTFloat                RCSG
      MpRightNDCF                       <dynamic>
---------------------------------------------------------------
mpBottomNDCF                  NhlTFloat                RCSG
      MpBottomNDCF                      <dynamic>
---------------------------------------------------------------
mpTopNDCF                     NhlTFloat                RCSG
      MpTopNDCF                         <dynamic>
---------------------------------------------------------------
mpLeftCornerLatF              NhlTFloat                RCSG
      MpLeftCornerLatF                  0.0
---------------------------------------------------------------
mpLeftCornerLonF              NhlTFloat                RCSG
      MpLeftCornerLonF                  0.0
```

mpRightCornerLatF MpRightCornerLatF	NhlTFloat	0.0	RCSG
mpRightCornerLonF MpRightCornerLonF	NhlTFloat	0.0	RCSG
mpLeftPointLatF MpLeftPointLatF	NhlTFloat	0.0	RCSG
mpLeftPointLonF MpLeftPointLonF	NhlTFloat	0.0	RCSG
mpRightPointLatF MpRightPointLatF	NhlTFloat	0.0	RCSG
mpRightPointLonF MpRightPointLonF	NhlTFloat	0.0	RCSG
mpBottomPointLatF MpBottomPointLatF	NhlTFloat	0.0	RCSG
mpBottomPointLonF MpBottomPointLonF	NhlTFloat	0.0	RCSG
mpTopPointLatF MpTopPointLatF	NhlTFloat	0.0	RCSG
mpTopPointLonF MpTopPointLonF	NhlTFloat	0.0	RCSG
mpLeftWindowF MpLeftWindowF	NhlTFloat	0.0	RCSG
mpRightWindowF MpRightWindowF	NhlTFloat	0.0	RCSG
mpBottomWindowF MpBottomWindowF	NhlTFloat	0.0	RCSG
mpTopWindowF MpTopWindowF	NhlTFloat	0.0	RCSG
mpLambertParallel1F MpLambertParallel1F	NhlTFloat	.001	RCSG
mpLambertParallel2F MpLambertParallel2F	NhlTFloat	89.999	RCSG
mpLambertMeridianF MpLambertMeridianF	NhlTFloat	0.0	RCSG
mpSatelliteDistF MpSatelliteDistF	NhlTFloat	1.0	RCSG
mpSatelliteAngle1F MpSatelliteAngle1F	NhlTFloat	0.0	RCSG
mpSatelliteAngle2F MpSatelliteAngle2F	NhlTFloat	0.0	RCSG

```
| mpEllipticalBoundary          NhlTBoolean             RCSG    |
|       MpEllipticalBoundary            False                   |
|---------------------------------------------------------------|
| mpGreatCircleLinesOn          NhlTBoolean             RCSG    |
|       MpGreatCircleLinesOn            False                   |
+---------------------------------------------------------------+
```

Composite resources

The **MapTransformation** class has no composite class objects.

Superclass resources

MapTransformation disables access to all the resources of its **Transformation** superclass.

Description

The **MapTransformation** object manages forward and reverse transformations between a two-dimensional rectangular data coordinate space representing latitude along the vertical axis and longitude along the horizontal axis and any of its 10 supported map projections. The data space extent may coincide with, encompass, or arbitrarily intersect the projected extent.

Creation and access to resources

You do not create objects of the **MapTransformation** class directly. You may, however, set its resources via objects that instantiate a **MapTransformation** child. Currently, the **MapPlot** object is the only HLU library object with a **MapTransformation** child; it allows access all resources of the **MapTransformation** class, depending on it to provide transformations of data representing geophysical and political regions of the earth. Using the **MapPlot** as a base plot, you can provide map transformations for data associated with other plot object types. In addition, you can transform graphics primitives defined in lat-lon data space into any of the map projections.

Setting the projection and map boundaries

You define a map transformation by setting the *mpProjection* resource, and then in most cases, choosing a center of projection and a rotation angle using the resources, *mpCenterLatF*, *mpCenterLonF*, and *mpCenterRotF*. Then you choose a method for defining the visible extent of the global surface using the *mpLimitMode* resource. This resource has eight possible settings:

- `MaximalArea` mode shows the maximum possible visible area depending on the projection. The **MapTransformation** ignores all other extent limiting resources when this mode is set.
- `LatLon` mode defines a rectangular area bounded by minimum and maximum latitudes and longitudes, and shows as much of this area as possible given the projection and the current projection center. Set the latitude boundaries using *mpMinLatF* and *mpMaxLatF* and the longitude boundaries using *mpMinLonF* and *mpMaxLonF*. Two other resources apply only when using `LatLon` limit mode. These are the boolean resources *mpRelativeCenterLon* and *mpRelativeCenterLat*. When *mpRelativeCenterLon* is set True the value of the *mpCenterLonF*

resource is interpreted as an offset in degrees from the longitude halfway between *mpMinLonF* and *mpMaxLonF*. Setting *mpRelativeCenterLat* True similarly modifies the interpretation of the *mpCenterLatF* resource.

- `Angles` mode limits the area based on positive angular distance in degrees away from the projection center in the four directions parallel to a side of the viewport (*not necessarily* parallel to lines of latitude and longitude). Use the resources *mpLeftAngleF*, *mpRightAngleF*, *mpBottomAngleF*, and *mpTopAngleF* to set angular limits. You cannot use this mode when the conic projection, `LambertConformal`, is set. Also, if *mpProjection* has the value `Satellite`, the **MapTransformation** interprets the angles as deviations from the line of sight of the viewing satellite.
- `NPC` mode limits the area based on "Normalized Projection Coordinates", which map the maximal area of the projection into a rectangle bounded by 0.0 and 1.0 horizontally and vertically. Use the resources *mpLeftNPCF*, *mpRightNPCF*, *mpBottomNPCF*, and *mpTopNPCF* to set normalized projection coordinate limits. Note that the **MapTransformation** keeps the NPC resources updated to their current value no matter which map limit mode is currently in effect.
- `NDC` mode limits the area based on the current location of the projected map in Normalized Device Coordinates. Use the resources *mpLeftNDCF*, *mpRightNDCF*, *mpBottomNDCF*, and *mpTopNDCF* to set normalized device coordinate limits. Note that each time these resources are applied using a SetValues call the mapping between NDC and the projected area changes. At the completion of a SetValues call, the **MapTransformation** internally sets the value of each of these resources to the new location of the original projected coordinate: that is, to one of the boundaries of the projected area.
- `Corners` mode limits the area to a rectangle with corners defined by the values of the resources *mpLeftCornerLatF*, *mpLeftCornerLonF*, *mpRightCornerLatF*, and *mpLeftCornerLonF*.
- `Points` mode limits the area to a rectangle containing the four points defined by the values of the resources *mpLeftPointLatF*, *mpLeftPointLonF*, *mpRightPointLatF*, *mpRightPointLonF*, *mpBottomPointLatF*, *mpBottomPointLonF*, *mpTopPointLatF*, and *mpTopPointLonF*
- `Window` mode limits the area based on the intermediate "window" coordinate system used by the low level utilities. You set window coordinate limits using the resources *mpLeftWindowF*, *mpRightWindowF*, *mpBottomWindowF*, and *mpTopWindowF*. The **MapTransformation** keeps the window coordinate system resources updated to their current values no matter which map limit mode is currently in effect.

`LambertConformal` projection resources

When the conic projection `LambertConformal` is set, the **MapTransformation** ignores the projection center and rotation resources. Instead, use *mpLambertParallel1F* and *mpLambertParallel2F* to set the two latitudinal edges of the cone, and use *mpLambertMeridianF* to set its longitudinal center.

`Satellite` projection resources

If the projection is `Satellite`, the **MapTransformation** provides several additional resources for controlling the perspective effect. *mpSatelliteDistF* defines the distance of the viewing satellite from the center of the globe in multiples of the global radius. *mpSatelliteAngle1F* defines the angular deviation of the line of sight from a line between the satellite position and the center of the globe. *mpSatelliteAngle2F* defines the direction of the angular deviation of the line of sight as a rotation from the horizontal axis of the viewplane.

Elliptical boundary

For any of the projections, you can set a resource, *mpEllipticalBoundary*, that limits the perimeter of the viewable area to an ellipse inscribed within the rectangular viewport.

Graphic primitive transformations

MapTransformation supports transformation into any of the map projections of graphic primitives using the data space primitive drawing routines NhlDataPolygon, NhlDataPolyline, and NhlDataPolymarker. Depending on the setting of the boolean resource *mpGreatCircleLinesOn*, lines or edges spanning the space between the user-supplied points of a polyline or polygon may be transformed in two different ways. One treats the lat-lon grid as a cartesian system; the other calculates the great circle route (the shortest distance on the surface of the globe) between each set of points.

Support functions

The **MapTransformation** object does not define any support functions, but inherits all the support functions available to its superclass.

Status

2. Range checking is inadequate for a number of the **MapTransformation** resources including *mpLambertMeridianF*, *mpLambertParallel2F*, *mpLambertParallel1F*, *mpCenterLatF*, and *mpCenterLonF*.

3. When *mpCenterRotF* has a non-zero value and *mpMapLimitMode* is set to NhlANGLES, it is possible that errors may occur even when the angle values are within the documented limits. This occurs in certain situations that have so far proved to be difficult to pin down. More work is required.

See also

- MapPlot object
- PlotManager object
- Transformation object
- Obj object

Copyright

Copyright 1987-1999 University Corporation for Atmospheric Research
The use of this Software is governed by a License Agreement.

NCAR Graphics is a registered trademark of the University Corporation for Atmospheric Research.

NcgmWorkstation class

The **NcgmWorkstation** manages communication with the NCAR Graphcs CGM Workstation output device.

Synopsis

```
Header file:              ncarg/hlu/NcgmWorkstation.h
Class name:               ncgmWorkstationClass
Class pointer:            NhlncgmWorkstationClass
Fortran class function: NHLFNCGMWORKSTATIONCLASS
Superclass:               Workstation
Composite classes:        <None>
```

Resources

Local resources

```
+---------------------------------------------------------------+
|                   NcgmWorkstation Resource Set                |
|---------------------------------------------------------------|
| NAME                          TYPE                    ACCESS  |
|       CLASS                           DEFAULT                 |
|===============================================================|
| wkMetaName                    NhlTString              RCG     |
|       WkMetaName                      "gmeta"                 |
+---------------------------------------------------------------+
```

Composite resources

The **NcgmWorkstation** object class has no composite class objects.

Superclass resources

You can set all resources defined by the superclasses of the **NcgmWorkstation** object class, including:

- **Workstation** resources

Description

The **NcgmWorkstation** object is derived from the **Workstation** object class. It inherits all of the color map management and workstation management functions of the **Workstation** class. The **NcgmWorkstation** opens and creates an output metafile. Whenever a child of the workstation is drawn, all output is written into the output metafile. The output metafile can be viewed using standard NCAR Graphics CGM viewing tools. The output metafile can also be translated into many other device formats using the NCAR Graphics application called **ctrans**.

This class adds only one resource to the **Workstation** set of resources. This resource enables the user to specify a full Unix path name for the output metafile.

An important thing to remember is that only one **NcgmWorkstation** object can be created at a time. Therefore, you must destroy the current instance of the **NcgmWorkstation** object before creating a new one. You can use the **NhlDestroy** function for this purpose.

Another thing to remember with the **NcgmWorkstation** object, is that the metafile is written using FORTRAN I/O. This means that the **NcgmWorkstation** object uses one of the available UNIT numbers. The **NcgmWorkstation** starts with UNIT 7, and increments until it finds a UNIT number that is not currently in use.

Support functions

The **NcgmWorkstation** object does not define any support functions, but inherits all the support functions available to its superclasses. This includes all the **Workstation** class functions.

See also

- NhlCreate function
- NhlSetValues function
- NhlDestroy function
- NhlClearWorkstation function
- NhlFrame function
- Workstation object
- Base object

Copyright

Copyright 1987-1999 University Corporation for Atmospheric Research
The use of this Software is governed by a License Agreement.

NCAR Graphics is a registered trademark of the University Corporation for Atmospheric Research.

Obj class

The **Obj** class is a simplified version of the **Base** class.

Synopsis

```
Header file:            ncarg/hlu/Base.h
Class fame:             objClass
Class pointer:          <Not referenceable>
Fortran class function: <Not referenceable>
Superclass:             <NONE>
Composite classes:      <NONE>
```

Resources

The Obj class has no public resources.

Description

The **Obj** is a simplified form of the **Base** class. Classes that are subclassed from **Obj** as opposed to **Base** do not have access to the **NhlDraw** support function. Otherwise, the **Base** and **Obj** classes should be considered equivalent from the API.

Support functions

All the **Base** class support functions are available with the exception of **NhlDraw**.

See also

- Base object

Copyright

Copyright 1987-1999 University Corporation for Atmospheric Research
The use of this Software is governed by a License Agreement.

NCAR Graphics is a registered trademark of the University Corporation for Atmospheric Research.

PSWorkstation class

The **PSWorkstation** object manages communication with the NCAR Graphics PS Workstation output device.

Synopsis

```
Header file:              ncarg/hlu/PSWorkstation.h
Class name:               psWorkstationClass
Class pointer:            NhlpsWorkstationClass
Fortran class function:   NHLFPSWORKSTATIONCLASS
Superclass:               Workstation
Composite classes:        <None>
```

Class-defined types

```
Type name:                NhlTPSFormat
Definition:               typedef enum {
                                  NhlPS           = 0,    /* "ps"         */
                                  NhlEPS          = 1,    /* "eps"        */
                                  NhlEPSI         = 2     /* "epsi"       */
                          } NhlPSFormat;

Type name:                NhlTVisualType
Definition:               typedef enum {
                                  NhlCOLOR        = 0,    /* "color"      */
                                  NhlMONOCHROME   = 3     /* "monochrome" */
                          } NhlVisualType;

Type name:                NhlTColorModel
Definition:               typedef enum {
                                  NhlCMYK         = 0,    /* "cmyk"       */
                                  NhlRGB          = 1     /* "rgb"        */
                          } NhlColorModel;

Type name:                NhlTWorkOrientation
Definition:               typedef enum {
                                  NhlPORTRAIT     = 0,    /* "portrait"   */
                                  NhlLANDSCAPE    = 6     /* "landscape"  */
                          } NhlWorkOrientation;
```

Resources

Local resources

```
+---------------------------------------------------------------+
|                       PSWorkstation resource set              |
|---------------------------------------------------------------|
| NAME                            TYPE                    ACCESS|
|       CLASS                             DEFAULT               |
```

```
|=============================================================|
| wkPSFormat                    NhlTPSFormat           RCG    |
|       WkPSFormat                     "ps"                   |
|-------------------------------------------------------------|
| wkVisualType                  NhlTVisualType         RCG    |
|       WkVisualType                   "color"                |
|-------------------------------------------------------------|
| wkColorModel                  NhlTColorModel         RCG    |
|       WkColorModel                   "rgb"                  |
|-------------------------------------------------------------|
| wkOrientation                 NhlTWorkOrientation    RCG    |
|       WkOrientation                  "portrait"             |
|-------------------------------------------------------------|
| wkPSFileName                  NhlTString             RCG    |
|       WkPSFileName                   <dynamic>              |
|-------------------------------------------------------------|
| wkFullBackground              NhlTBoolean            RCSG   |
|       WkPSFileName                   False                  |
|-------------------------------------------------------------|
| wkPSResolution                NhlTInteger            RCG    |
|       WkPSResolution                 1800                   |
|-------------------------------------------------------------|
| wkDeviceLowerX                NhlTInteger            RCG    |
|       WkDeviceLowerX                 36                     |
|-------------------------------------------------------------|
| wkDeviceLowerY                NhlTInteger            RCG    |
|       WkDeviceLowerY                 126                    |
|-------------------------------------------------------------|
| wkDeviceUpperX                NhlTInteger            RCG    |
|       WkDeviceUpperX                 576                    |
|-------------------------------------------------------------|
| wkDeviceUpperY                NhlTInteger            RCG    |
|       WkDeviceUpperY                 666                    |
+-------------------------------------------------------------+
```

Composite resources

The **PSWorkstation** object class has no composite class members.

Superclass resources

You can set all resources defined by the superclasses of the **PSWorkstation** object class, including:

- **Workstation** resources

Description

The **PSWorkstation** object is derived from the **Workstation** class and thus inherits all of the color map management and control functions associated with the **Workstation** class.

[More description to be written ASAP...]

Support functions

The **PSWorkstation** object does not define any support functions; it inherits all the support functions available to its superclass.

Status

See also

- NhlUpdateWorkstation function
- NhlClearWorkstation function
- Frame function
- Workstation object

Copyright

Copyright 1987-1999 University Corporation for Atmospheric Research
The use of this Software is governed by a License Agreement.

NCAR Graphics is a registered trademark of the University Corporation for Atmospheric Research.

PlotManager class

The **PlotManager** allows a plot object to manage overlays and annotations.

Synopsis

```
Header file:             ncarg/hlu/PlotManager.h
Class name:              plotManagerClass
Class pointer:           <Not referenceable>
Fortran class function:  <Not referenceable>
Superclass:              Transform
Composite classes:       LabelBar,Legend,TickMark,Title
```

Class-defined types

```
Type name:               NhlTAnnotationDisplayMode
Definition:
typedef enum _NhlAnnotationDisplayMode {
```

```
        NhlNOCREATE     = -1,   /* "NoCreate"    */
        NhlNEVER        = 0,    /* "Never"       */
        NhlALWAYS       = 1,    /* "Always"      */
        NhlCONDITIONAL  = 2     /* "Conditional" */
} NhlAnnotationDisplayMode;
```

Resources

Local resources

PlotManager resources are available to objects of the **Transform** class that include **PlotManager** as a composite class member, providing that the **Transform** class resource *tfPlotManagerOn* is set to its default value, True, when the object is created. Currently, there are five composite classes that include the **PlotManager**:

- **ContourPlot** object
- **XyPlot** object
- **IrregularPlot** object
- **LogLinPlot** object
- **MapPlot** object

```
+---------------------------------------------------------------+
|               PlotManager Resource Set                        |
|---------------------------------------------------------------|
| NAME                          TYPE                     ACCESS |
|       CLASS                           DEFAULT                  |
|===============================================================|
| pmOverlaySequenceIds          NhlTObjIdGenArray        G      |
|       PmOverlaySequenceIds            NULL                     |
|---------------------------------------------------------------|
| pmAnnoViews                   NhlTObjIdGenArray        CSG    |
|       PmAnnoViews                     NULL                     |
|---------------------------------------------------------------|
| pmAnnoManagers                NhlTObjIdGenArray        G      |
|       PmAnnoManagers                  NULL                     |
|---------------------------------------------------------------|
| pmTitleDisplayMode            NhlTAnnotationDisplayMode RCSG  |
|       PmDisplayTitles                 NoCreate                 |
|---------------------------------------------------------------|
| pmTitleZone                   NhlTInteger              RCSG   |
|       PmTitleZone                     4                        |
|---------------------------------------------------------------|
| pmTickMarkDisplayMode         NhlTAnnotationDisplayMode RCSG  |
|       PmDisplayTickMarks              NoCreate                 |
|---------------------------------------------------------------|
| pmTickMarkZone                NhlTInteger              RCSG   |
|       PmTickMarkZone                  2                        |
|---------------------------------------------------------------|
| pmLabelBarDisplayMode         NhlTAnnotationDisplayMode RCSG  |
|       PmDisplayLabelBar               NoCreate                 |
|---------------------------------------------------------------|
| pmLabelBarWidthF              NhlTFloat                RCSG   |
|       PmLabelBarWidthF                0.15                     |
```

pmLabelBarHeightF	NhlTFloat	RCSG
PmLabelBarHeightF	0.6	
pmLabelBarZone	NhlTInteger	RCSG
PmLabelBarZone	6	
pmLabelBarSide	NhlTJustification	RCSG
PmLabelBarSide	Right	
pmLabelBarParallelPosF	NhlTFloat	RCSG
PmLabelBarParallelPosF	0.5	
pmLabelBarOrthogonalPosF	NhlTFloat	RCSG
PmLabelBarOrthogonalPosF	0.02	
pmLegendDisplayMode	NhlTAnnotationDisplayMode	RCSG
PmDisplayLegend	NoCreate	
pmLegendWidthF	NhlTFloat	RCSG
PmLegendWidthF	0.55	
pmLegendHeightF	NhlTFloat	RCSG
PmLegendHeightF	0.18	
pmLegendZone	NhlTInteger	RCSG
PmLegendZone	7	
pmLegendSide	NhlTPosition	RCSG
PmLegendSide	Bottom	
pmLegendParallelPosF	NhlTFloat	RCSG
PmLegendParallelPosF	0.5	
pmLegendOrthogonalPosF	NhlTFloat	RCSG
PmLegendOrthogonalPosF	0.02	

Composite resources

TickMark resources

If *pmTickMarkDisplayMode* has the value `NoCreate` when a **PlotManager** composite object is created, no **TickMark** resources are available. Otherwise, you can set all **TickMark** resources for **PlotManager** composite objects except for the following resources that **PlotManager** disables:

- tmXBDataLeftF
- tmXBDataRightF
- tmXBStyle
- tmXBIrregularPoints
- tmXBIrrTensionF
- tmXTDataLeftF
- tmXTDataRightF
- tmXTStyle
- tmXTIrregularPoints
- tmXTIrrTensionF

- tmYLDataBottomF
- tmYLDataTopF
- tmYLStyle
- tmYLIrregularPoints
- tmYLIrrTensionF
- tmYRDataBottomF
- tmYRDataTopF
- tmYRStyle
- tmYRIrregularPoints
- tmYRIrrTensionF

PlotManager sets these resources itself based on its knowledge of the current transformation. Note that the **TickMark** object does not currently support ticks for Map transformations.

Title resources

If *pmTitleDisplayMode* has the value `NoCreate` when a **PlotManager** composite object is created, no **Title** resources are available. Otherwise, you can set all **Title** resources for **PlotManager** composite objects. However, note that **PlotManager** intercepts three **Title** resources, as follows:

- tiMainFontHeightF
    PlotManager sets the default value of *tiMainFontHeightF* dynamically based on the ratio of the actual plot viewport width to a reference viewport width of 0.6 and a reference font height of 0.025.
- tiXAxisFontHeightF
    PlotManager sets the default value of *tiXAxisFontHeightF* dynamically based on the ratio of the actual plot viewport width to a reference viewport width of 0.6 and a reference font height of 0.025.
- tiYAxisFontHeightF
    PlotManager sets the default value of *tiYAxisFontHeightF* dynamically based on the ratio of the actual plot viewport height to a reference viewport height of 0.6 and a reference font height of 0.025.

LabelBar resources

If *pmLabelBarDisplayMode* has the value `NoCreate` when a **PlotManager** composite object is created, no **LabelBar** resources are available. Otherwise, you can set all **LabelBar** resources for **PlotManager** composite objects, except for the following disabled resource:

- lbLabelBarOn

PlotManager intercepts two **LabelBar** resources, as follows:

- lbJustification
    PlotManager sets the initial default value of *lbJustification* to `CenterCenter`. It then constrains the value of this resource using the rules of the PlotManager Location Control Model.
- lbOrientation

At initialization and whenever the value of *lbOrientation* changes, **PlotManager** checks the currently set values of *pmLabelBarWidthF* and *pmLabelBarHeightF*. If neither resource is explicitly set, it swaps their values if the longest dimension is not currently parallel to the orientation.

Legend resources

If *pmLegendDisplayMode* has the value `NoCreate` when a **PlotManager** composite object is created, no **Legend** resources are available. Otherwise, you can set all **Legend** resources for **PlotManager** composite objects, except for the following disabled resource:

- lgLegendOn

PlotManager intercepts one **Legend** resource, as follows:

- lgJustification
  PlotManager sets the initial default value of *lgJustification* to `CenterCenter`. It then constrains the value of this resource using the rules of the PlotManager Location Control Model.

Superclass resources

You can set all resources defined by the superclasses of the **PlotManager** object class in any plot object containing a **PlotManager**, including:

- **Transform** resources
- **View** resources

Description

When an instance of the **Transform** class is instantiated with an active **PlotManager**, it becomes a *plot object*. A plot object can assume control of an arbitrary number of **View** objects, effectively organizing a collection of unrelated objects into a single plottable entity called simply a *plot*. The controlling plot object is known as the *base plot*; the objects it controls are *plot members*. When the base plot is drawn, all its plot members are drawn automatically. If you reposition the base plot, or resize it, or modify the data coordinate space in some way, all the plot members adjust themselves appropriately.

You never create an object belonging to the **PlotManager** class directly. Instead, the functionality of the **PlotManager** class becomes available to classes derived from **Transform** that include the **PlotManager** as a composite class member. By convention, these classes include the word *Plot* as part of their name. Currently, the following composite classes contain the **PlotManager** class: **ContourPlot**, **MapPlot**, **LogLinPlot**, **IrregularPlot**, **StreamlinePlot**, **VectorPlot**, and **XyPlot**. Each of these classes provides the capabilities of the **PlotManager** class by default, and therefore their instantiations are, by default, plot objects. Note that since all plot objects incorporate **PlotManager** functionality as an integral part of their behavior, it is equivalent -- when speaking of something that a plot object's

PlotManager does -- simply to say that the plot object does it.

You may turn off the **PlotManager** capabilities for a **Transform** object by setting the **Transform** class resource *tfPlotManagerOn* to False when the object is created. However, note that you cannot turn the object's **PlotManager** capabilities back on at a later time. An object created with its **PlotManager** disabled is *not* a plot object; instead it is called a *simple transform.* A simple transform can still function as a plot member.

There are two basic kinds of plot members: *overlays* and *annotations*. An overlay must be a *transform*, either a plot object (with or without its own plot members) or a simple transform. An annotation, on the other hand, can be any *view* from a simple **TextItem** to a plot object with its own plot members.

The base plot transforms an overlay's data coordinate space into its own coordinate space. Any portion of the overlay's coordinate space that lies outside the coordinate space of the base plot is clipped. The viewport of the overlay is modified to match that of the base plot. In addition, if the overlay is a plot object with plot members of its own, the base plot assumes responsibility for sizing and locating the overlay's plot members, just as if it owned them.

The base plot locates and sizes an annotation relative to its viewport or to its data coordinate space. Unlike an overlay, however, there is no fixed relationship between the base plot's data coordinate space and the annotation's data coordinate space. In fact, ordinary views, from which annotations are created, do not even have a data coordinate space. The viewport of an annotation may fall inside or outside the viewport of the base plot. If the annotation happens to be a plot object, it does *not* give up control of its own plot members to the base plot. Indeed, an *annotation plot* is itself a base plot. However, since an annotation is itself managed by a base plot, an annotation plot is a subordinate base plot. A subordinate base plot and the plot members it manages are known collectively as a *subplot*.

In contrast to a subordinate base plot, a base plot that manages itself (i.e. is not a plot member) is called a primary base plot. Note that all plot objects are, at creation, primary base plots. When a plot object becomes an overlay, it loses its base plot status. If it is added as an annotation, its status changes to subordinate base plot. The primary base plot controls the drawing of all its plot members, and you can only cause a plot member to be drawn by calling the **NhlDraw** function on the primary base plot. Attempting to draw any plot member (even a subordinate base plot) directly results in an error. Similarly, you cannot change the workstation of a plot member directly. When you change the workstation of the primary base plot, the workstation of all its plot members changes to match. Also, while an object is a plot member, you cannot add it to any other (or even the same) plot object, either as an overlay or as an annotation. If you need to do any of these things with a plot member, you must first remove it from the plot object that owns it.

Adding and removing overlays

You add the plot object that is to become an overlay to a base plot using the **NhlAddOverlay** function. The plot object must belong to the same **Workstation** as the base plot. You may call the function repeatedly to overlay multiple plot objects over a single base plot. Note that you can *only* add an overlay to a primary or subordinate base plot. Attempts to add an overlay to a plot that is itself an overlay will result in an error.

The overlays under the control of a single base plot are ordered, with the base plot first and each overlay following sequentially. The **NhlAddOverlay** function has a parameter that allows you to specify where to add a plot into the overlay sequence. This sequence determines the basic drawing order of the plot: the base plot is drawn first; each succeeding overlay is drawn on top of previous objects in the sequence. However, the **PlotManager** actually supports three drawing phases, known as the *predraw* phase, the *draw* phase, and the *postdraw* phase. Some plot objects, such as **ContourPlot** and **MapPlot** also support these drawing phases, allowing you to specify that certain elements of the plot, such as labels, fill, or lines, be rendered during a specific drawing phase. When the **PlotManager** executes a **Draw** command, it first performs the predraw phase for each plot object in the overlay sequence, then the draw, and finally the postdraw. By manipulating both the overlay sequence and the draw phase, you gain considerable flexibility in controlling which elements of the plot appear on top.

You can remove an overlay from a base plot by calling the function **NhlRemoveOverlay**. If the overlay you are removing is a plot object with overlays of its own, you can choose to remove them along with the specified overlay, or leave them behind with the base plot. Annotations originally belonging to the overlay are always removed along with it. If you call **NhlRemoveOverlay** on the base plot, the effect is to restore all its plot members to their original state before being added to the current base plot.

Adding and removing annotations

There are three ways for a **View** class object to become an annotation. First, it may be a composite class member of the **PlotManager** class, and therefore instantiated along with the **PlotManager** object itself when the plot object is created. The **PlotManager** class contains four composite classes that are used as annotations: **TickMark**, **Title**, **LabelBar**, and **Legend**. These are known as *intrinsic annotations*. Second, a plot object may create **View** objects of its own and then depend on the **PlotManager** to manage them as annotations. An example is the **ContourPlot** object's informational label. An annotation such as this is called an *embedded annotation*. Finally, you may create arbitrary **View** objects at the user level and add them as annotations to a plot object. These are called *external annotations*.

You may add a group of **View** objects as annotations to any plot object by setting the **PlotManager** array resource *pmAnnoViews* with the ids of the **View** objects. Annotations added in this manner replace any previously added external annotations. Alternatively, you can add a single **View** object as an annotation by calling the function **NhlAddAnnotation**. In this case, the **View** id is appended to the existing *pmAnnoViews* array. Either way, for each **View** object added, the **PlotManager** creates an **AnnoManager** object containing resources that allow all external annotations to be controlled in a uniform manner. Unlike overlays, you can add an annotation to any plot object, even if it is not currently a primary or subordinate base plot. The base plot that manages the plot object will immediately assume responsibility for locating and sizing the annotation.

If you add the annotation by calling **NhlAddAnnotation**, the id of the **AnnoManager** object created to control the view is the return value of the function. In any case, you can retrieve the id of the **AnnoManager** object by getting the value of the **PlotManager** array resource *pmAnnoManagers*. Each element of this array contains the id of the **AnnoManager** for the **View** object whose id is contained in the corresponding element of *pmAnnoViews*. Since the view has been informed of its status as an annotation, you can also retrieve the **AnnoManager** object id from the view itself, by getting the value of the resource *vpAnnoManagerId*. You may remove external annotations one at a time by calling the function **NhlRemoveAnnotation**. Note that the *pmAnnoViews* and *pmAnnoManagers* arrays *only* apply to external annotations (annotations added by the user). The **PlotManager** handles intrinsic and embedded annotations internally.

The **AnnoManager** object contains resources that allow you to fix the location and size of the annotation view relative to the viewport or the data coordinate space of the base plot. If the annotation is placed relative to the viewport, you use **AnnoManager** resources that conform to the *PlotManager Location Control Model*.

PlotManager Location Control Model

The **PlotManager** implements a scheme of *annotation zones* to simplify management of multiple annotations. Although at first glance it may seem somewhat complex, in practice it should be quite simple to use. You can place an annotation relative to any side of the plot viewport without reference to the viewport size or position. Simply by assigning different zones you can ensure (except under certain circumstances involving annotation plots, discussed below) that annotations do not overlap each other. And if you decide to change the viewport side where an annotation is located, you can generally expect that the new location will still look reasonable. Note, however, that the scheme does not currently prevent annotations from being located partially or completely outside the plot frame. If an annotation seems to have disappeared, you should check to make sure the current configuration has not caused it to appear outside the viewable area.

Standard zones

Zones are numbered beginning at 0, and you may use as many zones as necessary to locate annotations. Zones 0 and 1 have special characteristics that differ from the standard zones. However, note that for the purposes of establishing zonal boundaries, the bounding box of zone 1 is considered to be the plot viewport boundary. Beginning with zone 2, each zone extends outward from the bounding box of the previous zone. When the size and/or position of objects within a zone change, causing the zone's bounding box to expand or contract, the zones outside automatically adjust their shape and position to accommodate.

For an external annotation, you choose the zone by setting the **AnnoManager** resource, *amZone*. Intrinsic and embedded annotations that adhere to the *PlotManager Location Control Model* will have an equivalent resource with a name specific to the particular resource. Thus, the **PlotManager's** intrinsic **LabelBar** annotation has the resource *pmLabelBarZone*, and **ContourPlot** provides the resource *cnInfoLabelZone* for its embedded informational label annotation.

Once you have established the zone, you decide which side of the zone's interior boundary to use as a basis for determining the annotation's relative position. Here you use the **AnnoManager** resource *amSide* or some intrinsic or embedded equivalent. Selection of a side determines a coordinate system with one axis parallel to the side and another orthogonal to it. The origin of this system is on the side line at the point where the projection of the left or bottom edge of the viewport (whichever is perpendicular) crosses. The direction of the parallel axis is always toward increasing NDC values, while the direction of the orthogonal axis is outward from the center of the plot. You establish a position for the annotation relative to the zonal side origin by setting pallallel and orthogonal positional resources. For the **AnnoManager** object, these are *amParallelPosF* and *amOrthogonalPosF*. The units of the parallel position resource are 'fraction of the viewport width' if the side resource is `Top` or `Bottom` and 'fraction of viewport height' otherwise. Thus the value 1.0 in parallel position units represents either the right or top edge of the viewport, depending on the value of the side resource. When the units of the parallel position resource are ’fraction of the viewport width’ the units of the orthogonal position resource are ’fraction of viewport height’ and vice versa. This implies that, unless the viewport is square, the parallel and orthogonal units are not equal in size. The orthogonal position resource is constrained to positive values in order to prevent location of an annotation within a zone lower than its assigned zone.

Finally, you control the point of the annotation to be placed at the established location. Annotations that follow the model support a resource of type NhlJustification. However the justification point of these annotations is constrained to fall along the side of the annotation parallel with and closest to the zone’s interior boundary side. Therefore, depending on the side, you have a choice of three different possiblities. For the top or bottom sides, you can choose between left, center, and right. For left or right sides, you can choose between bottom, center, and top. Effectively, this constraint enforces the requirement that no part of an annotation’s extent be located in a zone with a lower number than its assigned zone. Note that *constraining* the justification point does not imply that the value of the resource as set by the user is modified; rather it means that the lower-level code acts as if the justification were set to the constrained value.

Annotation plots

Note that while the outer boundary of each annotation zone is based on the bounding boxes of the annotations it contains, annotations are positioned within a zone based on their viewports. For a basic view, such as a **TextItem**, its viewport and bounding box are the same. However, the bounding box of an annotation plot (a subordinate base plot) with its own annotations may extend outside its viewport. If the bounding box is outside the viewport in the negative orthogonal direction (toward the base plot’s center), the subordinate base plot’s annotations will overlap elements of the base plot. To avoid overlap in this case, you must remove all annotations along the inner side of the subordinate base plot.

Special zones

An annotation in one of the standard annotation zones cannot appear within the viewport of the base plot, and there is no way to make it overlay another annotation. Annotations placed in zones 0 and 1 have neither of these restrictions. Moreover, the actual bounding box of the annotation objects in these zones does not influence the boundaries of the standard annotation zones. Basically, zones 0 and 1 offer several convenient coordinate systems for placing objects anywhere within the plot frame.

The origin of zone 0 is the plot or subplot viewport center. The side resource (*amSide* for the **AnnoManager** object) still determines the directions of the parallel and orthogonal axes, just as for the standard annotation zones. Units are still either ’fraction of viewport width’ or ’fraction of viewport height,’ as appropriate. The positive direction for the parallel axis is still in the direction of increasing NDC values, and for the orthogonal axis away from the viewport center. Note that this means that values of 0.5, either for the parallel position resource or for the orthogonal position resource, fall on the viewport boundary. Both the parallel position and the orthogonal position may take on negative values. The justification resource is also unconstrained and therefore can take on any of the nine values available for the NhlJustification type.

Zone 1 has its origin at one of the viewport corners, and its parallel axis is coincident with the viewport side specified by the side resource. Unlike all other annotation zones, however, the direction of the orthogonal axis is *toward* rather than away from the viewport center. The parallel axis is still directed toward increasing NDC values, and like all other zones, the units are either 'fraction of viewport width' or 'fraction of viewport height,' as appropriate. As with zone 0, both the parallel position and the orthogonal position may take on negative values, and the justification resource is unconstrained. Note in particular that when the side resource is set to `Bottom`, the coordinate system has the origin at the lower left and the directionality of a standard Cartesian system (although the units are not necessarily of equal length in the X and Y directions). Also note that whatever the value of side, if you set both the parallel and orthogonal position resources to 1.0, you specify the location of the viewport corner opposite the origin.

Data-tracking annotations

Annotations positioned relative to the data coordinate space of the base plot are called *data-tracking annotations*. You set their location using a different set of positional resources. In the **AnnoManager** object, these are the resources *amDataXF* and *amDataYF*. There is no restriction on data-tracking annotations overlaying another plot feature. Therefore, these annotations are treated as belonging to zone 0 as far as their relationship to other annotations is concerned. Like other zone 0 annotations, data-tracking annotations can take on any value of the justification resource.

Support functions

PlotManager does not define any support functions. However, a number of the **Transform** support functions are designed to be used with objects of classes that use the **PlotManager** as a composite class member.

Status

1. Resources allowing the user to control the order of drawing of overlays within a single draw phase, have not yet been implemented.

2. Resources allowing the user to restrict the bounding box of the complete plot to a portion of the viewspace have not yet been implemented.

See also

- **AnnoManager** object
- **LabelBar** object
- **Legend** object
- **TickMark** object
- **Title** object
- **ContourPlot** object
- **IrregularPlot** object
- **LogLinPlot** object
- **MapPlot** object

- **NhlAddOverlay**function
- **NhlRemoveOverlay**function
- **NhlAddAnnotation**function
- **NhlRemoveAnnotation**function

Copyright

Copyright 1987-1999 University Corporation for Atmospheric Research
The use of this Software is governed by a License Agreement.

NCAR Graphics is a registered trademark of the University Corporation for Atmospheric Research.

ScalarField class

The **ScalarField** class supplies two-dimensional scalar field data.

Synopsis

```
Header file:              ncarg/hlu/ScalarField.h
Class name:               scalarFieldClass
Class pointer:            NhlscalarFieldClass
Fortran class pointer:    NHLFSCALARFIELDCLASS
Superclass:               DataItem
Composite classes:        <None>
```

Resources

Local resources

```
+---------------------------------------------------------------+
|                       ScalarField Resource Set                |
|---------------------------------------------------------------|
| NAME                          TYPE                    ACCESS  |
|       CLASS                           DEFAULT                 |
|===============================================================|
| sfDataArray                   NhlTGenArray            RCSG    |
|       SfDataArray                     NULL                    |
|---------------------------------------------------------------|
| sfXArray                      NhlTGenArray            RCSG    |
|       SfXArray                        NULL                    |
|---------------------------------------------------------------|
| sfYArray                      NhlTGenArray            RCSG    |
|       SfYArray                        NULL                    |
|---------------------------------------------------------------|
```

```
sfCopyData                    NhlTBoolean             RCSG
      diCopyData                      True
---------------------------------------------------------------
sfExchangeDimensions          NhlTBoolean             RCSG
      SfExchangeDimensions            False
---------------------------------------------------------------
sfMissingValueV               NhlTVariable            RCSG
      SfMissingValueV                 NULL
---------------------------------------------------------------
sfDataMinV                    NhlTVariable            RCSG
      SfDataMinV                      <Dynamic>
---------------------------------------------------------------
sfDataMaxV                    NhlTVariable            RCSG
      SfDataMaxV                      <Dynamic>
---------------------------------------------------------------
sfXCStartV                    NhlTVariable            RCSG
      SfXCStartV                      <Dynamic>
---------------------------------------------------------------
sfXCEndV                      NhlTVariable            RCSG
      SfXCEndV                        <Dynamic>
---------------------------------------------------------------
sfYCStartV                    NhlTVariable            RCSG
      SfYCStartV                      <Dynamic>
---------------------------------------------------------------
sfYCEndV                      NhlTVariable            RCSG
      SfYCEndV                        <Dynamic>
---------------------------------------------------------------
sfXCStartSubsetV              NhlTVariable            RCSG
      SfXCStartSubsetV                <Dynamic>
---------------------------------------------------------------
sfXCEndSubsetV                NhlTVariable            RCSG
      SfXCEndSubsetV                  <Dynamic>
---------------------------------------------------------------
sfYCStartSubsetV              NhlTVariable            RCSG
      SfYCStartSubsetV                <Dynamic>
---------------------------------------------------------------
sfYCEndSubsetV                NhlTVariable            RCSG
      SfYCEndSubsetV                  <Dynamic>
---------------------------------------------------------------
sfXCStartIndex                NhlTInteger             RCSG
      SfXCStartIndex                  <Dynamic>
---------------------------------------------------------------
sfXCEndIndex                  NhlTInteger             RCSG
      SfXCEndIndex                    <Dynamic>
---------------------------------------------------------------
sfYCStartIndex                NhlTInteger             RCSG
      SfYCStartIndex                  <Dynamic>
---------------------------------------------------------------
sfYCEndIndex                  NhlTInteger             RCSG
      SfYCEndIndex                    <Dynamic>
---------------------------------------------------------------
sfXCActualStartF              NhlTFloat               G
      SfXCActualStartF                <Dynamic>
---------------------------------------------------------------
sfXCActualEndF                NhlTFloat               G
      SfXCActualEndF                  <Dynamic>
---------------------------------------------------------------
sfYCActualStartF              NhlTFloat               G
      SfYCActualStartF                <Dynamic>
---------------------------------------------------------------
sfYCActualEndF                NhlTFloat               G
```

SfYCActualEndF	<Dynamic>	
sfXCStride SfXCStride	NhlTInteger 1	RCSG
sfYCStride SfYCStride	NhlTInteger 1	RCSG

Composite resources

The **ScalarField** object class has no composite class objects.

Superclass resources

Currently, no superclasses of the **ScalarField** object class define any resources.

Description

The **ScalarField** class encapsulates the functionality required to pass a two-dimensional scalar field data array to a receiving HLU object. One resource must be set to create a **ScalarField** object: the two-dimensional array resource *sfDataArray*, which contains a reference to the storage location of the data array in memory. **ScalarField** handles both regularly and irregularly spaced rectangular data arrays. It assumes the grid is regularly spaced unless you set one or both of the supplementary array resources, *sfXArray* and *sfYArray*, that specify the possibly irregularly spaced location of data points along their respective data coordinate axes.

The **ScalarField** object automatically converts data specified using a variety of numerical types into the NhlTFloat type expected by HLU data receiver objects. The types it currently handles are:

- NhlTByte
- NhlTCharacter
- NhlTShort
- NhlTInteger
- NhlTLong
- NhlTFloat
- NhlTDouble

Many of the supplementary resources may also be set using any of these types. Note that scalar resources capable of accepting any type have the suffix 'V' at the end of their name.

ScalarField can pre-process the array in a number of ways before handing it off to the receiving object. You can exchange the dimensions of the array, effectively turning the data on its side. You can reverse the direction of the data along either or both axes with suitable specification of the endpoints in data coordinate space. You can also make an overly dense array of data more sparse by specifying an index stride value along either or both the X and the Y dimensions.

Instead of sending the complete array to the receiver object, you can pass it any contiguous rectangular subset of the array, specified either by array index or by location in the data coordinate space. Unless the data array dimension sizes also change, **ScalarField** preserves the subset resource settings when the data array changes. This makes it easy to step through a subset of a time- or level-related series of data arrays. If the dimension sizes change, **ScalarField** assumes that the meaning of the data has also changed and that therefore the currently defined subset, besides potentially being out of range, is most likely not meaningful. In this case, the subset resources are reset to include all elements of the data arrays.

Support functions

The **ScalarField** object does not define any support functions, but inherits all the support functions available to its superclass.

Status

1. Note that even if the values of the coordinate array resources (*sfXArray* and *sfYArray*) are actually equally spaced, plot objects such as **ContourPlot** currently use the presence of a coordinate array as a signal that the data are irregularly spaced along the axis in question. In this case, the plot object uses an **IrregularTransformation** object rather than a **LogLinTransformation** object when plotting the data. The practical consequence, besides a small performance penalty, is that it will then be necessary to overlay the plot on a **LogLinPlot** if it is desired to make the X-Axis logarithmic. If the axis is actually regularly spaced, you can avoid this problem by setting *sf...Array* to `NULL`, *sf...CStartV* to the value of the array's first element, and *sf...CEndV* to the value of the array's last element.

See also

- NhlCreate function
- NhlSetValues function
- NhlGetValues function
- NhlAddData function
- ContourPlot object
- DataItem object
- Base object

Copyright

Copyright 1987-1999 University Corporation for Atmospheric Research
The use of this Software is governed by a License Agreement.

NCAR Graphics is a registered trademark of the University Corporation for Atmospheric Research.

StreamlinePlot class

The **StreamlinePlot** class represents a vector field by drawing streamlines.

Synopsis

```
Header file:            ncarg/hlu/StreamlinePlot.h
Class name:             streamlinePlotClass
Class pointer:          NhlstreamlinePlotClass
Fortran class function: NHLFSTREAMLINEPLOTCLASS
Superclass:             DataComm
Composite classes:      LogLinTransformation,IrregularTransformation,PlotManager

Data specific class
Class name:             streamlinePlotDataDepClass
Class pointer:          NhlstreamlinePlotDataDepClass
Fortran class function: NHLFSTREAMLINEPLOTDATADEPCLASS
Superclass:             DataSpec
```

Class-defined types

Resources

Local resources

StreamlinePlot Resource Set

NAME / CLASS	TYPE / DEFAULT	ACCESS
stVectorFieldData StVectorFieldData	NhlTInteger <none>	RCSG
stStreamlineDrawOrder StStreamlineDrawOrder	NhlTDrawOrder "Draw"	RCSG
stMapDirection StMapDirection	NhlTBoolean True	RCSG
stStepSizeF StStepSizeF	NhlTFloat <dynamic>	RCSG
stMinLineSpacingF StMinLineSpacingF	NhlTFloatGenArray <dynamic>	RCSG
stMinStepFactorF StMinStepFactorF	NhlTFloatGenArray 2.0	RCSG
stLengthCheckCount	NhlTInteger	RCSG

Resource	Class	Type	Default	Access
	StLengthCheckCount		35	
stCrossoverCheckCount	StCrossoverCheckCount	NhlTInteger	-1	RCSG
stLineStartStride	StLineStartStride	NhlTInteger	2	RCSG
stArrowStride	StArrowStride	NhlTInteger	2	RCSG
stArrowLengthF	StArrowLengthF	NhlTFloat	<dynamic>	RCSG
stMinArrowSpacingF	StMinArrowSpacingF	NhlTFloatGenArray	0.0	RCSG
stLineThicknessF	LineThicknessF	NhlTFloat	1.0	RCSG
stLineColor	LineColor	NhlTInteger	"Foreground"	RCSG
stNoDataLabelOn	AnnotationLabelsOn	NhlTBoolean	True	RCSG
stNoDataLabelString	StNoDataLabelString	NhlTString	<dynamic>	RCSG
stZeroFLabelOn	AnnotationLabelsOn	NhlTBoolean	True	RCSG
stZeroFLabelString	StZeroFLabelString	NhlTString	<dynamic>	RCSG
stZeroFLabelFontHeightF	FontHeightF	NhlTFloat	<dynamic>	RCSG
stZeroFLabelTextDirection	TextDirection	NhlTTextDirection	"Across"	RCSG
stZeroFLabelFont	Font	NhlTFont	"pwritx"	RCSG
stZeroFLabelFontColor	FontColor	NhlTColorIndex	"Foreground"	RCSG
stZeroFLabelFontAspectF	FontAspectF	NhlTFloat	1.3125	RCSG
stZeroFLabelFontThicknessF	FontThicknessF	NhlTFloat	1.0	RCSG
stZeroFLabelFontQuality	FontQuality	NhlTFontQuality	"High"	RCSG
stZeroFLabelConstantSpacingF	TextConstantSpacingF	NhlTFloat	0.0	RCSG
stZeroFLabelAngleF	TextAngleF	NhlTFloat	0.0	RCSG

```
|-----------------------------------------------------------------|
| stZeroFLabelFuncCode          NhlTCharacter           RCSG      |
|       TextFuncCode                   :                          |
|-----------------------------------------------------------------|
| stZeroFLabelBackgroundColor   NhlTColorIndex          RCSG      |
|       FillBackgroundColor            "Background"               |
|-----------------------------------------------------------------|
| stZeroFLabelPerimOn           NhlTBoolean             RCSG      |
|       EdgesOn                        True                       |
|-----------------------------------------------------------------|
| stZeroFLabelPerimSpaceF       NhlTFloat               RCSG      |
|       EdgeBorderWidthF               0.33                       |
|-----------------------------------------------------------------|
| stZeroFLabelPerimColor        NhlTColorIndex          RCSG      |
|       EdgeColor                      "Foreground"               |
|-----------------------------------------------------------------|
| stZeroFLabelPerimThicknessF   NhlTFloat               RCSG      |
|       EdgeThicknessF                 1.0                        |
|-----------------------------------------------------------------|
| stZeroFLabelZone              NhlTInteger             RCSG      |
|       StZeroFLabelZone               0                          |
|-----------------------------------------------------------------|
| stZeroFLabelSide              NhlTPosition            RCSG      |
|       StZeroFLabelSide               "Bottom"                   |
|-----------------------------------------------------------------|
| stZeroFLabelJust              NhlTJustification       RCSG      |
|       StZeroFLabelJust               "CenterCenter"             |
|-----------------------------------------------------------------|
| stZeroFLabelParallelPosF      NhlTFloat               RCSG      |
|       StZeroFLabelParallelPosF       0.0                        |
|-----------------------------------------------------------------|
| stZeroFLabelOrthogonalPosF    NhlTFloat               RCSG      |
|       StZeroFLabelOrthogonalPosF     0.0                        |
+-----------------------------------------------------------------+
```

Data specific resources

The **StreamlinePlot** class does not currently use any data specific resources.

Composite resources

Transformation resources

Transformation class resources specify the extent and direction of the data coordinate system. As the superclass of both **LogLinTransformation** and **IrregularTransformation**, you can access all **Transformation** resources. However, note that **StreamlinePlot** intercepts these resources, as follows:

- trXMinF
  By default *trXMinF* is set to the minimum data coordinate value along the X Axis, as determined from the contents of the **VectorField** object.
- trXMaxF
  By default *trXMaxF* is set to the maximum data coordinate value along the X Axis, as determined from the contents of the **VectorField** object.
- trXReverse
  By default *trXReverse* is set based on the direction of the X Axis implied by the contents of

the **VectorField** object.

- trYMinF
  By default *trYMinF* is set to the minimum data coordinate value along the Y Axis, as determined from the contents of the **VectorField** object.
- trYMaxF
  By default *trYMaxF* is set to the maximum data coordinate value along the Y Axis, as determined from the contents of the **VectorField** object.
- trYReverse
  By default *trYReverse* is set based on the direction of the Y Axis implied by the contents of the **VectorField** object.

LogLinTransformation resources

The **StreamlinePlot** class uses the **LogLinTransformation** to handle its transformations as long as neither of the **VectorField** array resources, *vfXArray* or *vfYArray*, is set. **LogLinTransformation** has resources for selecting between linear and logarithmic coordinates for each axis.

You can access all **LogLinTransformation** resources. However, note that **StreamlinePlot** intercepts **LogLinTransformation** resources, as follows:

- trXLog
  StreamlinePlot issues a warning if *trXLog* is set True when the set value of *trXMinF* is less than or equal to 0.0. In this case it resets *trXLog* to False. If the **IrregularTransformation** resource *trXAxisType* is set, **StreamlinePlot** sets *trXLog* True if *trXAxisType* is set to `LogAxis`. If *trXAxisType* is set to any other value, it sets *trXLog* False.
- trYLog
  StreamlinePlot issues a warning if *trYLog* is set True when the set value of *trYMinF* is less than or equal to 0.0. In this case it resets *trYLog* to False. If the **IrregularTransformation** resource *trYAxisType* is set, **StreamlinePlot** sets *trYLog* True if *trYAxisType* is set to `LogAxis`. If *trYAxisType* is set to any other value, it sets *trYLog* False.

IrregularTransformation resources

StreamlinePlot automatically uses the **IrregularTransformation** instead of the **LogLinTransformation** to handle its transformations if either of the **VectorField** arrays *vfXArray* or *vfYArray* is set, implying that one or both of the coordinate axes is irregularly spaced.

All **Transformation** superclass resources are accessible. However, **StreamlinePlot** blocks access to all resources specific to the **IrregularTransformation** except for:

- trXTensionF
- trYTensionF

In addition, **StreamlinePlot** intercepts these **IrregularTransformation** resources:

- trXAxisType
  If the **VectorField** resource *vfXArray* is non-`NULL` and *trXAxisType* is set to any other value

than `IrregularAxis`, **StreamlinePlot** switches to a coordinate extent bounded by 0 and the length of the X-Axis dimension minus one. If *trXAxisType* is not set, but the **LogLinTransformation** resource *trXLog* is set, **StreamlinePlot** sets *trXAxisType* to `LogAxis` if *trXLog* is True; if *trXLog* is False, it changes *trXAxisType* to `LinearAxis` if it had been set to `LogAxis` and leaves it unchanged otherwise. *trXAxisType* can be set to `LogAxis` without error only when the X-Axis coordinate extent as passed from the **VectorField** is entirely positive. If this is not the case, *trXAxisType* will default to `LinearAxis`.

V4.1 Status Note 1

- trYAxisType

  If the **VectorField** resource *vfYArray* is non-`NULL` and *trYAxisType* is set to any other value than `IrregularAxis`, **StreamlinePlot** switches to a coordinate extent bounded by 0 and the length of the Y-Axis dimension minus one. If *trYAxisType* is not set, but the **LogLinTransformation** resource *trYLog* is set, **StreamlinePlot** sets *trYAxisType* to `LogAxis` if *trYLog* is True; if *trYLog* is False, it changes *trYAxisType* to `LinearAxis` if it had been set to `LogAxis` and leaves it unchanged otherwise. *trYAxisType* can be set to `LogAxis` without error only when the Y-Axis coordinate extent as passed from the **VectorField** is entirely positive. If this is not the case, *trYAxisType* will default to `LinearAxis`.

  V4.1 Status Note 1

PlotManager resources

If *tfPlotManagerOn* is True when a **StreamlinePlot** object is created, you can access all **PlotManager** resources. However, note that **StreamlinePlot** intercepts certain **PlotManager** resources, as follows:

- pmLabelBarDisplayMode

  The *pmLabelBarDisplayMode* resource is hard-coded to its default value of NhlNOCREATE in **XyPlot**.
- pmLegendDisplayMode

  The *pmLegendDisplayMode* resource is hard-coded to its default value of NhlNOCREATE in **StreamlinePlot**.
- pmTickMarkDisplayMode

  The default value of *pmTickMarkDisplayMode* is set to `Conditional`.
- pmTitleDisplayMode

  The default value of *pmTitleDisplayMode* is set to `Conditional`.

Except for **Legend** and **LabelBar**, you can access resources for any of the composite classes of the **PlotManager** class. However, the **PlotManager** class modifies the access and behavior of some of the resources belonging to these classes, as follows:

- TickMark resources as modified by PlotManager
- Title resources as modified by PlotManager

Superclass resources

You can set all resources defined by the superclasses of the **StreamlinePlot** object class, including:

- **DataComm** resources
- **Transform** resources
- **View** resources

Description

A **StreamlinePlot** object, or **streamlineplot**, represents a vector field using streamlines, based on two-dimensional data provided by an instance of the **VectorField** class. It supports title and tick mark annotations that automatically adjust themselves to the state of the **streamlineplot**.

The **StreamlinePlot** class provides a number of resources that you can use to influence the way the streamlines are generated and displayed.

Data input

The only resource you must set in order to generate a **streamlineplot** is *stVectorFieldData*. This resource specifies the id of an existing **VectorField** object. The **vectorfield** accepts data in a variety of types and configurations, and allows you to specify how it is to be interpreted. As received by the **streamlineplot**, the data are accompanied by information specifying the extents of the data in the data coordinate system, the minimum and maximum data magnitudes, as well as the minimum and maximum values of each component of the vector. Also, if either or both of the data coordinate axes is irregularly spaced, the **vectorfield** communicates information defining this spacing.

Coordinate transformations

StreamlinePlot intrinsically supports linear, logarithmic, and irregular rectangular gridded data coordinate spaces. It does *not yet*, on its own, support the transformation of an irregular data space into either a linear or a logarithmic space. However, such transformations are easily accomplished using the overlay mechanism.

StreamlinePlot instantiates child objects to manage transformations between the data coordinate system and NDC space. The **LogLinTransformation** manages linear and logarithmic transformations, and the **IrregularTransformation** manages the transformation if one or both axes are irregularly spaced. By default the data coordinate extents are based on the extents of the supplied **VectorField** object data, but you may adjust them using resources belonging to the transformation objects.

Use of the LogLinTransformation object

StreamlinePlot uses a **LogLinTransformation** object as long as the **VectorField** resources *vfXArray* and *vfYArray* are both `NULL`. The coordinate extents of a linear axis may arbitrarily intersect or encompass the data extent. If you set a logarithmic axis, then the coordinate extent of that axis must be entirely positive, but otherwise may intersect or extend beyond the data extent.

Use of the IrregularTransformation object

If either of the **VectorField** coordinate array resources *vfXArray* and *vfYArray* are non-`NULL`, then **StreamlinePlot** uses an **IrregularTransformation** object. Note that **StreamlinePlot** treats an axis with an associated coordinate array as irregular even if the coordinate array actually has evenly spaced values. **StreamlinePlot** represents an irregular axis not by stretching and compressing various regions of the plot, but by displaying it with irregularly spaced tick marks.

In addition to a small performance penalty, there are some restrictions associated with use of the **IrregularTransformation** object. Although you may limit the coordinate extent to a subspace of the data coordinate extent of the **VectorField** object data, you are not allowed to define a coordinate range that extends outside the range of the data coordinates of an irregular axis. Using the **IrregularTransformation** resources trXAxisType or trYAxisType, it is possible to set an irregular axis to `LinearAxis` or even, under certain conditions, to `LogAxis`, but the results are probably not what you want. Since **StreamlinePlot** does not intrinsically support a linearization transformation for irregularly spaced data, it can only switch to a linear system by replacing the data coordinates with array index coordinates, which are, in fact, linearly spaced. To properly transform irregularly spaced data into a linear or logarithmic coordinate system, you must use the overlay mechanism (V4.1 Status Note 1).

Overlays

In addition to the built-in transformation support, you can map a **streamlineplot** into a variety of other coordinate spaces by adding it as an overlay to a base plot. You can overlay a **streamlineplot** on a **mapplot** to transform the data into any of 10 different map projections. You can transform irregularly gridded vector data into a linear or logarithmic space by overlaying the **streamlineplot** on a **loglinplot**. You can also make a **streamlineplot** into an overlay of any other plot object, including a **contourplot**, an **irregularplot**, a **vectorplot**, or an **xyplot**. Use the **NhlAddOverlay** function to perform an overlay.

Mapping the vector direction

In certain situations, such as when the coordinate system is irregular or when the units along each coordinate axis are different, you may want to draw streamlines within a coordinate space but without transforming the directional components relative to the space. **StreamlinePlot** has a boolean resource called *stMapDirection* that allows you to specify whether the direction as well as the location is to be mapped into the coordinate space defined by the transformation currently in effect. When you set this resource to False, the direction is rendered in a uniform coordinate system. The U component of the vector data will be parallel to the X Axis, and the V component will be parallel to the Y Axis with units of equal size in each direction.

Draw order

The **StreamlinePlot** class allows you specify when the streamlines are drawn in relation to other plot elements of an overlay. You control the drawing order using the *stStreamlineDrawOrder* resource. There are three drawing phases: the *predraw* phase, the *draw* phase, and the *postdraw* phase. By default, the **StreamlinePlot** object draws its streamlines during the draw phase. When a **streamlineplot** is drawn by itself, the drawing order is not important. However, when a **streamlineplot** is an overlay, you often need to adjust the drawing order to ensure that the features you want to see remain visible.

Annotations

Like all plot objects, a **streamlineplot** is by default instantiated with a **plotmanager** to manage overlays and annotations on its behalf. **StreamlinePlot** enables two **PlotManager** intrinsic annotations, and also supplies a zero field message label of its own. The enabled **PlotManager** annotations include **TickMark** and **Title**. **StreamlinePlot** displays tick marks by default. Titles will appear if you set the appropriate title string resource to a non-NULL string value. As with any plot object, you can also add arbitrary user-defined external annotations to a **streamlineplot**.

Streamline generation resources

StreamlinePlot supplies several resources that influence the way it generates streamlines. The most important of these are *stStepSizeF*, *stMinLineSpacingF*, and *stLineStartStride*.

stStepSizeF specifies the basic step size (in NDC units) that **StreamlinePlot** uses to calculate the next point on the streamline. Although **StreamlinePlot** can usually be counted on to come up with a reasonable value for the step size based on the viewport and the size of the data grid, and although it performs some adaptive step sizing based on the field topology, you will find, in general, that setting a smaller step size will result in a more accurate plot. The drawback is that the streamlines will take a bit longer to generate.

stMinLineSpacingF specifies how close in NDC space streamlines are allowed to approach each other before being terminated. **StreamlinePlot** dynamically calculates a default value for this resource, again based on the viewport size and the number of elements in the data grid. As the value of this resource increases, the streamlines gradually disintegrate into a series of short disconnected lines. Smaller values result in a denser plot with fewer separate lines.

The *stLineStartStride* resource also affects the streamline density, but not in quite the same way as *stMinLineSpacingF*. It specifies at how many points within the data grid streamlines may be started. Streamlines always begin at the center of a 'grid box', that is, at a point in data space located equally distant from four points on the grid where vector data values are defined. If you set *stLineStartStride* to a value greater than 1, some of these potential starting points will not be considered eligible for starting a streamline. For instance, when *stLineStartStride* is set to its default value of 2, only every other grid box (along each data dimension) will be initially eligible. As the drawing operation progresses, many of these initially eligible boxes become ineligible when a streamline in progress passes through the area defined by the outline of the box.

Streamline style resources

You can set the streamline color using *stLineColor* and you can set the thickness of the streamline using *stLineThicknessF*. Setting the streamline thickness also affects the thickness of the lines used to draw the directional arrows.

Directional arrow resources

You can set the length of the lines used to draw each directional arrow using the resource *stArrowLengthF*.

You can control the directional arrow spacing in two different ways. The resource *stArrowStride* works in a similar manner to *stLineStartStride*. It specifies which data grid boxes are eligible for a directional arrow. For example, when *stArrowStride* has its default value, 2, every second grid box (along each dimension) is eligible for a directional arrow.

The *stArrowStride* resource works well unless a non-linear transformation is in effect. In such cases, as for instance in many map transformations, the arrows may be reasonably spaced in the lower latitudes, but become unacceptably crowded near the poles. The resource *stMinArrowSpacingF* was designed to help solve this problem. It allows a directional arrow to be drawn only if at least the NDC distance specified by its value has been added to the streamline since the previous directional arrow.

Zero field annotation

StreamlinePlot provides a zero field annotation that outputs a diagnostic message if you attempt to draw an instance using a **vectorfield** that contains only zero magnitude vectors or missing values. This annotation also outputs a message if you draw without setting the *stVectorFieldData* resource to the id of an existing **vectorfield**. You can individually control the contents of each of these messages as well as whether they should appear at all. The applicable resources are:

- stZeroFLabelOn
- stZeroFLabelString

for controlling the zero field message, and

- stNoDataLabelOn
- stNoDataLabelString

for the no data message.

All the remaining resources for the zero field annotation have the prefix *stZeroFLabel*. These include a complete set of text attribute resources, as well as resources for controlling position according to the locational conventions of the PlotManager Location Control Model.

Status

1. The support for irregular transformations is at a transistional stage. Eventually, **StreamlinePlot** will be able to perform transformations from irregular coordinates to linear and logarithmic coordinates without using the overlay mechanism. This will eliminate the need for a switch to the index coordinate system.

See also

- NhlCreate function
- NhlDestroy function

- NhlSetValues function
- NhlGetValues function
- NhlAddData function
- VectorField object
- PlotManager object
- Title object
- TickMark object
- DataComm object
- Transform object
- View object
- Base object

Copyright

Copyright 1987-1999 University Corporation for Atmospheric Research
The use of this Software is governed by a License Agreement.

NCAR Graphics is a registered trademark of the University Corporation for Atmospheric Research.

Style class

The **Style** class provides its subclasses with the ability to group attributes.

Synopsis

```
Header file:              ncarg/hlu/Style.h
Class name:               styleClass
Class pointer:            <Not referenceable>
Fortran class function:   <Not referenceable>
Superclass:               Base
Composite classes:        <None>
```

Resources

Local resources

The **Style** class does not define any public resources.

Composite resources

The **Style** object class has no composite class resources.

Superclass resources

The superclass of the **Style** object class does not define any resources.

Description

The **Style** class is the non-instantiable superclass of a set of classes used to group the attributes of an entity in the HLU class library, in order to treat them as a unit.

Currently the **Style** class has one publicly accessible subclass: the **GraphicStyle** class.

Support functions

The **Style** class does not define any support functions, but inherits all the support functions available to its superclass.

Status

See also

- Base class

Copyright

Copyright 1987-1999 University Corporation for Atmospheric Research
The use of this Software is governed by a License Agreement.

NCAR Graphics is a registered trademark of the University Corporation for Atmospheric Research.

TextItem class

The **TextItem** object draws arbitrary text.

Synopsis

```
Header file:             ncarg/hlu/TextItem.h
Class name:              textItemClass
Class pointer:           NhltextItemClass
Fortran class function: NHLFTEXTITEMCLASS
Superclass:              View
Composite classes:       <None>
```

Class-defined types

```
Type name:               NhlTFontQuality
Definition:
typedef enum {
        NhlHIGH          = 0,    /* "High"        */
        NhlMEDIUM        = 1,    /* "Medium"      */
        NhlLOW           = 2     /* "Low"         */
} NhlFontQuality;

Type name:               NhlTTextDirection
Definition:
typedef enum {
        NhlDOWN          = 0,    /* "Down"        */
        NhlACROSS        = 1,    /* "Across"      */
} NhlTextDirection;
```

Resources

Local resources

TextItem resources are logically arranged into three groups:

```
 Orientation  - Resources affecting text location and orientation
 Appearance   - Resources that affect tick mark appearance
 Bounding Box - Resources affecting the text bounding box
|================================================================|
|                       TextItem Resource Set                    |
|================================================================|
| Text Location and Orientation Resources                        |
|----------------------------------------------------------------|
| NAME                         TYPE                    ACCESS    |
|       CLASS                          DEFAULT                    |
|================================================================|
| txPosXF                      NhlTFloat               RCSG      |
|       TxPosXF                        <Dynamic>                  |
|----------------------------------------------------------------|
| txPosYF                      NhlTFloat               RCSG      |
|       TxPosYF                        <Dynamic>                  |
|----------------------------------------------------------------|
| txAngleF                     NhlTFloat               RCSG      |
|       TextAngleF                     0.0                        |
|----------------------------------------------------------------|
| txJust                       NhlTJustification       RCSG      |
|       TextJustification            NhlCENTERCENTER              |
|----------------------------------------------------------------|
```

NAME	CLASS	TYPE	DEFAULT	ACCESS
txDirection	TextDirection	NhlTTextDirection	NhlACROSS	RCSG

Text Appearance Resources

NAME	CLASS	TYPE	DEFAULT	ACCESS
txString	TxString	NhlTString	<dynamic>	RCSG
txFont	Font	NhlTFont	pwritx	RCSG
txFontQuality	FontQuality	NhlTFontQuality	"high"	RCSG
txFontColor	FontColor	NhlTColorIndex	NhlFOREGROUND	RCSG
txFontHeightF	FontHeightF	NhlTFloat	.05	RCSG
txFontAspectF	FontAspectF	NhlTFloat	1.3125	RCSG
txFontThicknessF	FontThicknessF	NhlTFloat	1.0	RCSG
txConstantSpacingF	TextConstantSpacingF	NhlTFloat	0.0	RCSG
txFuncCode	TextFuncCode	NhlTCharacter	:	RCSG

Text Bounding Box Resources

NAME	CLASS	TYPE	DEFAULT	ACCESS
txPerimOn	EdgesOn	NhlTBoolean	False	RCSG
txPerimColor	EdgeColor	NhlTColorIndex	NhlFOREGROUND	RCSG
txPerimThicknessF	EdgeThicknessF	NhlTFloat	1.0	RCSG
txPerimDashPattern	EdgeDashPattern	NhlTDashIndex	0	RCSG
txPerimDashLengthF	EdgeDashSegLenF	NhlTFloat	0.15	RCSG
txPerimSpaceF	EdgeBorderWidthF	NhlTFloat	0.5	RCSG
txBackgroundFillColor	FillBackgroundColor	NhlTColorIndex	NhlTRANSPARENT	RCSG

Composite resources

The **TextItem** object class has no composite class objects.

Superclass resources

You may set any resources belonging to the superclasses of the **TextItem** object class, including:

- **View** resources

Description

The **TextItem**, as its name implies, draws text. The text can have any font, any color, any angle, and be any size. You can also draw a rectangular perimeter around the text and optionally provide a solid colored background to the area inside the perimeter box. The **TextItem** is derived from the **View** class and therefore inherits all of the functionality of the **View** class.

The location and size of the text to be drawn can be set using two distinct and separate methods. The two methods can not be mixed because the resources for each method are assigned values based on the placement determined from the other.

The first method is simply using the regular **View** resources *vpXF*, *vpYF*, *vpWidthF*, and *vpHeightF*. However, the **View** resources will be ignored if any of the **TextItem** resources that effect size or position are set at the same time. These resources are: *txPosXF*, *txPosYF*, *txAngleF*, *txJust*, *txDirection*, *txFontAspectF*, *txConstantSpacingF*, and *txFontHeightF*.

When the **View** object resources *vpWidthF*, and *vpHeightF* are set, the value of the *txFontHeightF* resource automatically scales proportionially. Additionally, if the values of *vpWidthF* and *vpHeightF* resource do not preserve the aspect ratio, either the height or width resource are reduced to produce an aspect ratio preserving transformation.

For the second placement method, the **TextItem** uses the resources *txPosXF*, and *txPosYF* in combination with the *txJust* resource. Using this method, it is very easy to determine exactly how the text will be justified relative to other objects on the output frame.

The two position resources *txPosXF* and *txPosYF* specify where the justification point of the text string will be placed. You set the justification point using any of the nine values available to the NhlJustification type resource *txJust*. The default value of *txJust* is NhlCENTERCENTER meaning that the **TextItem** centers the text along both its axes. These axes may be rotated from the NDC coordinate axes if you give *txAngleF* a non-zero value, or they may be exchanged depending on the value given to *txDirection*.

There are 32 HLU Library Fonts, numbered in the range 0 to 37. You specify the desired font using the NhlFont type resource *txFont*. You may set this resource as a string using the name of the font or as an integer using the font index. If you specify the integer index you should note that there are some gaps in the numbering system used for the font indexes.

You can control various attributes of the **TextItem** string's appearance by setting the resources *txFontQuality*, *txFontColor*, *txFontHeightF*, *txFontAspectF*, *txFontThicknessF*, and *txConstantSpacingF*.

The HLU **TextItem** allows you to specify Text Function Codes just as you would for the low-level NCAR Graphics package **Plotchar**. Through the use of function codes, you can control the appearance of the text from within the string *txString*. You can change fonts, vary the size of characters, set the text direction, and use up to five levels of subscripts and superscripts. You can also embed "carriage returns" within the string for multi-line output. You can modify the character used as a function code sentinel using the NhlTCharacter resource *txFuncCode*. Keep in mind, however, that for many normal text operations you do not need to learn how to use function codes.

If you set the boolean resource *txPerimOn* True, the **TextItem** draws a rectangular box around the text string. The box is aligned with the axes of the text string after taking any rotation as a result of *txAngleF* into account. You control the line attributes of the perimeter line using the resources *txPerimColor*, *txPerimThicknessF*, *txPerimDashPattern*, and *txPerimDashLengthF*. You can also set the spacing or margin between the boundary of the characters and the perimeter as a fraction of the current *txFontHeightF* using the resource *txPerimSpaceF*. In addition you can provide a solid- colored background to the space within the rectangular perimeter by setting the resource *txBackgroundFillColor*. The default value of this resource is NhlTRANSPARENT, implying no background fill.

Support functions

The **TextItem** object does not define any support functions, but inherits all the support functions available to its superclass.

Status

See also

- NhlCreate function
- NhlDestroy function
- NhlSetValues function
- NhlGetValues function
- View object
- Base object

Copyright

Copyright 1987-1999 University Corporation for Atmospheric Research
The use of this Software is governed by a License Agreement.

NCAR Graphics is a registered trademark of the University Corporation for Atmospheric Research.

TickMark class

The **TickMark** object draws tick marks along any of the sides of a plot.

Synopsis

```
Header file:             ncarg/hlu/TickMark.h
Class name:              tickMarkClass
Class pointer:           NhltickMarkClass
Fortran class function:  NHLFTICKMARKCLASS
Superclass:              View
Composite classes:       <None>
```

Class-defined types

```
Type name:               NhlTTickMarkMode
Definition:
typedef enum {
        NhlAUTOMATIC    = 0,    /* "Automatic"  */
        NhlMANUAL       = 1,    /* "Manual"     */
        NhlEXPLICIT     = 2     /* "Explicit"   */
} NhlTickMarkMode;

Type name:               NhlTTickMarkStyle
Definition:
typedef enum {
        NhlLOG          = 0,    /* "Log"        */
        NhlLINEAR       = 1,    /* "Linear"     */
        NhlIRREGULAR    = 2,    /* "Irregular"  */
        NhlGEOGRAPHIC   = 3,    /* "Geographic" */
        NhlTIME         = 4     /* "Time"       */
} NhlTickMarkStyle;
```

Resources

Local resources

```
TickMark resources are logically arranged into six groups:

 General      - Resources of a non-specific nature
 Borders      - Resources for drawing the grid boundary
 Grid lines   - Resources to control major and minor grid lines
```

```
Axis scaling - Resources to set the scaling of the four axes
Scale labels - Labels at major tick marks
Appearance   - Resources that affect tick mark appearance
===============================================================
                       TickMark Resource Set
===============================================================
 General Resources

 Resources of a non-specific nature.
---------------------------------------------------------------
 NAME                          TYPE                    ACCESS
       CLASS                           DEFAULT
===============================================================
 tmSciNoteCutoff               NhlTInteger             RCSG
       TmSciNoteCutoff                 6
---------------------------------------------------------------
 tmXUseBottom                  NhlTBoolean             RCSG
       TmXUseBottom                    True
---------------------------------------------------------------
 tmYUseLeft                    NhlTBoolean             RCSG
       TmYUseLeft                      True
===============================================================
 Border Resources

 These resources control the drawing of the grid boundary.
---------------------------------------------------------------
 NAME                          TYPE                    ACCESS
       CLASS                           DEFAULT
===============================================================
 tmXBBorderOn                  NhlTBoolean             RCSG
       EdgesOn                         True
- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
 tmXTBorderOn                  NhlTBoolean             RCSG
       EdgesOn                         True
- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
 tmYLBorderOn                  NhlTBoolean             RCSG
       EdgesOn                         True
- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
 tmYRBorderOn                  NhlTBoolean             RCSG
       EdgesOn                         True
---------------------------------------------------------------
 tmBorderThicknessF            NhlTFloat               RCSG
       EdgeThicknessF                  2.0
---------------------------------------------------------------
 tmBorderLineColor             NhlTColorIndex          RCSG
       EdgeColor                       "Foreground"
===============================================================
 Grid Line Resources

 The following resources relate to the major and minor grid
 lines which run top to bottom or left to right across the
 graph.
---------------------------------------------------------------
 NAME                          TYPE                    ACCESS
       CLASS                           DEFAULT
===============================================================
 Major grid lines
---------------------------------------------------------------
 tmXMajorGrid                  NhlTBoolean             RCSG
       TmXMajorGrid                    False
- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
```

```
 tmYMajorGrid                  NhlTBoolean             RCSG
       TmXMajorGrid                    False
---------------------------------------------------------------
 tmXMajorGridThicknessF        NhlTFloat               RCSG
       TmMajorGridThicknessesF         2.0
- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
 tmYMajorGridThicknessF        NhlTFloat               RCSG
       TmMajorGridThicknessesF         2.0
---------------------------------------------------------------
 tmXMajorGridLineColor         NhlTColorIndex          RCSG
       LineColor                       "Foreground"
- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
 tmYMajorGridLineColor         NhlTColorIndex          RCSG
       LineColor                       "Foreground"
---------------------------------------------------------------
 tmXMajorGridLineDashPattern   NhlTDashIndex           RCSG
       TmMajorGridLineDashPatterns     "SolidLine"
- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
 tmYMajorGridLineDashPattern   NhlTDashIndex           RCSG
       TmMajorGridLineDashPatterns     "SolidLine"
---------------------------------------------------------------
 Minor grid lines
---------------------------------------------------------------
 tmXMinorGrid                  NhlTBoolean             RCSG
       TmXMinorGrid                    False
- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
 tmYMinorGrid                  NhlTBoolean             RCSG
       TmXMinorGrid                    False
---------------------------------------------------------------
 tmXMinorGridThicknessF        NhlTFloat               RCSG
       TmMinorGridThicknessesF         1.0
- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
 tmYMinorGridThicknessF        NhlTFloat               RCSG
       TmMinorGridThicknessesF         1.0
---------------------------------------------------------------
 tmXMinorGridLineColor         NhlTColorIndex          RCSG
       LineColor                       "Foreground"
- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
 tmYMinorGridLineColor         NhlTColorIndex          RCSG
       LineColor                       "Foreground"
---------------------------------------------------------------
 tmXMinorGridLineDashPattern   NhlTDashIndex           RCSG
       TmMinorGridLineDashPatterns     "SolidLine"
- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
 tmYMinorGridLineDashPattern   NhlTDashIndex           RCSG
       TmMinorGridLineDashPatterns     "SolidLine"
===============================================================
 Axis Scaling Resources

 The following resources relate to the four scaleable axes:
 X axes (top and bottom) and Y axes (left and right).

 Note that tmXUseBottom can be used to cause the top axis to
 conform to the bottom axis scaling, and tmYUseLeft can be used
 to cause the right axis to conform to the left axis scaling.
---------------------------------------------------------------
 NAME                          TYPE                    ACCESS
       CLASS                           DEFAULT
===============================================================
 tmXBMode                      NhlTTickMarkMode        RCSG
       TmXBMode                        "Automatic"
```

tmXTMode TmXTMode	NhlTTickMarkMode "Automatic"	RCSG	
tmYLMode TmYLMode	NhlTTickMarkMode "Automatic"	RCSG	
tmYRMode TmYRMode	NhlTTickMarkMode "Automatic"	RCSG	
tmXBMaxTicks TmXBMaxTicks	NhlTInteger 7	RCSG	
tmXTMaxTicks TmXTMaxTicks	NhlTInteger 7	RCSG	
tmYLMaxTicks TmYLMaxTicks	NhlTInteger 7	RCSG	
tmYRMaxTicks TmYRMaxTicks	NhlTInteger 7	RCSG	
tmXBAutoPrecision TmAutoPrecision	NhlTBoolean True	RCSG	
tmXTAutoPrecision TmAutoPrecision	NhlTBoolean True	RCSG	
tmYLAutoPrecision TmAutoPrecision	NhlTBoolean True	RCSG	
tmYRAutoPrecision TmAutoPrecision	NhlTBoolean True	RCSG	
tmXBPrecision TmPrecisions	NhlTInteger 4	RCSG	
tmXTPrecision TmPrecisions	NhlTInteger 4	RCSG	
tmYLPrecision TmPrecisions	NhlTInteger 4	RCSG	
tmYRPrecision TmPrecisions	NhlTInteger 4	RCSG	
tmXBFormat NumberFormat	NhlTString "0@*+^sg"	RCSG	
tmXTFormat NumberFormat	NhlTString "0@*+^sg"	RCSG	
tmYLFormat NumberFormat	NhlTString "0@*+^sg"	RCSG	
tmYRFormat NumberFormat	NhlTString "0@*+^sg"	RCSG	
tmXBStyle TmXBStyle	NhlTTickMarkStyle "Linear"	RCSG	

tmXTStyle TmXTStyle	NhlTTickMarkStyle "Linear"	RCSG
tmYLStyle TmYLStyle	NhlTTickMarkStyle "Linear"	RCSG
tmYRStyle TmYRStyle	NhlTTickMarkStyle "Linear"	RCSG
The following Data resources relate to an axis scale mode of "Automatic", "Manual", or "Explicit".		
tmXBDataLeftF TmXBDataLeftF	NhlTFloat 0.0	RCSG
tmXBDataRightF TmXBDataRightF	NhlTFloat 1.0	RCSG
tmXTDataLeftF TmXTDataLeftF	NhlTFloat 0.0	RCSG
tmXTDataRightF TmXTDataRightF	NhlTFloat 1.0	RCSG
tmYLDataBottomF TmYLDataBottomF	NhlTFloat 0.0	RCSG
tmYLDataTopF TmYLDataTopF	NhlTFloat 1.0	RCSG
tmYRDataBottomF TmYRDataBottomF	NhlTFloat 0.0	RCSG
tmYRDataTopF TmYRDataTopF	NhlTFloat 1.0	RCSG
The following three resources per scaled axis are required for a scale mode of either "Manual" or "Explicit".		
tmXBTickStartF TmXBTickStartF	NhlTFloat 0.0	RCSG
tmXBTickEndF TmXBTickEndF	NhlTFloat 0.0	RCSG
tmXBTickSpacingF TmXBTickSpacingF	NhlTFloat 0.0	RCSG
tmXTTickStartF TmXTTickStartF	NhlTFloat 0.0	RCSG
tmXTTickEndF TmXTTickEndF	NhlTFloat 0.0	RCSG
tmXTTickSpacingF TmXTTickSpacingF	NhlTFloat 0.0	RCSG
tmYLTickStartF TmYLTickStartF	NhlTFloat 0.0	RCSG
tmYLTickEndF	NhlTFloat	RCSG

TmYLTickEndF	0.0	
tmYLTickSpacingF TmYLTickSpacingF	NhlTFloat 0.0	RCSG
tmYRTickStartF TmYRTickStartF	NhlTFloat 0.0	RCSG
tmYRTickEndF TmYRTickEndF	NhlTFloat 0.0	RCSG
tmYRTickSpacingF TmYRTickSpacingF	NhlTFloat 0.0	RCSG
tmXBIrrTensionF TmXBIrrTensionF	NhlTFloat 2.0	RCSG
tmXTIrrTensionF TmXTIrrTensionF	NhlTFloat 2.0	RCSG
tmYLIrrTensionF TmYLIrrTensionF	NhlTFloat 2.0	RCSG
tmYRIrrTensionF TmYRIrrTensionF	NhlTFloat 2.0	RCSG
tmXBIrregularPoints TmXBIrregularPoints	NhlTFloatGenArray NULL	RCSG
tmXTIrregularPoints TmXTIrregularPoints	NhlTFloatGenArray NULL	RCSG
tmYLIrregularPoints TmYLIrregularPoints	NhlTFloatGenArray NULL	RCSG
tmYRIrregularPoints TmYRIrregularPoints	NhlTFloatGenArray NULL	RCSG
The following array resource pairs are used if and only if the Tm**Mode resource is EXPLICIT for the chosen axis.		
tmXBValues TmXBValues	NhlTFloatGenArray NULL	RCSG
tmXBLabels TmXBLabels	NhlTStringGenArray NULL	RCSG
tmXBMinorValues TmXBMinorValues	NhlTFloatGenArray NULL	RCSG
tmXTValues TmXTValues	NhlTFloatGenArray NULL	RCSG
tmXTLabels TmXTLabels	NhlTStringGenArray NULL	RCSG
tmXTMinorValues TmXTMinorValues	NhlTFloatGenArray NULL	RCSG
tmYLValues TmYLValues	NhlTFloatGenArray NULL	RCSG

NAME / CLASS	TYPE / DEFAULT	ACCESS
tmYLLabels TmYLLabels	NhlTStringGenArray NULL	RCSG
tmYLMinorValues TmYLMinorValues	NhlTFloatGenArray NULL	RCSG
tmYRValues TmYRValues	NhlTFloatGenArray NULL	RCSG
tmYRLabels TmYRLabels	NhlTStringGenArray NULL	RCSG
tmYRMinorValues TmYRMinorValues	NhlTFloatGenArray NULL	RCSG

Label Resources

The following resources relate to writing scale labels at major tick mark locations.

NAME CLASS	TYPE DEFAULT	ACCESS
tmXBLabelsOn TmXBLabelsOn	NhlTBoolean True	RCSG
tmXTLabelsOn TmXTLabelsOn	NhlTBoolean False	RCSG
tmYLLabelsOn TmYLLabelsOn	NhlTBoolean True	RCSG
tmYRLabelsOn TmYRLabelsOn	NhlTBoolean False	RCSG
tmXBLabelStride TmXBLabelStride	NhlTInteger 0	RCSG
tmXTLabelStride TmXTLabelStride	NhlTInteger 0	RCSG
tmYLLabelStride TmYLLabelStride	NhlTInteger 0	RCSG
tmYRLabelStride TmYRLabelStride	NhlTInteger 0	RCSG
tmXBLabelFuncCode TextFuncCode	NhlTCharacter :	RCSG
tmXTLabelFuncCode TextFuncCode	NhlTCharacter :	RCSG
tmYLLabelFuncCode TextFuncCode	NhlTCharacter :	RCSG
tmYRLabelFuncCode TextFuncCode	NhlTCharacter :	RCSG

tmXBLabelFont	NhlTFont	RCSG
Font	"pwritx"	
tmXTLabelFont	NhlTFont	RCSG
Font	"pwritx"	
tmYLLabelFont	NhlTFont	RCSG
Font	"pwritx"	
tmYRLabelFont	NhlTFont	RCSG
Font	"pwritx"	
tmXBLabelFontHeightF	NhlTFloat	RCSG
FontHeightF	0.02	
tmXTLabelFontHeightF	NhlTFloat	RCSG
FontHeightF	0.02	
tmYLLabelFontHeightF	NhlTFloat	RCSG
FontHeightF	0.02	
tmYRLabelFontHeightF	NhlTFloat	RCSG
FontHeightF	0.02	
tmXBLabelFontColor	NhlTColorIndex	RCSG
FontColor	"Foreground"	
tmXTLabelFontColor	NhlTColorIndex	RCSG
FontColor	"Foreground"	
tmYLLabelFontColor	NhlTColorIndex	RCSG
FontColor	"Foreground"	
tmYRLabelFontColor	NhlTColorIndex	RCSG
FontColor	"Foreground"	
tmXBLabelFontAspectF	NhlTFloat	RCSG
FontAspectF	1.3125	
tmXTLabelFontAspectF	NhlTFloat	RCSG
FontAspectF	1.3125	
tmYLLabelFontAspectF	NhlTFloat	RCSG
FontAspectF	1.3125	
tmYRLabelFontAspectF	NhlTFloat	RCSG
FontAspectF	1.3125	
tmXBLabelFontThicknessF	NhlTFloat	RCSG
FontThicknessF	1.0	
tmXTLabelFontThicknessF	NhlTFloat	RCSG
FontThicknessF	1.0	
tmYLLabelFontThicknessF	NhlTFloat	RCSG
FontThicknesF	1.0	
tmYRLabelFontThicknessF	NhlTFloat	RCSG
FontThicknessF	1.0	
tmXBLabelFontQuality	NhlTFontQuality	RCSG

FontQuality	"High"	
tmXTLabelFontQuality	NhlTFontQuality	RCSG
FontQuality	"High"	
tmYLLabelFontQuality	NhlTFontQuality	RCSG
FontQuality	"High"	
tmYRLabelFontQuality	NhlTFontQuality	RCSG
FontQuality	"High"	
tmXBLabelConstantSpacingF	NhlTFloat	RCSG
TextConstantSpacingF	0.0	
tmXTLabelConstantSpacingF	NhlTFloat	RCSG
TextConstantSpacingF	0.0	
tmYLLabelConstantSpacingF	NhlTFloat	RCSG
TextConstantSpacingF	0.0	
tmYRLabelConstantSpacingF	NhlTFloat	RCSG
TextConstantSpacingF	0.0	
tmXBLabelJust	NhlTJustification	RCSG
TmXBLabelJust	"TopCenter"	
tmXTLabelJust	NhlTJustification	RCSG
TmXTLabelJust	"BottomCenter"	
tmYLLabelJust	NhlTJustification	RCSG
TmYLLabelJust	"CenterRight"	
tmYRLabelJust	NhlTJustification	RCSG
TmYRLabelJust	"TopLeft"	
tmXBLabelAngleF	NhlTFloat	RCSG
TextAngleF	0.0	
tmXTLabelAngleF	NhlTFloat	RCSG
TextAngleF	0.0	
tmYLLabelAngleF	NhlTFloat	RCSG
TextAngleF	0.0	
tmYRLabelAngleF	NhlTFloat	RCSG
TextAngleF	0.0	
tmXBLabelDirection	NhlTTextDirection	RCSG
TextDirection	"Across"	
tmXTLabelDirection	NhlTTextDirection	RCSG
TextDirection	"Across"	
tmYLLabelDirection	NhlTTextDirection	RCSG
TextDirection	"Across"	
tmYRLabelDirection	NhlTTextDirection	RCSG
TextDirection	"Across"	
tmXBLabelDeltaF	NhlTFloat	RCSG
TmXBLabelDeltaF	0.0	

tmXTLabelDeltaF	NhlTFloat	RCSG	TmXTLabelDeltaF	0.0
tmYLLabelDeltaF	NhlTFloat	RCSG	TmYLLabelDeltaF	0.0
tmYRLabelDeltaF	NhlTFloat	RCSG	TmYRLabelDeltaF	0.0

TickMark Appearance Resources

NAME	TYPE	ACCESS	CLASS	DEFAULT
tmXBOn	NhlTBoolean	RCSG	TmXBOn	True
tmXTOn	NhlTBoolean	RCSG	TmXTOn	True
tmYLOn	NhlTBoolean	RCSG	TmYLOn	True
tmYROn	NhlTBoolean	RCSG	TmYROn	True
tmXBMinorPerMajor	NhlTInteger	RCSG	TmXMinorPerMajor	<dynamic>
tmXTMinorPerMajor	NhlTInteger	RCSG	TmXMinorPerMajor	<dynamic>
tmYLMinorPerMajor	NhlTInteger	RCSG	TmYMinorPerMajor	<dynamic>
tmYRMinorPerMajor	NhlTInteger	RCSG	TmYMinorPerMajor	<dynamic>
tmXBMajorThicknessF	NhlTFloat	RCSG	TmMajorThicknessesF	2.0
tmXTMajorThicknessF	NhlTFloat	RCSG	TmMajorThicknessesF	2.0
tmYLMajorThicknessF	NhlTFloat	RCSG	TmMajorThicknessesF	2.0
tmYRMajorThicknessF	NhlTFloat	RCSG	TmMajorThicknessesF	2.0
tmXBMajorLineColor	NhlTColorIndex	RCSG	LineColor	"Foreground"
tmXTMajorLineColor	NhlTColorIndex	RCSG	LineColor	"Foreground"
tmYLMajorLineColor	NhlTColorIndex	RCSG	LineColor	"Foreground"
tmYRMajorLineColor	NhlTColorIndex	RCSG		

LineColor		"Foreground"	
tmXBMajorLengthF	NhlTFloat		RCSG
TmMajorLengthsF		<dynamic>	
tmXTMajorLengthF	NhlTFloat		RCSG
TmMajorLengthsF		<dynamic>	
tmYLMajorLengthF	NhlTFloat		RCSG
TmMajorLengthsF		<dynamic>	
tmYRMajorLengthF	NhlTFloat		RCSG
TmMajorLengthsF		<dynamic>	
tmXBMajorOutwardLengthF	NhlTFloat		RCSG
TmMajorOutwardLengthsF		0.0	
tmXTMajorOutwardLengthF	NhlTFloat		RCSG
TmMajorOutwardLengthsF		0.0	
tmYLMajorOutwardLengthF	NhlTFloat		RCSG
TmMajorOutwardLengthsF		0.0	
tmYRMajorOutwardLengthF	NhlTFloat		RCSG
TmMajorOutwardLengthsF		0.0	
tmXBMinorOn	NhlTBoolean		RCSG
TmXBMinorOn		True	
tmXTMinorOn	NhlTBoolean		RCSG
TmXTMinorOn		True	
tmYLMinorOn	NhlTBoolean		RCSG
TmYLMinorOn		True	
tmYRMinorOn	NhlTBoolean		RCSG
TmYRMinorOn		True	
tmXBMinorThicknessF	NhlTFloat		RCSG
TmMinorThicknessesF		1.0	
tmXTMinorThicknessF	NhlTFloat		RCSG
TmMinorThicknessesF		1.0	
tmYLMinorThicknessF	NhlTFloat		RCSG
TmMinorThicknessesF		1.0	
tmYRMinorThicknessF	NhlTFloat		RCSG
TmMinorThicknessesF		1.0	
tmXBMinorLineColor	NhlTColorIndex		RCSG
LineColor		"Foreground"	
tmXTMinorLineColor	NhlTColorIndex		RCSG
LineColor		"Foreground"	
tmYLMinorLineColor	NhlTColorIndex		RCSG
LineColor		"Foreground"	
tmYRMinorLineColor	NhlTColorIndex		RCSG
LineColor		"Foreground"	

```
-------------------------------------------------------------
 tmXBMinorLengthF             NhlTFloat               RCSG
       TmMinorLengthsF                <dynamic>
- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
 tmXTMinorLengthF             NhlTFloat               RCSG
       TmMinorLengthsF                <dynamic>
- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
 tmYLMinorLengthF             NhlTFloat               RCSG
       TmMinorLengthsF                <dynamic>
- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
 tmYRMinorLengthF             NhlTFloat               RCSG
       TmMinorLengthsF                <dynamic>
-------------------------------------------------------------
 tmXBMinorOutwardLengthF      NhlTFloat               RCSG
       TmMinorOutwardLengthsF        0.0
- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
 tmXTMinorOutwardLengthF      NhlTFloat               RCSG
       TmMinorOutwardLengthsF        0.0
- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
 tmYLMinorOutwardLengthF      NhlTFloat               RCSG
       TmMinorOutwardLengthsF        0.0
- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
 tmYRMinorOutwardLengthF      NhlTFloat               RCSG
       TmMinorOutwardLengthsF        0.0
-------------------------------------------------------------
```

Superclass resources

Normally, the **TickMark** class is being used as a composite class member of a plot class, so its superclass resources are usually overridden by the resources of the composite class. These include resources of the following superclass:

- **View** resources

Description

The complexity of the **TickMark** object stems from trying to provide as much configurability as possible. The more customized you want to make your tick marks, the more resources you'll have to learn. The main function of the **TickMark** object is to draw a border, major tick marks, minor tick marks, and tick mark labels. Each axis is broken into two parts, a top and a bottom for the X Axis and a left and right part for the Y Axis. These four distinct parts have the same resources but with different prefixes. The X-Axis prefixes are *tmXB* and *tmXT* for the bottom and top parts of the X Axis respectively. The Y-Axis prefixes are *tmYL* and *tmYR* for the left and right parts of the Y Axis respectively. The two parts can be combined by setting the resource *tmXUseBottom* and *tmYUseLeft* resources True. What these do is copy most of the resources for the bottom tick marks into the resource for the top tick for the X Axis and the same type of operation for the Y Axis. See the description of *tmXUseBottom* and *tmYUseLeft* for a list of resources affected by setting these resources. By default *tmXUseBottom* and *tmYUseLeft* are set True.

Use as a composite class

The **TickMark** class is most commonly instantiated as a composite class of the **PlotManager**. It is one of the four types of intrinsic annotation available to plot object classes such as the **XyPlot** or **ContourPlot**. Used in this way, you no longer control some of the **TickMark's** resources, since the controlling **PlotManager** sets them automatically based on its knowledge of the data extents and the transformation in effect. However, the great majority of **TickMark** resources remain under your control, allowing you complete freedom to configure such things as the spacing of the tick marks, the content and appearance of the labels, and whether or not to create a grid over the data.

Modes

Each side of each axis can operate in one of three modes: `Automatic`, `Manual`, or `Explicit`. `Automatic` mode, the default, depends primarily only on the resources that specify the data values at each end of the axis. These are *tmXBDataLeftF* and *tmXBDataRightF* for the bottom X Axis and *tmYLDataTopF* and *tmYLDataBottomF* for the left Y Axis. If not otherwise specified, the left and bottom data value resources default to a value of 0.0 while the right and top resources default to a value of 1.0. Based on the data range, **TickMark** computes "nice" tick mark values with a maximum limit on the number of major tick marks it generates. The resources for setting the maximum tick mark limit for each axis are *tmXBMaxTicks*, *tmXTMaxTicks*, *tmYRMaxTicks*, and *tmYLMaxTicks*.

When the **TickMark** is instantiated as a composite class of the **PlotManager**, the base plot takes care of setting the data left, right, bottom, and top resources, thus ensuring consistency with the base plot's data coordinate space.

`Manual` mode uses additional resources that specify a starting tick coordinate value, an ending tick coordinate value, and a value that sets the spacing between the major tick marks. The resources that control the start, end, and spacing for the bottom X Axis are *tmXBTickStartF*, *tmXBTickEndF*, and *tmXBTickSpacingF*. If you do not set the tick start or end values, the default is to use the data left and right or top and bottom values. If you do not set the spacing value, **TickMark** issues a warning message and reverts to `Automatic` mode.

`Explicit` mode expects you to provide an array containing the values of the data coordinate points for the desired major tick marks and another array containing a list of strings to use as the tick mark labels. You therefore have complete control over where a tick mark is placed and what the label says. Using `Explicit` mode, you can create a different scale for each side of the axis. For the bottom X Axis, the resource *tmXBValues* contains the array of values, while *tmXBLabels* contains the label strings. You may optionally also set the array resource *tmYRMinorValues* to specify the locations of minor ticks along the axis.

Styles

Tick marks currently have three different styles and will have five styles eventually. Styles should be thought of as selecting the type of transformation for the tick mark axis to use. The three implemented styles are `Linear`, `Log`, and `Irregular`. The styles `Geographic` and `Time` will be added in the near future. Setting an axis' style to `Log` means that the axis will display `Log` tick marks. The only difference between `Linear`, `Log`, and `Irregular` style tick marks that the user should be aware of is when in `Manual` mode, the *tm...TickSpacingF* resources should be set to the number of decades to space major tick marks instead of an actual data value. Otherwise all resources function identically regardless of which style and mode combination is selected.

`Irregular` style tick marks are a new feature for tick marks. It allows the user to specify how data should be mapped to the output frame. The basic concept is the user provides a set of data points that the user wants to be evenly spaced along the axis. The **TickMark** object then sets up a mapping and chooses tick marks in the standard way.

When the **TickMark** is instantiated as a composite class of the **PlotManager**, the base plot takes care of setting the **TickMark** style based on its knowledge of the data transformation currently in effect.

Label text attributes

TickMark provides for the tick labels all of the text attributes supported by the **TextItem** class, and the names of the resources are consistent with the conventions established for the HLU library as a whole. For example, you would set the label font for the bottom tick mark labels using the resource *tmXBLabelFont*. This name can be derived from the equivalent **TextItem** resource, *txFont*, by replacing the **TextItem** prefix *tx* with the prefix *tmXBLabel*.

Numerical label formatting

There are several levels of control available for the format of the numerical labels that **TickMark** generates when `Automatic` or `Manual` mode is in effect. For each axis there is an integer and a boolean resource for controlling the precision (number of significant digits); there is also a single integer resource that sets, for generically formatted labels on any of the axes, the threshold string length used to decide when to switch to an exponential representation of the value. In addition, for each axis there is a format string resource that allows very detailed control of the number's appearance using the HLU Floating Point Format Specification. It is possible to override the boolean and integer resources by explicitly setting the format string resource in certain ways.

For the bottom X Axis, the two resources that control the precision are the boolean resource *tmXBAutoPrecision* and the integer resource *tmXBPrecision*. If *tmXBAutoPrecision* is True, **TickMark** determines an appropriate precision to use based on the range between the minimum label value and the maximum label value. If *tmXBMode* is set to `Manual`, **TickMark** also considers the value of *tmXBTickSpacingF* when determining the precision. Otherwise, if *tmXBAutoPrecision* is False, the precision is set to the value of *tmXBPrecision*. Given the default value of the string format resource *tmXBFormat*, the precision specifies the number of significant digits that will be used to represent the label with the greatest absolute value in the set of bottom X-Axis labels. The remaining labels will have the same number of digits to the left of the decimal point, but possibly fewer to the right, since their assumed leftmost significant digit is set to coincide with the actual leftmost significant digit of the maximum absolute value among the labels. For example, if the labels range from 9 to 11 in steps of 0.5 the label strings will be output as follows: "9.0 9.5 10.0 10.5 11.0".

Many other options are possible if you change the string formatting resource *tmXBFormat* from its default value of "0@*+^sg". For instance, if you remove the "*+" characters that cause all otherwise unspecified conversion fields to be set dynamically, the leftmost significant digit conversion field will assume its 'unspecified' state. **TickMark** will respond by formatting each label using the actual number of significant digits that the precision specifies. For the previous example the resulting output would be: "9.00 9.50 10.0 10.5 11.0". If *tmXBStyle* is set to `Log`, **TickMark** forces several of the formatting options to ensure that only an exponential format without a mantissa is available. Consult the HLU Floating Point Format Specification to learn more of the details of manipulating the output format of numerical labels. Of course, each of the other **TickMark** axes has resources equivalent to the bottom X-Axis resources discussed here.

The resource *tmSciNoteCutoff* applies to all **TickMark** labels. Assuming the applicable format string is using the 'g' or 'G' generic conversion specifier, the value of *tmSciNoteCutoff* determines the exponent switch length. If the number of digits required to represent the value using a non-exponential representation is less than the exponent switch length, then the non-exponential representation is used. Otherwise, the representation (exponential or non-exponential) that uses the fewest number of characters is used. You can override the value of *tmSciNoteCutoff* for any individual side by setting the exponent switch length conversion field explicitly in the format string.

Support functions

The **TickMark** object does not define any support functions, but inherits all the support functions available to its superclass.

Status

1. If the coordinate values in the *tm...Values* array(s) are *too* irregular, the irregular transformation will fail. What constitutes *too* irregular remains to be defined.

2. Values of the *tm...TensionF* parameter greater than about 10.0 cause instability in the irregular transformation.

See also

- NhlCreate function
- NhlDestroy function
- NhlSetValues function
- NhlGetValues function
- View object
- Base object

Copyright

Copyright 1987-1999 University Corporation for Atmospheric Research
The use of this Software is governed by a License Agreement.

NCAR Graphics is a registered trademark of the University Corporation for Atmospheric Research.

Title class

The **Title** object draws main, X, and Y axis titles for a parent plot or overlay object.

Synopsis

```
Header file:             ncarg/hlu/Title.h
Class name:              titleClass
Class pointer:           NhltitleClass
Fortran class function:  NHLFTITLECLASS
Superclass:              View
Composite classes:       <None>
```

Resources

Local resources

Title Resource Set

NAME	CLASS	TYPE	DEFAULT	ACCESS
tiDeltaF	TiDeltaF	NhlTFloat	1.5	RCSG
tiUseMainAttributes	TiUseMainAttributes	NhlTBoolean	False	RCSG
tiMainOn	TiMainOn	NhlTBoolean	<Dynamic>	RCSG
tiMainString	TiMainString	NhlTString	"Main"	RCSG
tiMainSide	TiMainSide	NhlTPosition	"Top"	RCSG
tiMainPosition	TiMainPosition	NhlTPosition	"Center"	RCSG
tiMainOffsetXF	TiMainOffsetXF	NhlTFloat	0.0	RCSG
tiMainOffsetYF	TiMainOffsetYF	NhlTFloat	0.0	RCSG
tiMainJust	TextJustification	NhlTJustification	"CenterCenter"	RCSG

tiMainFont	Font	NhlTFont	0	RCSG
tiMainFontColor	FontColor	NhlTColorIndex	"Foreground"	RCSG
tiMainFontHeightF	FontHeightF	NhlTFloat	0.025	RCSG
tiMainFontAspectF	FontAspectF	NhlTFloat	1.3125	RCSG
tiMainConstantSpacingF	TextConstantSpacingF	NhlTFloat	0.0	RCSG
tiMainFontThicknessF	FontThicknessF	NhlTFloat	1.0	RCSG
tiMainFuncCode	TextFuncCode	NhlTCharacter	:	RCSG
tiMainFontQuality	FontQuality	NhlTFontQuality	"High"	RCSG
tiMainAngleF	TextAngleF	NhlTFloat	0.00	RCSG
tiMainDirection	TextDirection	NhlTTextDirection	"Across"	RCSG
tiXAxisOn	TiXAxisOn	NhlTBoolean	<Dynamic>	RCSG
tiXAxisString	TiXAxisString	NhlTString	"XAxis"	RCSG
tiXAxisSide	TiXAxisSide	NhlTPosition	"Bottom"	RCSG
tiXAxisPosition	TiXAxisPosition	NhlTPosition	"Center"	RCSG
tiXAxisOffsetXF	TiXAxisOffsetXF	NhlTFloat	0.0	RCSG
tiXAxisOffsetYF	TiXAxisOffsetYF	NhlTFloat	0.0	RCSG
tiXAxisJust	TextJustification	NhlTColorIndex	"CenterCenter"	RCSG
tiXAxisFont	Font	NhlTFont	0	RCSG
tiXAxisFontColor	FontColor	NhlTColorIndex	"Foreground"	RCSG
tiXAxisFontHeightF	FontHeightF	NhlTFloat	0.025	RCSG
tiXAxisFontAspectF		NhlTFloat		RCSG

	FontAspectF		1.3125	
tiXAxisConstantSpacingF	TextConstantSpacingF	NhlTFloat	0.0	RCSG
tiXAxisFontThicknessF	FontThicknessF	NhlTFloat	1.0	RCSG
tiXAxisFuncCode	TextFuncCode	NhlTCharacter	:	RCSG
tiXAxisFontQuality	FontQuality	NhlTFontQuality	"High"	RCSG
tiXAxisAngleF	TextAngleF	NhlTFloat	0.0	RCSG
tiXAxisDirection	TextDirection	NhlTTextDirection	"Across"	RCSG
tiYAxisOn	TiYAxisOn	NhlTBoolean	<Dynamic>	RCSG
tiYAxisString	TiYAxisString	NhlTString	"YAxis"	RCSG
tiYAxisSide	TiYAxisSide	NhlTPosition	"Left"	RCSG
tiYAxisPosition	TiYAxisPosition	NhlTPosition	"Center"	RCSG
tiYAxisOffsetXF	TiYAxisOffsetXF	NhlTFloat	0.0	RCSG
tiYAxisOffsetYF	TiYAxisOffsetYF	NhlTFloat	0.0	RCSG
tiYAxisJust	TextJustification	NhlTJustification	"CenterCenter"	RCSG
tiYAxisFont	Font	NhlTFont	0	RCSG
tiYAxisFontColor	FontColor	NhlTColorIndex	"Foreground"	RCSG
tiYAxisFontHeightF	FontHeightF	NhlTFloat	0.025	RCSG
tiYAxisFontAspectF	FontAspectF	NhlTFloat	1.3125	RCSG
tiYAxisConstantSpacingF	TextConstantSpacingF	NhlTFloat	0.0	RCSG
tiYAxisFontThicknessF	FontThicknessF	NhlTFloat	1.0	RCSG
tiYAxisFuncCode	TextFuncCode	NhlTCharacter	:	RCSG

```
|---------------------------------------------------------------|
| tiYAxisFontQuality           NhlTFontQuality        RCSG     |
|       FontQuality                   "High"                    |
|---------------------------------------------------------------|
| tiYAxisAngleF                NhlTFloat              RCSG     |
|       YAxisTextAngleF               90.0                      |
|---------------------------------------------------------------|
| tiYAxisDirection             NhlTTextDirection      RCSG     |
|       YAxisTextDirection            "Across"                  |
+---------------------------------------------------------------+
```

Composite resources

The **Title** object class intercepts all the resources of its child **TextItem** objects. Therefore, there are no composite resources that can be set for the **Title** object class.

Superclass resources

You may set any resources belonging to the superclasses of the **Title** object class, including:

- **View** resources

Description

Most often used as a plot annotation, the **Title** class draws titles around its viewport. The **Title** viewport is usually somewhat larger than the viewport belonging to the plot, because it may include the area occupied by other plot annotations, mostly commonly **TickMark** labels. Three titles can be drawn by the **Title** object: the *Main* title, the *X-Axis* title and the *Y-Axis* title. The Main title and the X-Axis titles appear either above or below the viewport, while the Y-Axis title appears to the left or right of the viewport. You can turn the individual titles on or off using the resources *tiMainOn*, *tiXAxisOn*, and *tiYAxisOn*. By default the Main title is centered above the viewport, the X-Axis title is centered below the viewport, and the Y-Axis title is centered to the left of the viewport. These default positions can be adjusted by various resources.

Names for the resources used to control the text attributes of the titles are derived in a consistent way from the corresponding **TextItem** resources: the individual title's name is prepended to the **TextItem** resource name. For example, the **TextItem** resource for setting the content string is *txString*. For its three titles, the **Title** object provides the resources *tiMainString*, *tiXAxisString*, and *tiYAxisString*. All other text attribute resources for the titles are constructed in the same fashion. However, note that **Title** uses its own scheme for positioning the titles relative to the viewport. Hence there are no resources that correspond to the **TextItem** *txPosXF* and *txPosYF* resources. Also, the **Title** object does not currently support perimeter outlines or background fill of the area occupied by the text string.

Positioning the Titles

The *ti...Side* resources specify on which side of the viewport the title is to appear. *tiMainSide* and *tiXAxisSide* may be set to `Top` or `Bottom`. *tiYAxisSide* may be set to `Left` or `Right`. The *ti...Position* resources specify the initial position of the title justification point along the chosen side. *tiMainPosition* and *tiXAxisPosition* allow the values `Left`, `Center`, and `Right`, while *tiYAxisPosition* accepts the values `Bottom`, `Center`, and `Top`.

The resource *tiDeltaF* applies to all the titles, and specifies, in units of the title's current font height, how far out from the viewport edge the justification point of the titles should be displaced. Note that certain settings of the justification mode, angle, and direction may result in a portion of the title overlapping the **Title** viewport area. In this case, the title's position is adjusted farther outward such that no part of the title text remains inside the viewport. When the X-Axis title and the Main title share the same side of the viewport, the Main title's position is established outside the X-Axis title. The outer bounding line of the X-Axis title is in effect treated as a viewport boundary for the Main title.

From the positions thus established, you can displace the individual titles in any direction using the **Title** offset resources, *tiMainOffsetXF*, *tiMainOffsetYF*, *tiXAxisOffsetXF*, *tiXAxisOffsetYF*, *tiYAxisOffsetXF*, or *tiYAxisOffsetYF*. Since these resources are applied after the previously mentioned adjustment, they can be used to move the titles arbitrarily. It is possible, for instance, to move a title to a location within the viewport. All the offset resources use NDC units, their positive directions coinciding with the directions of the NDC system.

Support functions

The **Title** object does not define any support functions, but inherits all the support functions available to its superclass.

Status

See also

- NhlCreate function
- NhlDestroy function
- NhlSetValues function
- NhlGetValues function
- View object
- Base object

Copyright

Copyright 1987-1999 University Corporation for Atmospheric Research
The use of this Software is governed by a License Agreement.

NCAR Graphics is a registered trademark of the University Corporation for Atmospheric Research.

Transformation class

The **Transformation** classes manage forward and reverse transformations for the **Transform** classes.

Synopsis

Header file: ncarg/hlu/Transformation.h
Class name: transformationClass
Class pointer: <*Not referenceable*>
Fortran class function: <*Not referenceable*>
Superclass: Obj
Composite classes: <*None*>

Resources

Local resources

Transformation Resource Set

NAME	CLASS	TYPE	DEFAULT	ACCESS
trXMinF	TrXMinF	NhlTFloat	0.0	RCSG
trXMaxF	TrXMaxF	NhlTFloat	1.0	RCSG
trXReverse	TrXReverse	NhlTBoolean	False	RCSG
trYMinF	TrYMinF	NhlTFloat	0.0	RCSG
trYMaxF	TrYMaxF	NhlTFloat	1.0	RCSG
trYReverse	TrYReverse	NhlTBoolean	False	RCSG
trLineInterpolationOn	TrLineInterpolationOn	NhlTBoolean	False	RCSG

Composite resources

The **Transformation** object class has no composite class resources.

Superclass resources

The superclass of the **Transformation** object class does not define any resources.

Description

Transformation class objects perform forward and reverse transformations between data and NDC space for objects belonging to **Transform** class. You do not instantiate any of these objects directly. Rather, **Transform** class objects create them as needed, and provide controlled access to their resources.

The **Transformation** class is never itself instantiated. However, it does provide some basic resources that its subclasses may use to specify the extent and direction of the data space. Currently there are three subclasses of the **Transformation** class:

- The **LogLinTransformation** provides transformations to and from a linear or logarithmic coordinate space. Either or both coordinate axes may be logarithmic.
- The **IrregularTransformation** provides transformations to and from rectangular coordinate spaces where each coordinate axis may be either irregularly spaced, linear, or logarithmic.
- The **MapTransformation** provides transformations to and from any of 10 different standard map projections. It disables access to all the basic **Transformation** class resources, since it requires more specialized resources for defining extent and direction adequately.

Support functions

There are no special support functions defined for the **Transformation** class or its superclass.

Status notes

1. When line interpolation is turned on, the quality of the interpolation into data space is not always very good, particularly for log plots.

See also

- LogLinTransformation object
- IrregularTransformation object
- MapTransformation object
- Transform object
- Obj object

Copyright

Copyright 1987-1999 University Corporation for Atmospheric Research
The use of this Software is governed by a License Agreement.

NCAR Graphics is a registered trademark of the University Corporation for Atmospheric Research.

Transform class

The **Transform** class provides its subclasses with the ability to handle coordinate transformations.

Synopsis

```
Header file:             ncarg/hlu/Transform.h
Class name:              transformClass
Class pointer:           <Not referenceable>
Fortran class function:  <Not referenceable>
Superclass:              View
Composite classes:       <None>
```

Resources

Local resources

```
+---------------------------------------------------------------+
|                     Transform Resource Set                    |
|---------------------------------------------------------------|
| NAME                          TYPE                    ACCESS  |
|       CLASS                           DEFAULT                 |
|===============================================================|
| tfDoNDCOverlay                NhlTBoolean             RCSG    |
|       TfDoNDCOverlay                  False                   |
|---------------------------------------------------------------|
| tfPlotManagerOn               NhlTBoolean             RCG     |
|       TfPlotManagerOn                 True                    |
+---------------------------------------------------------------+
```

Composite resources

The **Transform** object class has no composite class resources.

Superclass resources

For any **Transform** class object, you can set all resources defined by the **Transform** class's superclasses, including:

- **View** resources

Description

Objects that belong to the **Transform** class are viewable objects that support transformations between data coordinate space and NDC space. A **Transform** object may be called simply a **transform**. Most **Transform** subclasses have their own specialized plot elements that you can render by calling the object's **Draw** method.

Transformation composite class

All subclasses of **Transform** include as a composite class member at least one subclass of the **Transformation** class to perform the forward and reverse transformations. There are currently three **Transformation** classes available to the subclasses of **Transform**:

- The **LogLinTransformation** provides transformations to and from a linear or logarithmic coordinate space. Either or both coordinate axes may be logarithmic.
- The **IrregularTransformation** provides transformations to and from rectangular coordinate spaces where each coordinate axis may be either irregularly spaced, linear, or logarithmic.
- The **MapTransformation** provides transformations to and from any of 10 different standard map projections.

PlotManager composite class

Subclasses of the **Transform** class may contain the **PlotManager** class as a composite class member. The **PlotManager** manages annotations and data space overlays on behalf of the **transform**. A **transform** instantiated with an active **PlotManager** is is known as a plot object.

Graphics primitives

In addition to their specialized plot elements, objects belonging to the **Transform** class have the ability to act as a "canvas" for drawing arbitrary graphics primitives. You may specify these primitives in data space using the **NhlDataPolyline**, **NhlDataPolygon**, and **NhlDataPolymarker** routines or in NDC space using the routines **NhlNDCPolyline**, **NhlNDCPolygon**, and **NhlNDCPolymarker**. When drawing in data space, lines between the points supplied as parameters to the **NhlDataPolyline** and **NhlDataPolygon** routines are rendered in the data space: if the space is curved, these lines will be curved as well.

These graphics primitive functions all operate in immediate mode: they appear as part of the **Workstation** output as soon as the functions execute. They are *not* part of the object in the same sense as the object's plot elements since they do not appear as a result of calling the object's **Draw** method. Currently, there is no way to save these primitives for later manipulation or reproduction.

Support functions

Transform defines the following support functions:

NhlNDCToData
: Given arrays containing the X and Y coordinates of points in NDC space, the **NhlNDCToData** function returns arrays containing the X and Y coordinates of the same points in data coordinate space.

NhlDataToNDC
: Given arrays containing the X and Y coordinates of points in data coordinate space, the **NhlDataToNDC** function returns arrays containing the X and Y coordinates of the same points in NDC space.

NhlDataPolyline
: Given arrays containing the X and Y coordinates of points in data coordinate space, the **NhlDataPolyine** function outputs an immediate-mode polyline connecting each of the points. Intermediate points between the given points along the path of the line segments in data space are sampled in order that the output polyline represents the path of "straight" line segments in data space. The resulting polyline may therefore appear curved in NDC space.

NhlNDCPolyline
: Given arrays containing the X and Y coordinates of points in NDC space, the **NhlNDCPolyine** function outputs an immediate-mode polyline connecting each of the points.

NhlDataPolygon
: Given arrays containing the X and Y coordinates of points in data coordinate space, the **NhlDataPolygon** function outputs an immediate-mode polygon consisting of a filled area and an optional edge with the points at the vertices. Intermediate points between the given points along the path of the line segments in data space are sampled in order that the edges of the output polygon represent the path of "straight" line segments in data space. The resulting polygon may therefore appear curved in NDC space.

NhlNDCPolygon
: Given arrays containing the X and Y coordinates of points in NDC space, the **NhlNDCPolygon** function outputs an immediate-mode polygon consisting of a filled area and an optional edge with the points at the vertices.

NhlDataPolymarker
: Given arrays containing the X and Y coordinates of points in data coordinate space, the **NhlDataPolymarker** function outputs immediate-mode markers at each of the points.

NhlNDCPolymarker
: Given arrays containing the X and Y coordinates of points in NDC space, the **NhlNDCPolymarker** function outputs immediate-mode markers at each of the points.

NhlIsTransform
: This function just determines if a given object id identifies a **Transform** class object.

The following support functions are used to add overlays and annotations to plot objects, and also to remove them.

NhlAddOverlay
The **NhlAddOverlay** function adds transform as an overlay of a base plot.

NhlRemoveOverlay
The **NhlRemoveOverlay** function removes an overlay from a base plot.

NhlAddAnnotation
The **NhlAddAnnotation** function adds an arbitrary user-created **View** object to a plot object as an annotation, returning the id of a controlling **AnnoManager** object.

NhlRemoveAnnotation
The **NhlRemoveAnnotation** function removes the association between a user-created **View** object and the plot object to which it belongs as an annotation. It destroys the controlling **AnnoManager** object.

Status

See also

- **Transformation** object
- **LogLinTransformation** object
- **IrregularTransformation** object
- **MapTransformation** object
- **PlotManager** object

Copyright

Copyright 1987-1999 University Corporation for Atmospheric Research
The use of this Software is governed by a License Agreement.

NCAR Graphics is a registered trademark of the University Corporation for Atmospheric Research.

VectorField class

The **VectorField** class supplies two-dimensional vector field data.

Synopsis

```
Header file:            ncarg/hlu/VectorField.h
Class name:             vectorFieldClass
```

```
Class pointer:          NhlvectorFieldClass
Fortran class pointer:  NHLFVECTORFIELDCLASS
Superclass:             DataItem
Composite classes:      <None>
```

Resources

Local resources

NAME	CLASS	TYPE	DEFAULT	ACCESS
vfDataArray	VfDataArray	NhlTGenArray	NULL	RCSG
vfUDataArray	VfUDataArray	NhlTGenArray	NULL	RCSG
vfVDataArray	VfVDataArray	NhlTGenArray	NULL	RCSG
vfXArray	VfXArray	NhlTGenArray	NULL	RCSG
vfYArray	VfYArray	NhlTGenArray	NULL	RCSG
vfPolarData	VfPolarData	NhlTBoolean	False	RCSG
vfCopyData	diCopyData	NhlTBoolean	True	RCSG
vfExchangeDimensions	VfExchangeDimensions	NhlTBoolean	False	RCSG
vfExchangeUVData	VfExchangeUVData	NhlTBoolean	False	RCSG
vfSingleMissingValue	VfSingleMissingValue	NhlTBoolean	True	RCSG
vfMissingUValueV	VfMissingUValueV	NhlTVariable	NULL	RCSG
vfMissingVValueV	VfMissingVValueV	NhlTVariable	NULL	RCSG
vfMagMinV	VfMagMinV	NhlTVariable	<Dynamic>	RCSG
vfMagMaxV	VfMagMaxV	NhlTVariable	<Dynamic>	RCSG

Table title: VectorField Resource Set

```
vfUMinV                       NhlTVariable            RCSG
      VfUMinV                         <Dynamic>
---------------------------------------------------------------
vfUMaxV                       NhlTVariable            RCSG
      VfUMaxV                         <Dynamic>
---------------------------------------------------------------
vfVMinV                       NhlTVariable            RCSG
      VfVMinV                         <Dynamic>
---------------------------------------------------------------
vfVMaxV                       NhlTVariable            RCSG
      VfVMaxV                         <Dynamic>
---------------------------------------------------------------
vfXCStartV                    NhlTVariable            RCSG
      VfXCStartV                      <Dynamic>
---------------------------------------------------------------
vfXCEndV                      NhlTVariable            RCSG
      VfXCEndV                        <Dynamic>
---------------------------------------------------------------
vfYCStartV                    NhlTVariable            RCSG
      VfYCStartV                      <Dynamic>
---------------------------------------------------------------
vfYCEndV                      NhlTVariable            RCSG
      VfYCEndV                        <Dynamic>
---------------------------------------------------------------
vfXCStartSubsetV              NhlTVariable            RCSG
      VfXCStartSubsetV                <Dynamic>
---------------------------------------------------------------
vfXCEndSubsetV                NhlTVariable            RCSG
      VfXCEndSubsetV                  <Dynamic>
---------------------------------------------------------------
vfYCStartSubsetV              NhlTVariable            RCSG
      VfYCStartSubsetV                <Dynamic>
---------------------------------------------------------------
vfYCEndSubsetV                NhlTVariable            RCSG
      VfYCEndSubsetV                  <Dynamic>
---------------------------------------------------------------
vfXCStartIndex                NhlTInteger             RCSG
      VfXCStartIndex                  <Dynamic>
---------------------------------------------------------------
vfXCEndIndex                  NhlTInteger             RCSG
      VfXCEndIndex                    <Dynamic>
---------------------------------------------------------------
vfYCStartIndex                NhlTInteger             RCSG
      VfYCStartIndex                  <Dynamic>
---------------------------------------------------------------
vfYCEndIndex                  NhlTInteger             RCSG
      VfYCEndIndex                    <Dynamic>
---------------------------------------------------------------
vfXCActualStartF              NhlTFloat               RCSG
      VfXCActualStartF                <Dynamic>
---------------------------------------------------------------
vfXCActualEndF                NhlTFloat               RCSG
      VfXCActualEndF                  <Dynamic>
---------------------------------------------------------------
vfYCActualStartF              NhlTFloat               RCSG
      VfYCActualStartF                <Dynamic>
---------------------------------------------------------------
vfYCActualEndF                NhlTFloat               RCSG
      VfYCActualEndF                  <Dynamic>
---------------------------------------------------------------
vfXCStride                    NhlTInteger             RCSG
```

```
|       VfXCStride                    1                                 |
|---------------------------------------------------------------------|
| vfYCStride                    NhlTInteger            RCSG           |
|       VfYCStride                    1                                 |
+---------------------------------------------------------------------+
```

Composite resources

The **VectorField** object class has no composite class objects.

Superclass resources

Currently, no superclasses of the **VectorField** object class define any resources.

Description

The **VectorField** class encapsulates the functionality required to pass two-dimensional vector field data to a receiving plot object. **VectorField** allows you to package the field data in two different ways: as two separate 2-dimensional arrays or as a single 3-dimensional array with two elements in the slowest varying dimension. To present the data as two 2-D arrays, set the resources *vfUDataArray* and *vfVDataArray*. To pass in a single 3-D array, set instead the resource *vfDataArray*.

VectorField handles both regularly and irregularly spaced rectangular data arrays. It assumes the grid is regularly spaced unless you set one or both of the supplementary array resources, *vfXArray* and *vfYArray*, that specify the possibly irregularly spaced location of data points along their respective data coordinate axes.

The **VectorField** object automatically converts data specified using a variety of numerical types into the NhlTFloat type expected by the receiving plot object. The types it currently handles are as follows:

- NhlTByte
- NhlTCharacter
- NhlTShort
- NhlTInteger
- NhlTLong
- NhlTFloat
- NhlTDouble

Many of the supplementary resources may also be set using any of these types. Note that scalar resources capable of accepting any type have the suffix ’V’ at the end of their name.

VectorField can pre-process the data arrays in a number of ways before handing them off to the receiving object. You can exchange the array dimensions or the U and V components of the data. You can reverse the direction of the data along either or both axes with suitable specification of the endpoints in data coordinate space. You can also make an overly dense array of data more sparse by specifying an index stride value along either or both the X and the Y dimensions.

Instead of sending the complete arrays to the receiver object, you can pass it any contiguous rectangular subset of the arrays, specified either by array index or by location in the data coordinate space. Unless the data dimension sizes also change, **VectorField** preserves the subset resource settings when the data change. This makes it easy to step through a subset of a time or level related series of data. If the dimension sizes change, **VectorField** assumes that the meaning of the data has also changed and that therefore the currently defined subset, besides potentially being out of range, is most likely not meaningful. In this case, the subset resources are reset to include all elements of the data arrays.

Support functions

The **VectorField** object does not define any support functions, but inherits all the support functions available to its superclass.

Status

1. Note that even if the values of the coordinate array resources (*vfXArray* and *vfYArray*) are actually equally spaced, plot objects such as **ContourPlot** currently use the presence of a coordinate array as a signal that the data are irregularly spaced along the axis in question. In this case the plot object uses an **IrregularTransformation** object rather than a **LogLinTransformation** object when plotting the data. The practical consequence, besides a small performance penalty, is that it will then be necessary to overlay the plot on a **LogLinPlot** if it is desired to make the X Axis logarithmic. If the axis is actually regularly spaced, you can avoid this problem by setting *vf...Array* to `NULL`, *vf...CStartV* to the value of the array's first element, and *vf...CEndV* to the value of the array's last element.

See also

- NhlCreate function
- NhlSetValues function
- NhlGetValues function
- NhlAddData function
- VectorPlot object
- DataItem object
- Base object

Copyright

Copyright 1987-1999 University Corporation for Atmospheric Research
The use of this Software is governed by a License Agreement.

NCAR Graphics is a registered trademark of the University Corporation for Atmospheric Research.

VectorPlot class

The **VectorPlot** class represents a vector field by drawing arrows at points of a grid.

Synopsis

```
Header file:             ncarg/hlu/VectorPlot.h
Class name:              vectorPlotClass
Class pointer:           NhlvectorPlotClass
Fortran class function: NHLFVECTORPLOTCLASS
Superclass:              DataComm
Composite classes:       LogLinTransformation,IrregularTransformation,PlotManager

Data specific class
Class name:              vectorPlotDataDepClass
Class pointer:           NhlvectorPlotDataDepClass
Fortran class function: NHLFVECTORPLOTDATADEPCLASS
Superclass:              DataSpec
```

Class-defined types

```
Type name:               NhlTVectorPositionMode
Definition:
typedef enum _NhlVectorPositionMode {
        NhlARROWHEAD            = 0,            /* ArrowHead    */
        NhlARROWTAIL            = 1,            /* ArrowTail    */
        NhlARROWCENTER          = 2             /* ArrowCenter  */
} NhlVectorPositionMode;

Type name:               NhlTVectorGlyphStyle
Definition:
typedef enum _NhlVectorGlyphStyle {
        NhlLINEARROW            = 0,            /* LineArrow    */
        NhlFILLARROW            = 1,            /* FillArrow    */
        NhlWINDBARB             = 2             /* WindBarb     */
} NhlVectorGlyphStyle;
```

Resources

Local resources

```
+---------------------------------------------------------------+
|               VectorPlot Resource Set                         |
|---------------------------------------------------------------|
| NAME                          TYPE                    ACCESS  |
|       CLASS                           DEFAULT                 |
|===============================================================|
| vcVectorFieldData             NhlTInteger             RCSG    |
```

Resource	Class	Type	Default	Access
	VcVectorFieldData		<none>	
vcScalarFieldData	VcScalarFieldData	NhlTInteger	<none>	RCSG
vcMapDirection	VcMapDirection	NhlTBoolean	True	RCSG
vcPositionMode	VcPositionMode	NhlTVectorPositionMode	"ArrowCenter"	RCSG
vcVectorDrawOrder	VcVectorDrawOrder	NhlTDrawOrder	"Draw"	RCSG
vcGlyphStyle	VcGlyphStyle	NhlTVectorGlyphStyle	"LineArrow"	RCSG
vcMinDistanceF	VcMinDistanceF	NhlTFloat	0.0	RCSG
vcMinMagnitudeF	VcMinMagnitudeF	NhlTFloat	0.0	RCSG
vcMaxMagnitudeF	VcMaxMagnitudeF	NhlTFloat	0.0	RCSG
vcRefMagnitudeF	VcRefMagnitudeF	NhlTFloat	0.0	RCSG
vcRefLengthF	VcRefLengthF	NhlTFloat	<dynamic>	RCSG
vcMinFracLengthF	VcMinFracLengthF	NhlTFloat	0.0	RCSG
vcLevels	Levels	NhlTFloatGenArray	<dynamic>	RCSG
vcLevelCount	VcLevelCount	NhlTInteger	16	RCSG
vcLevelSelectionMode	LevelSelectionMode	NhlTLevelSelectionMode	"AutomaticLevels"	RCSG
vcMaxLevelCount	MaxLevelCount	NhlTInteger	16	RCSG
vcLevelSpacingF	LevelSpacingF	NhlTFloat	<dynamic>	RCSG
vcMinLevelValF	MinLevelValF	NhlTFloat	<dynamic>	RCSG
vcMaxLevelValF	MaxLevelValF	NhlTFloat	<dynamic>	RCSG
vcLevelColors	VcLevelColors	NhlTColorIndexGenArray	<dynamic>	RCSG
vcUseScalarArray	VcUseScalarArray	NhlTBoolean	False	RCSG

Resource name / Class name	Type / Default	Access
vcScalarMissingValColor VcScalarMissingValColor	NhlTColorIndex "Foreground"	RCSG
vcMonoLineArrowColor VcMonoLineArrowColor	NhlTBoolean True	RCSG
vcLineArrowColor LineColor	NhlTColorIndex "Foreground"	RCSG
vcLineArrowThicknessF LineThicknessF	NhlTFloat 1.0	RCSG
vcLineArrowHeadMinSizeF VcLineArrowHeadMinSizeF	NhlTFloat 0.005	RCSG
vcLineArrowHeadMaxSizeF VcLineArrowHeadMaxSizeF	NhlTFloat 0.05	RCSG
vcFillArrowsOn VcFillArrowsOn	NhlTBoolean False	RCSG
vcMonoFillArrowFillColor VcMonoFillArrowFillColor	NhlTBoolean True	RCSG
vcFillArrowFillColor FillColor	NhlTColorIndex "Foreground"	RCSG
vcFillOverEdge VcFillOverEdge	NhlTBoolean True	RCSG
vcMonoFillArrowEdgeColor VcMonoFillArrowEdgeColor	NhlTBoolean True	RCSG
vcFillArrowEdgeColor EdgeColor	NhlTColorIndex "Background"	RCSG
vcFillArrowEdgeThicknessF EdgeThicknessF	NhlTFloat 2.0	RCSG
vcFillArrowWidthF VcFillArrowWidthF	NhlTFloat 0.1	RCSG
vcFillArrowMinFracWidthF VcFillArrowMinFracWidthF	NhlTFloat 0.25	RCSG
vcFillArrowHeadXF VcFillArrowHeadXF	NhlTFloat 0.36	RCSG
vcFillArrowHeadMinFracXF VcFillArrowHeadMinFracXF	NhlTFloat 0.25	RCSG
vcFillArrowHeadInteriorXF VcFillArrowHeadInteriorXF	NhlTFloat 0.33	RCSG
vcFillArrowHeadYF VcFillArrowHeadYF	NhlTFloat 0.12	RCSG
vcFillArrowHeadMinFracYF VcFillArrowHeadMinFracYF	NhlTFloat 0.25	RCSG

```
vcMonoWindBarbColor            NhlTBoolean             RCSG
      VcMonoWindBarbColor              True
----------------------------------------------------------------
vcWindBarbColor                NhlTColorIndex          RCSG
      LineColor                        "Foreground"
----------------------------------------------------------------
vcWindBarbLineThicknessF       NhlTFloat               RCSG
      LineThicknessF                   1.0
----------------------------------------------------------------
vcWindBarbTickAngleF           NhlTFloat               RCSG
      VcWindBarbTickAngleF             62.0
----------------------------------------------------------------
vcWindBarbTickLengthF           NhlTFloat              RCSG
      VcWindBarbTickLengthF            0.3
----------------------------------------------------------------
vcWindBarbTickSpacingF         NhlTFloat               RCSG
      VcWindBarbTickSpacingF           0.125
----------------------------------------------------------------
vcWindBarbCalmCircleSizeF      NhlTFloat               RCSG
      VcWindBarbCalmCircleSizeF        0.25
----------------------------------------------------------------
vcWindBarbScaleFactorF         NhlTFloat               RCSG
      VcWindBarbScaleFactorF           1.0
----------------------------------------------------------------
vcUseRefAnnoRes                NhlTBoolean             RCSG
      VcUseRefAnnoRes                  False
----------------------------------------------------------------
vcRefAnnoOn                    NhlTBoolean             RCSG
      VcRefAnnoOn                      True
----------------------------------------------------------------
vcRefAnnoOrientation           NhlTOrientation         RCSG
      VcRefAnnoOrientation             "Vertical"
----------------------------------------------------------------
vcRefAnnoExplicitMagnitudeF    NhlTFloat               RCSG
      VcRefAnnoExplicitMagnitudeF      0.0
----------------------------------------------------------------
vcRefAnnoArrowLineColor        NhlTColorIndex          RCSG
      LineColor                        "Foreground"
----------------------------------------------------------------
vcRefAnnoArrowFillColor        NhlTColorIndex          RCSG
      FillColor                        "Foreground"
----------------------------------------------------------------
vcRefAnnoArrowEdgeColor        NhlTColorIndex          RCSG
      EdgeColor                        "Background"
----------------------------------------------------------------
vcRefAnnoArrowUseVecColor      NhlTBoolean             RCSG
      VcRefAnnoArrowUseVecColor        True
----------------------------------------------------------------
vcRefAnnoArrowAngleF           NhlTFloat               RCSG
      VcRefAnnoArrowAngleF             0.0
----------------------------------------------------------------
vcRefAnnoArrowSpaceF           NhlTFloat               RCSG
      VcRefAnnoArrowSpaceF             2.0
----------------------------------------------------------------
vcRefAnnoArrowMinOffsetF       NhlTFloat               RCSG
      VcRefAnnoArrowMinOffsetF         0.25
----------------------------------------------------------------
vcRefAnnoString1On             NhlTBoolean             RCSG
      VcRefAnnoString1On               True
----------------------------------------------------------------
vcRefAnnoString1               NhlTString              RCSG
```

VcRefAnnoString1	<dynamic>	
vcRefAnnoString2On	NhlTBoolean	RCSG
VcRefAnnoString2On	True	
vcRefAnnoString2	NhlTString	RCSG
VcRefAnnoString2	<dynamic>	
vcRefAnnoFormat	NhlTString	RCSG
NumberFormat	"*+^sg"	
vcRefAnnoFontHeightF	NhlTFloat	RCSG
FontHeightF	<dynamic>	
vcRefAnnoTextDirection	NhlTTextDirection	RCSG
TextDirection	"Across"	
vcRefAnnoFont	NhlTFont	RCSG
Font	"pwritx"	
vcRefAnnoFontColor	NhlTColorIndex	RCSG
FontColor	"Foreground"	
vcRefAnnoFontAspectF	NhlTFloat	RCSG
FontAspectF	1.3125	
vcRefAnnoFontThicknessF	NhlTFloat	RCSG
FontThicknessF	1.0	
vcRefAnnoFontQuality	NhlTFontQuality	RCSG
FontQuality	"High"	
vcRefAnnoConstantSpacingF	NhlTFloat	RCSG
TextConstantSpacingF	0.0	
vcRefAnnoAngleF	NhlTFloat	RCSG
TextAngleF	0.0	
vcRefAnnoFuncCode	NhlTCharacter	RCSG
TextFuncCode	:	
vcRefAnnoBackgroundColor	NhlTColorIndex	RCSG
FillBackgroundColor	"Background"	
vcRefAnnoPerimOn	NhlTBoolean	RCSG
EdgesOn	True	
vcRefAnnoPerimSpaceF	NhlTFloat	RCSG
EdgeBorderWidthF	0.33	
vcRefAnnoPerimColor	NhlTColorIndex	RCSG
EdgeColor	"Foreground"	
vcRefAnnoPerimThicknessF	NhlTFloat	RCSG
EdgeThicknessF	1.0	
vcRefAnnoZone	NhlTInteger	RCSG
VcRefAnnoZone	4	
vcRefAnnoSide	NhlTPosition	RCSG
VcRefAnnoSide	"Bottom"	

vcRefAnnoJust	VcRefAnnoJust	NhlTJustification	"TopRight"	RCSG
vcRefAnnoParallelPosF	VcRefAnnoParallelPosF	NhlTFloat	1.0	RCSG
vcRefAnnoOrthogonalPosF	VcRefAnnoOrthogonalPosF	NhlTFloat	0.02	RCSG
vcMinAnnoOn	VcMinAnnoOn	NhlTBoolean	False	RCSG
vcMinAnnoOrientation	VcMinAnnoOrientation	NhlTOrientation	"Vertical"	RCSG
vcMinAnnoExplicitMagnitudeF	VcMinAnnoExplicitMagnitudeF	NhlTFloat	0.0	RCSG
vcMinAnnoArrowLineColor	LineColor	NhlTColorIndex	"Foreground"	RCSG
vcMinAnnoArrowFillColor	FillColor	NhlTColorIndex	"Foreground"	RCSG
vcMinAnnoArrowEdgeColor	EdgeColor	NhlTColorIndex	"Background"	RCSG
vcMinAnnoArrowUseVecColor	VcMinAnnoArrowUseVecColor	NhlTBoolean	True	RCSG
vcMinAnnoArrowAngleF	VcMinAnnoArrowAngleF	NhlTFloat	0.0	RCSG
vcMinAnnoArrowSpaceF	VcMinAnnoArrowSpaceF	NhlTFloat	2.0	RCSG
vcMinAnnoArrowMinOffsetF	VcMinAnnoArrowMinOffsetF	NhlTFloat	0.25	RCSG
vcMinAnnoString1On	VcMinAnnoString1On	NhlTBoolean	True	RCSG
vcMinAnnoString1	VcMinAnnoString1	NhlTString	<dynamic>	RCSG
vcMinAnnoString2On	VcMinAnnoString2On	NhlTBoolean	True	RCSG
vcMinAnnoString2	VcMinAnnoString2	NhlTString	<dynamic>	RCSG
vcMinAnnoFormat	NumberFormat	NhlTString	"*+^sg"	RCSG
vcMinAnnoFontHeightF	FontHeightF	NhlTFloat	<dynamic>	RCSG
vcMinAnnoTextDirection	TextDirection	NhlTTextDirection	"Across"	RCSG

```
vcMinAnnoFont                  NhlTFont                RCSG
      Font                             "pwritx"
----------------------------------------------------------------
vcMinAnnoFontColor             NhlTColorIndex          RCSG
      FontColor                        "Foreground"
----------------------------------------------------------------
vcMinAnnoFontAspectF           NhlTFloat               RCSG
      FontAspectF                      1.3125
----------------------------------------------------------------
vcMinAnnoFontThicknessF        NhlTFloat               RCSG
      FontThicknessF                   1.0
----------------------------------------------------------------
vcMinAnnoFontQuality           NhlTFontQuality         RCSG
      FontQuality                      "High"
----------------------------------------------------------------
vcMinAnnoConstantSpacingF      NhlTFloat               RCSG
      TextConstantSpacingF             0.0
----------------------------------------------------------------
vcMinAnnoAngleF                NhlTFloat               RCSG
      TextAngleF                       0.0
----------------------------------------------------------------
vcMinAnnoFuncCode              NhlTCharacter           RCSG
      TextFuncCode                     :
----------------------------------------------------------------
vcMinAnnoBackgroundColor       NhlTColorIndex          RCSG
      FillBackgroundColor              "Background"
----------------------------------------------------------------
vcMinAnnoPerimOn               NhlTBoolean             RCSG
      EdgesOn                          True
----------------------------------------------------------------
vcMinAnnoPerimSpaceF           NhlTFloat               RCSG
      EdgeBorderWidthF                 0.33
----------------------------------------------------------------
vcMinAnnoPerimColor            NhlTColorIndex          RCSG
      EdgeColor                        "Foreground"
----------------------------------------------------------------
vcMinAnnoPerimThicknessF       NhlTFloat               RCSG
      EdgeThicknessF                   1.0
----------------------------------------------------------------
vcMinAnnoZone                  NhlTInteger             RCSG
      VcMinAnnoZone                    5
----------------------------------------------------------------
vcMinAnnoSide                  NhlTPosition            RCSG
      VcMinAnnoSide                    "Bottom"
----------------------------------------------------------------
vcMinAnnoJust                  NhlTJustification       RCSG
      VcMinAnnoJust                    "TopRight"
----------------------------------------------------------------
vcMinAnnoParallelPosF          NhlTFloat               RCSG
      VcMinAnnoParallelPosF            1.0
----------------------------------------------------------------
vcMinAnnoOrthogonalPosF        NhlTFloat               RCSG
      VcMinAnnoOrthogonalPosF          0.02
----------------------------------------------------------------
vcNoDataLabelOn                NhlTBoolean             RCSG
      AnnotationLabelsOn               True
----------------------------------------------------------------
vcNoDataLabelString            NhlTString              RCSG
      VcNoDataLabelString              <dynamic>
----------------------------------------------------------------
vcZeroFLabelOn                 NhlTBoolean             RCSG
```

AnnotationLabelsOn	True	
vcZeroFLabelString VcZeroFLabelString	NhlTString <dynamic>	RCSG
vcZeroFLabelFormat NumberFormat	NhlTString "*+^sg"	RCSG
vcZeroFLabelFontHeightF FontHeightF	NhlTFloat <dynamic>	RCSG
vcZeroFLabelTextDirection TextDirection	NhlTTextDirection "Across"	RCSG
vcZeroFLabelFont Font	NhlTFont "pwritx"	RCSG
vcZeroFLabelFontColor FontColor	NhlTColorIndex "Foreground"	RCSG
vcZeroFLabelFontAspectF FontAspectF	NhlTFloat 1.3125	RCSG
vcZeroFLabelFontThicknessF FontThicknessF	NhlTFloat 1.0	RCSG
vcZeroFLabelFontQuality FontQuality	NhlTFontQuality "High"	RCSG
vcZeroFLabelConstantSpacingF TextConstantSpacingF	NhlTFloat 0.0	RCSG
vcZeroFLabelAngleF TextAngleF	NhlTFloat 0.0	RCSG
vcZeroFLabelFuncCode TextFuncCode	NhlTCharacter :	RCSG
vcZeroFLabelBackgroundColor FillBackgroundColor	NhlTColorIndex "Background"	RCSG
vcZeroFLabelPerimOn EdgesOn	NhlTBoolean True	RCSG
vcZeroFLabelPerimSpaceF EdgeBorderWidthF	NhlTFloat 0.33	RCSG
vcZeroFLabelPerimColor EdgeColor	NhlTColorIndex "Foreground"	RCSG
vcZeroFLabelPerimThicknessF EdgeThicknessF	NhlTFloat 1.0	RCSG
vcZeroFLabelZone VcZeroFLabelZone	NhlTInteger 0	RCSG
vcZeroFLabelSide VcZeroFLabelSide	NhlTPosition "Bottom"	RCSG
vcZeroFLabelJust VcZeroFLabelJust	NhlTJustification "CenterCenter"	RCSG

vcZeroFLabelParallelPosF	VcZeroFLabelParallelPosF	NhlTFloat	0.0	RCSG
vcZeroFLabelOrthogonalPosF	VcZeroFLabelOrthogonalPosF	NhlTFloat	0.0	RCSG
vcMagnitudeScalingMode	VcMagnitudeScalingMode	NhlTScalingMode	"ScaleFactor"	RCSG
vcMagnitudeScaleValueF	VcMagnitudeScaleValueF	NhlTFloat	1.0	RCSG
vcMagnitudeScaleFactorF	VcMagnitudeScaleFactorF	NhlTFloat	1.0	RCSG
vcMagnitudeFormat	NumberFormat	NhlTString	"*+^sg"	RCSG
vcScalarValueScalingMode	VcScalarValueScalingMode	NhlTScalingMode	"ScaleFactor"	RCSG
vcScalarValueScaleValueF	VcScalarValueScaleValueF	NhlTFloat	1.0	RCSG
vcScalarValueScaleFactorF	VcScalarValueScaleFactorF	NhlTFloat	1.0	RCSG
vcScalarValueFormat	NumberFormat	NhlTString	"*+^sg"	RCSG
vcExplicitLabelBarLabelsOn	VcExplicitLabelBarLabelsOn	NhlTBoolean	False	RCSG
vcLabelBarEndLabelsOn	VcLabelBarEndLabelsOn	NhlTBoolean	False	RCSG
vcLabelsOn	VcLabelsOn	NhlTBoolean	False	RCSG
vcLabelsUseVectorColor	VcLabelsUseVectorColor	NhlTBoolean	False	RCSG
vcLabelFontColor	FontColor	NhlTColorIndex	NhlFOREGROUND	RCSG
vcLabelFontHeightF	FontHeightF	NhlTFloat	<dynamic>	RCSG

Data specific resources

The **VectorPlot** class does not currently use any data specific resources.

Composite resources

Transformation resources

Transformation class resources specify the extent and direction of the data coordinate system. As the superclass of both **LogLinTransformation** and **IrregularTransformation**, you can access all **Transformation** resources. However, note that **VectorPlot** intercepts these resources, as follows:

- trXMinF
  By default *trXMinF* is set to the minimum data coordinate value along the X Axis, as determined from the contents of the **VectorField** object.
- trXMaxF
  By default *trXMaxF* is set to the maximum data coordinate value along the X Axis, as determined from the contents of the **VectorField** object.
- trXReverse
  By default *trXReverse* is set based on the direction of the X Axis implied by the contents of the **VectorField** object.
- trYMinF
  By default *trYMinF* is set to the minimum data coordinate value along the Y Axis, as determined from the contents of the **VectorField** object.
- trYMaxF
  By default *trYMaxF* is set to the maximum data coordinate value along the Y Axis, as determined from the contents of the **VectorField** object.
- trYReverse
  By default *trYReverse* is set based on the direction of the Y Axis implied by the contents of the **VectorField** object.

LogLinTransformation resources

The **VectorPlot** class uses the **LogLinTransformation** to handle its transformations as long as neither of the **VectorField** array resources, *vfXArray* or *vfYArray*, is set. **LogLinTransformation** has resources for selecting between linear and logarithmic coordinates for each axis.

You can access all **LogLinTransformation** resources. However, note that **VectorPlot** intercepts **LogLinTransformation** resources, as follows:

- trXLog
  VectorPlot issues a warning if *trXLog* is set True when the set value of *trXMinF* is less than or equal to 0.0. In this case it resets *trXLog* to False. If the **IrregularTransformation** resource *trXAxisType* is set, **VectorPlot** sets *trXLog* True if *trXAxisType* is set to `LogAxis`. If *trXAxisType* is set to any other value, it sets *trXLog* False.
- trYLog
  VectorPlot issues a warning if *trYLog* is set True when the set value of *trYMinF* is less than or equal to 0.0. In this case it resets *trYLog* to False. If the **IrregularTransformation** resource *trYAxisType* is set, **VectorPlot** sets *trYLog* True if *trYAxisType* is set to `LogAxis`. If *trYAxisType* is set to any other value, it sets *trYLog* False.

IrregularTransformation resources

VectorPlot automatically uses the **IrregularTransformation** instead of the **LogLinTransformation** to handle its transformations if either of the **VectorField** arrays, *vfXArray* or *vfYArray* is set, implying that one or both of the coordinate axes is irregularly spaced.

All **Transformation** superclass resources are accessible. However, **VectorPlot** blocks access to all resources specific to the **IrregularTransformation** except for:

- trXTensionF
- trYTensionF

and for the following intercepted resources:

- trXAxisType
  If the **VectorField** resource *vfXArray* is non-`NULL` and *trXAxisType* is set to any other value than `IrregularAxis`, **VectorPlot** switches to a coordinate extent bounded by 0 and the length of the X-Axis dimension minus one. If *trXAxisType* is not set, but the **LogLinTransformation** resource *trXLog* is set, **VectorPlot** sets *trXAxisType* to `LogAxis` if *trXLog* is True; if *trXLog* is False, it changes *trXAxisType* to `LinearAxis` if it had been set to `LogAxis` and leaves it unchanged otherwise. *trXAxisType* can be set to `LogAxis` without error only when the X-Axis coordinate extent as passed from the **VectorField** is entirely positive. If this is not the case, *trXAxisType* will default to `LinearAxis`.

  V4.1 Status Note 1
- trYAxisType
  If the **VectorField** resource *vfYArray* is non-`NULL` and *trYAxisType* is set to any other value than `IrregularAxis`, **VectorPlot** switches to a coordinate extent bounded by 0 and the length of the Y-Axis dimension minus one. If *trYAxisType* is not set, but the **LogLinTransformation** resource *trYLog* is set, **VectorPlot** sets *trYAxisType* to `LogAxis` if *trYLog* is True; if *trYLog* is False, it changes *trYAxisType* to `LinearAxis` if it had been set to `LogAxis` and leaves it unchanged otherwise. *trYAxisType* can be set to `LogAxis` without error only when the Y-Axis coordinate extent as passed from the **VectorField** is entirely positive. If this is not the case, *trYAxisType* will default to `LinearAxis`.

  V4.1 Status Note 1

PlotManager resources

If *tfPlotManagerOn* is True when a **VectorPlot** object is created, you can access all **PlotManager** resources. However, note that **VectorPlot** intercepts certain **PlotManager** resources, as follows:

- pmLabelBarDisplayMode
  The default value of *pmLabelBarDisplayMode* is set to `Never`.
- pmLegendDisplayMode
  The *pmLegendDisplayMode* resource is hard-coded to its default value of NhlNOCREATE in **VectorPlot**.
- pmTickMarkDisplayMode
  The default value of *pmTickMarkDisplayMode* is set to `Conditional`.
- pmTitleDisplayMode
  The default value of *pmTitleDisplayMode* is set to `Conditional`.

Except for **Legend**, you can access resources for any of the composite classes of the **PlotManager** class. However, the **PlotManager** class modifies the access and behavior of some of the resources belonging to these classes, as follows:

- TickMark resources as modified by PlotManager
- Title resources as modified by PlotManager
- LabelBar resources as modified by PlotManager

The **VectorPlot** class itself modifies the access and behavior of a number of **LabelBar** resources.

Additional modifications to LabelBar resources

The **VectorPlot** class disables a number of **LabelBar** resources, since it sets them automatically based on the current values of certain relevant **VectorPlot** resources. The disabled resources include:

- lbBoxCount
- lbMonoFillColor
- lbFillColor
- lbFillColors
- lbMonoFillPattern
- lbFillPattern
- lbFillPatterns
- lbMonoFillScale
- lbFillScaleF
- lbFillScales

In addition, the **VectorPlot** class intercepts certain **LabelBar** resources, as follows:

- lbLabelStrings
  Any set value of this resource is ignored unless the **VectorPlot** resource *vcExplicitLabelBarLabelsOn* is set True.
- lbLabelFuncCode
  Any set value of this resource is ignored unless the **VectorPlot** resource *vcExplicitLabelBarLabelsOn* is set True.
- lbLabelAlignment
  Any set value of this resource is ignored unless the **VectorPlot** resource *vcExplicitLabelBarLabelsOn* is set True.

Superclass resources

You can set all resources defined by the superclasses of the **VectorPlot** object class, including:

- **DataComm** resources
- **Transform** resources
- **View** resources

Description

A **VectorPlot** object, or **vectorplot**, represents a vector field by drawing glyphs that represent magnitude and direction at grid points based on two-dimensional data provided by an instance of the **VectorField** class. Three glyph styles are available: a basic line-drawn arrow, a filled arrow with an optional edge line, and a standard wind barb. Each of the glyph styles may be individually colored based on their magnitude or another associated scalar field. The edge of the filled arrow may have a separate color from the interior. In the rest of this document the word "arrow" may refer generically to any of the three glyph styles. **VectorPlot** intrinsically supports several annotational items, including tick marks, titles, one or two reference vector glyphs with explanatory text, and a labelbar that provides a key to the color scheme used with individually colored vector arrows. These annotations automatically adjust themselves to the current state of the **vectorplot**.

The **VectorPlot** class provides many resources that you can use to tailor the plot output to your taste. However, only one resource must be set explicitly in order to generate a meaningful vector plot. You need focus only on resources that modify the **VectorPlot** features important to you. All the others will default to a generally usable value. Resources that provide control over related attributes of the various **VectorPlot** elements have related names. As an example, all resources associated with the reference vector annotation begin with the prefix *vcRefAnno*.

Data input

The only resource you must set in order to generate an actual vector plot is *vcVectorFieldData*. This resource specifies the id of an existing **VectorField** object. The **vectorfield** accepts data in a variety of types and configurations, and allows you to specify how it is to be interpreted. As received by the **vectorplot**, the data are accompanied by information specifying the extents of the data in the data coordinate system, the minimum and maximum data magnitudes, as well as the minimum and maximum values of each component of the vector. Also, if either or both of the data coordinate axes are irregularly spaced, the **vectorfield** communicates information defining this spacing.

If you want to color the vector arrows individually based on data other than the vector magnitudes, you must set the boolean resource *vcUseScalarArray* True and also supply the **vectorplot** with a **ScalarField** object. You do this by setting the resource *vcScalarFieldData* using the id of an existing **scalarfield**. Currently, **VectorPlot** requires that the **scalarfield** have the same number of elements along each dimension (after subsetting and striding is performed) as does each component of the **vectorfield**. It also assumes that the data coordinate extents of the **scalarfield** match those of the **vectorfield** and ignores any explicitly set **scalarfield** coordinate extent values.

Coordinate transformations

VectorPlot intrinsically supports linear, logarithmic, and irregular rectangular gridded data coordinate spaces. It does *not yet*, on its own, support the transformation of an irregular data space into either a linear or a logarithmic space. However, such transformations are easily accomplished using the overlay mechanism.

VectorPlot instantiates child objects to manage transformations between the data coordinate system and NDC space. The **LogLinTransformation** manages linear and logarithmic transformations, and the

IrregularTransformation manages the transformation if one or both axes are irregularly spaced. By default the data coordinate extents are based on the extents of the supplied **VectorField** object data, but you may adjust them using resources belonging to the transformation objects.

Use of the LogLinTransformation object

VectorPlot uses a **LogLinTransformation** object as long as the **VectorField** resources *vfXArray* and *vfYArray* are both `NULL`. The coordinate extents of a linear axis may arbitrarily intersect or encompass the data extent. If you set a logarithmic axis, then the coordinate extent of that axis must be entirely positive, but otherwise may intersect or extend beyond the data extent.

Use of the IrregularTransformation object

If either of the **VectorField** coordinate array resources *vfXArray* and *vfYArray* are non-`NULL`, then **VectorPlot** uses an **IrregularTransformation** object. Note that **VectorPlot** treats an axis with an associated coordinate array as irregular even if the coordinate array actually has evenly spaced values. **VectorPlot** represents an irregular axis not by stretching and compressing various regions of the plot, but by displaying it with irregularly spaced tick marks.

In addition to a small performance penalty, there are some restrictions associated with use of **IrregularTransformation** object. Although you may limit the coordinate extent to a subspace of the data coordinate extent of the **VectorField** object data, you are not allowed to define a coordinate range that extends outside the range of the data coordinates of an irregular axis. Using the **IrregularTransformation** resources trXAxisType or trYAxisType, it is possible to set an irregular axis to `LinearAxis` or even, under certain conditions, to `LogAxis`, but the results are probably not what you want. Since **VectorPlot** does not intrinsically support a linearization transformation for irregularly spaced data, it can only switch to a linear system by replacing the data coordinates with array index coordinates, which are, in fact, linearly spaced. To properly transform irregularly spaced data into a linear or logarithmic coordinate system, you must use the overlay mechanism (V4.1 Status Note 1).

Overlays

In addition to the built-in transformation support, you can map a **vectorplot** into a variety of other coordinate spaces by adding it as an overlay to a base plot. You can overlay a **vectorplot** on a **mapplot** to transform the data into any of 10 different map projections. You can transform irregularly gridded vector data into a linear or logarithmic space by overlaying the **vectorplot** on a **loglinplot**. You can also make a **vectorplot** into an overlay of any other plot object, including a **contourplot**, an **irregularplot**, a **streamlineplot**, or an **xyplot**. Use the **NhlAddOverlay** function to perform an overlay.

Mapping vector direction

In certain situations, such as when the coordinate system is irregular or when the units along each coordinate axis are different, you may want to locate the vectors within the coordinate space, but it may not make sense to transform their direction relative to the space. For instance, suppose you create a wind profile plot with degrees of longitude along the X Axis and geopotential height along the Y Axis. Assume the vector data represent the eastward and northward horizontal components of the flow in meters per second. It is clear in this case that the direction of the vectors is unrelated to the coordinate system in which they are located.

VectorPlot has a boolean resource called *vcMapDirection* that allows you to specify whether the direction as well as the location is to be mapped into the coordinate space defined by the transformation currently in effect. When you set this resource False, the direction is rendered in an independent uniform coordinate system. The U component of the vector data will be parallel to the X Axis, and the V component will be parallel to the Y Axis with units of equal size in each direction.

Draw order

The **VectorPlot** class allows you specify when the vector arrows are drawn in relation to other plot elements of an overlay. You control the drawing order using the *vcVectorDrawOrder* resource. There are three drawing phases: the *predraw* phase, the *draw* phase, and the *postdraw* phase. By default, the **VectorPlot** object draws its arrows during the draw phase. When a **VectorPlot** is drawn by itself, the drawing order is not important. However, when a **vectorplot** is an overlay, you often need to adjust the drawing order to ensure that the features you most want to see remain visible.

Note that **VectorPlot** annotations are always drawn during the *postdraw* phase.

Annotations

Like all plot objects, a **vectorplot** is by default instantiated with a **plotmanager** to manage overlays and annotations on its behalf. **VectorPlot** supplies several annotations of its own in addition to enabling three **PlotManager** intrinsic annotations. The annotations supplied by **VectorPlot** include two vector reference annotations and a zero field message label. These are described in more detail later on. The enabled **PlotManager** annotations include **TickMark**, **Title**, and **LabelBar**. **VectorPlot** displays tick marks by default. Titles will appear if you set the appropriate title string resource to a non-NULL string value. The **labelbar**, which provides a key to the color scheme used with individually colored arrows, only appears if you set *pmLabelBarDisplayMode* appropriately. As with any plot object, you can also add arbitrary user-defined external annotations to a **vectorplot**.

All these annotations (**TickMark** and **Title** partially excepted) subscribe to the locational conventions defined by the PlotManager Location Control Model.

Glyph styles

VectorPlot supports three glyph styles: a basic line-drawn arrow, a filled arrow with an optional edge line, and a standard wind barb representation. Each style has its own set of resources, distinguishable by name. By default, **VectorPlot** uses line-drawn arrows. Set the *vcGlyphStyle* resource to select the desired glyph style.

Line-drawn arrows

The resource *vcMonoLineArrowColor* controls whether line-drawn arrows are to use a single color for every vector. If set to its default value, True, you specify the single arrow color using the resource *vcLineArrowColor*. Otherwise, the arrows are individually colored based on the array resources *vcLevels* and *vcLevelColors* and the scalar value associated with the vector.

Line-drawn vector arrows have a fixed shape that the user cannot modify. However, you can set a minimum and maximum size for the vector arrowhead using the resources *vcLineArrowHeadMinSizeF* and *vcLineArrowHeadMaxSizeF*. This allows you to ensure that the arrowheads remain recognizable for small-magnitude vectors while not becoming execessively large for high-magnitude vectors.

You can set the thickness of the polylines used to render line-drawn arrows using the resource *vcLineArrowThicknessF*. You may notice that as the lines are made thicker, the arrows begin to take on a rather fuzzy look. This is an unavoidable consequence of using thick lines, (at least where there is no control of the line join method), and one of the motivations for the development of filled vector arrows.

Filled arrows

VectorPlot draws filled vector arrows using a solid-filled area with an edge outline. Filled arrows have several advantages over line-drawn arrows:

- The edges can be sharply defined no matter how wide you make the arrow body or head.
- There is more area available to apply colors to the arrow and therefore the colors are more visible.
- You have much greater control over the shape of the vector arrows.
- Since the edge can be drawn in a separate color from the fill, you can distinguish the individual vector arrows more easily in areas where the vectors overlap, or where the fill has a similar color to the background color.

Naturally, filled arrows have a few drawbacks as well:

- They take a bit longer to draw.
- If you are overlaying the **vectorplot** on another plot, such as a **contourplot**, the filled arrows may obscure too much of the information underneath unless you set the fill color to `Transparent`.
- Since there are more resources to play with, you may find yourself spending more time figuring out how they work and what looks best for your data.

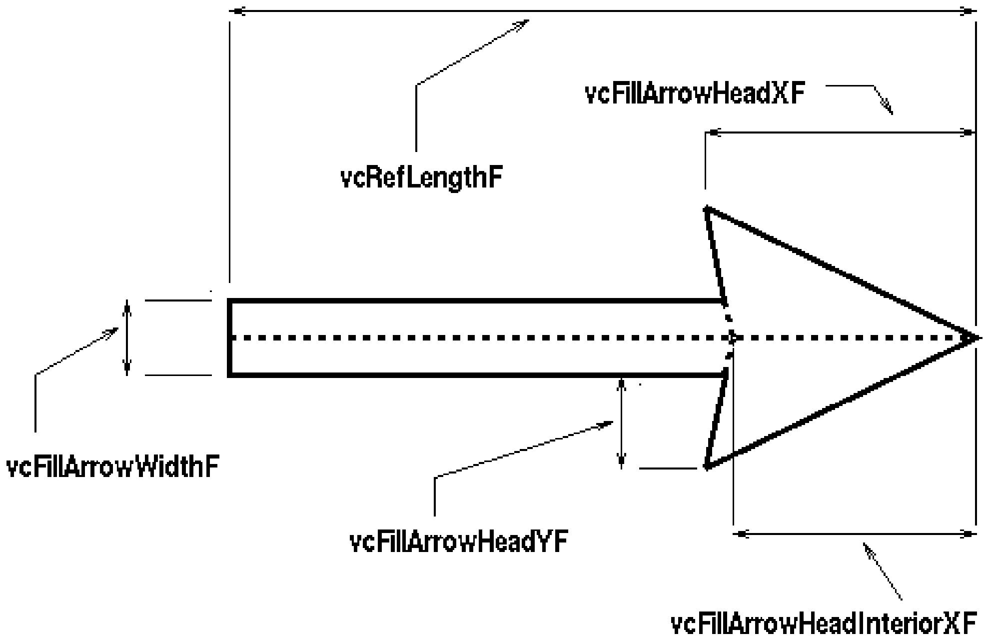
Either the filled area or the edge or both may be colored individually based on the values in the *vcLevelColors* array resource. The resource *vcMonoFillArrowFillColor* controls whether to draw the fill using a single color. If set True, the resource *vcFillArrowFillColor* specifies the fill color; otherwise the *vcLevelColors* array applies. Likewise *vcMonoFillArrowEdgeColor* controls whether the edge should be a single color. If set True, *vcFillArrowEdgeColor* specifies the fill color; if False, the *vcLevelColors* array once again applies. You can eliminate either the edge or the filled portion of a filled arrow by setting the appropriate color resource to `Transparent` (-1).

The resource *vcFillOverEdge* specifies whether the edge should be drawn first with the fill on top or vice versa. Drawing the fill on top ensures that the full extent of the fill area appears, even when the vector arrow is very small. However, for this to work properly, the resource *vcFillArrowEdgeThicknessF* should be set to a value greater than 1.0. Otherwise the edges will appear broken and poorly formed. Since the default value of *vcFillOverEdge* is True, *vcFillArrowEdgeThicknessF* is set by default to 2.0.

There are seven resources for controlling the shape of filled vector arrows. These allow for many variations not only in the appearance of the arrows at the reference length but also in the way the appearance changes as the length of the arrow changes with vector magnitude. The figure below indicates which part of the arrow each of the following resources specifies:

- *vcFillArrowWidthF*
- *vcFillArrowHeadXF*
- *vcFillArrowHeadInteriorXF*
- *vcFillArrowHeadYF*

Each of these resources is specified as a fraction of the value of *vcRefLengthF*.



The resources *vcFillArrowMinFracWidthF*, *vcFillArrowHeadMinFracXF*, and *vcFillArrowHeadMinFracYF* set minimum sizes for *vcFillArrowWidthF*, *vcFillArrowHeadXF*, and *vcFillArrowHeadYF* respectively. They are specified as fractions of the resource with which they are associated. So, for example, if you set *vcFillArrowMinFracWidthF* to 0.25, the width used for the minimum length arrow will be equal to 0.25 * *vcFillArrowWidthF* * *vcRefLengthF* NDC units. The widths used for intermediate-length arrows will be sized proportionally between the minimum width and the reference width. This implies that if you set *vcFillArrowMinFracWidthF* to 1.0, the arrow width will remain constant for all filled arrows, no matter what their length. Setting any of these *...MinFrac...* resources to 0.0 causes their associated arrow dimension to vary in strict proportion to the arrow length.

See the example vc03n for an illustration of some of the stylistic variations that are possible using the fill arrow resources.

Wind barbs

Wind barb glyphs consist of a shaft parallel to the vector direction, and zero or more pennants and/or ticks spaced along the shaft starting at the end nearest the direction from which the flow is coming. However, if the magnitude of the velocity is less than 2.5, a circle replaces the shaft. Half ticks represent 5 units of magnitude, full ticks represent 10 units, and pennants represent 50 units. By convention, the units are knots. The pennants are drawn using a filled polygon, while the ticks, the shaft, and the calm circle are all rendered with polylines. Unlike the other glyph styles, wind barbs maintain a basically uniform length for all magnitudes. You can control this length with the resource *vcRefLengthF*.

You can use the resource *vcWindBarbScaleFactorF* to convert vector fields represented with other units into knots in order to render wind speeds according to the usual convention. Note that this resource operates independently of the resources *vcMagnitudeScaleFactorF* and *vcMagnitudeScaleValueF*, both of which only apply to **VectorPlot** annotations.

The resource *vcMonoWindBarbColor* controls whether all wind barb glyphs appear in a single color. If set to its default value, True, you specify this color using the resource *vcWindBarbColor*. Otherwise, the wind barb glyphs are individually colored based on the array resources *vcLevels* and *vcLevelColors* and the scalar value associated with the vector.

vcWindBarbAngleF controls the angle of the ticks and the hypotenuse of the pennant triangles. It is measured in degrees clockwise from the vector direction. Use a negative value to represent southern hemisphere wind barbs, which, by convention, have their ticks on the opposite side of shaft from northern hemisphere wind barbs. *vcWindBarbTickLengthF* controls the length of the wind barb ticks and the hypotenuse of the triangles forming the pennants. *vcWindBarbTickSpacingF* controls the spacing between ticks along the shaft. Pennants are spaced by half this amount. The resource *vcWindBarbCalmCircleSizeF* sets the size of the circle used to render magnitudes less than 2.5. Set the thickness of the polyline elements of the wind barb glyphs using the resource *vcWindBarbLineThicknessF*.

Arrow length, spacing, and position

One of the difficulties of representing a vector field using arrows at the grid points is that when the transformation is non-linear or a dense dataset is used, the arrows become bunched together in areas or are too crowded everywhere. The vectors must either become so small that it is difficult to distinguish between them, or they overlap and degenerate into an amorphous mass. Although there is no absolute cure for this problem, **VectorPlot** provides a number of features to help you present the data as coherently as possible.

Reference length and magnitude

If *vcGlyphStyle* is set to `LineArrow` or `FillArrow`, the length of each vector arrow is determined in relation to a vector reference magnitude specified by the resource *vcRefMagnitudeF* and a vector reference length specified in NDC units by the resource *vcRefLengthF*. Vectors whose magnitude is equal to the reference magnitude are drawn at the reference length. By default, the reference magnitude is the maximum magnitude in the **vectorfield**, but it may be set to any arbitrary value. The default reference length is determined dynamically based on the viewport and the number of elements along each data dimension. Note that the length of each vector arrow increases as you increase the reference length but decreases as you increase the reference magnitude.

If *vcGlyphStyle* is set to `WindBarb`, things work a bit differently: vector magnitude is now represented by varying the number of ticks and/or pennants on the wind barb, rather than by varying the size of the glyph as a whole. **VectorPlot** draws most wind barbs using a uniform length, determined solely by the value of the *vcRefLengthF* resource. This length actually consists of the length of the wind barb shaft plus the projection of a full wind barb tick onto the axis of the shaft. Therefore, wind barbs with only a half tick or with no tick are a bit shorter than the length specified by *vcRefLengthF*. The *vcRefMagnitudeF* resource has no effect on the length of wind barb glyphs. However, it does still have a role to play in setting up the reference vector annotation.

Minimum length and magnitude

By default, except when *vcGlyphStyle* is set to `WindBarb`, the size of each vector arrow differs from the reference length by the ratio of its magnitude to the reference magnitude. In practice, a common result of this strictly proportional representation is that low-magnitude vectors become too small to be rendered with full detail, particularly on low-resolution devices. As a result, their direction, especially, becomes difficult to decipher. Therefore, **VectorPlot** provides a resource, *vcMinFracLengthF*, that allows you to specify, as a fraction of the reference length, a minimum vector arrow length. When this resource is set to a value greater than 0.0, the smallest magnitude vector is rendered at the specified fraction of the reference length, and intermediate magnitude vectors are rendered at proportionally intermediate lengths. Setting *vcMinFracLengthF* to 1.0 causes all vector arrows to be drawn at the reference length. If *vcGlyphStyle* is set to `WindBarb`, setting *vcMinFracLengthF* has no effect: effectively it is forced to the value 1.0.

The resource *vcMinMagnitudeF* specifies the minimum magnitude a vector must have in order to qualify for drawing. You can use it to eliminate low-velocity vectors in order to concentrate attention on the dominant features of a vector field. However, note that *vcMinMagnitudeF* also has another role. When it has a non-zero value and *vcMinFracLengthF* is also non-zero, *vcMinMagnitudeF* specifies the vector magnitude that is rendered at the minimum length specified by *vcMinFracLengthF*.

Ensuring uniform arrow sizing over a series of plots

In order to ensure that within a series of plots of related data a particular length always represents the same vector magnitude, you must always first set the reference magnitude, *vcRefMagnitudeF*, explicitly. Otherwise, whatever magnitude happens to be the maximum in each plot will be rendered at the reference length. In addition, if you set *vcMinFracLengthF* to a non-zero value, you should be careful to set *vcMinMagnitudeF* non-zero as well. Otherwise the length specified by *vcMinFracLengthF* will be used to represent whatever magnitude happens to be the minimum in each plot. If you do not want any low-magnitude vectors actually eliminated, simply set *vcMinMagnitudeF* to a small positive number that is less than the smallest magnitude in any of your datasets.

Controlling the spacing between vector arrows

There are two ways to reduce the crowding of a too-densely populated **vectorplot**. One is to specify a stride greater than unity along one or both dimensions of the **vectorfield**, using the **VectorField** resources *vfXCStride* and/or *vfYCStride*. This method has the effect of reducing the density uniformly throughout the data space, and may be the most appropriate if your plot is overly dense throughout. The second way is to set the resource *vcMinDistanceF* to an NDC value representing the minimum distance you want to allow between the locations of neighboring vector arrows. This method requires a bit more computing time, but it ensures that the minimum distance between vectors is maintained even after arbitrary non-linear transformations, like many map projections, from data space to NDC space.

Arrow position

Using the resource *vcPositionMode*, you can control whether the tail, the head, or the center of a vector arrow is positioned at the location of the grid point in data space.

Selecting levels

In order to support vector arrows individually colored according to a scalar value, **VectorPlot** uses an array of level values to subdivide the range of scalars. By default, the levels apply to the range of vector magnitudes. However, if *vcUseScalarArray* is set True and a **scalarfield** has been provided, the levels will apply to the range of the **scalarfield** data. The level array resource *vcLevels* is specified in the same way as the **ContourPlot** resource *cnLevels*.

By appropriately setting the *vcLevelSelectionMode* resource, you can choose from four level selection modes to set up the *vcLevels* array. The default mode, `AutomaticLevels`, is easy to use. It selects a "nice" spacing starting from the smallest relatively round number greater than the minimum data value, such that the number of levels is as close as possible to, but less than, the value of the resource *vcMaxLevelCount*. `EqualSpacedLevels` mode defines exactly *vcMaxLevelCount* levels spaced evenly from the data minimum value to the data maximum value. In `ManualLevels` mode, you set the maximum and minimum levels using the resources *vcMinLevelValF* and *vcMaxLevelValF* with a spacing defined by the resource *vcLevelSpacingF*. Finally, `Explicit` mode allows you to define the level values yourself by setting the array resource *vcLevels*. The `ManualLevels` and `ExplicitLevels` modes have the advantage that they are independent of the maximum and minimum values of the particular dataset you are plotting, and therefore can be used to enforce consistency in the plots of a series of related datasets. `ExplicitLevels` is the only mode that allows you to establish variably spaced levels.

Once you have established the levels, you can retrieve the number of levels actually used by getting the value of the read-only resource *vcLevelCount*.

The reference vector and minimum vector annotations

VectorPlot supports two annotations that each display a single vector arrow sandwiched between optional text. One of these, the reference vector annotation, appears by default. You control it using resources with the prefix *vcRefAnno*. By default, it displays a vector arrow drawn at the reference length along with a text string representing the reference magnitude above and the string "Reference Vector" below. The second, the minimum vector annotation, must be turned on explicitly in order to appear. Its resources have the prefix *vcMinAnno...*, and by default it displays an arrow drawn at the minimum length along with text representing the minimum magnitude above and the words "Minimum Vector" below. When the arrow-sizing resources are set such that the relation between arrow length and vector magnitude is not strictly proportional, the minimum vector annotation can be useful for helping to convey the relationship between the various arrow sizes in the plot.

The names given to these annotations are based on the way **VectorPlot** configures them by default. However, they are both capable of displaying arrows representing any arbitrary magnitude (using resources *vcRefAnnoExplicitMagnitudeF* or *vcMinAnnoExplicitMagnitudeF*), as well as arbitrary text. In addition, **VectorPlot** recognizes a set of substitution substrings that allows you to place within this text a number of key values drawn dynamically from the data, including maximum and minimum values of the magnitude, of each of the vector components, and of the associated **ScalarField** data, if present.

There is a boolean resource called *vcUseRefAnnoRes*, that when set True causes the minimum vector annotation (as well as the zero field annotation) to use the same values for many of the appearance-related resources that the reference vector annotation uses. This resource acts as a shortcut for achieving a consistent appearance.

In other respects, the reference and the minimum vector annotations function identically. The rest of this section discusses only the reference vector annotation but applies equally to the minimum vector annotation if you simply substitute the equivalent minimum vector annotation resource.

Reference vector annotation size and position

You control the size of the reference vector annotation indirectly, based on the text string length, the font height and aspect ratio, and the amount of space occupied by the arrow. Besides the size of the arrow itself, there are three resources that affect how much space the arrow will occupy. These are:

- *vcRefAnnoArrowAngleF*
- *vcRefAnnoArrowSpaceF*
- *vcRefAnnoArrowMinOffsetF*

The resources that control the position of the reference vector annotation follow the locational conventions of the PlotManager Location Control Model. They include:

- *vcRefAnnoZone*
- *vcRefAnnoSide*
- *vcRefAnnoJust*
- *vcRefAnnoParallelPosF*
- *vcRefAnnoOrthogonalPosF*

Reference vector annotation arrow color

By default, the reference vector annotation arrow takes its colors from the colors specified for the arrows in the plot. If the arrows in the plot are individually colored based on vector magnitude, the reference vector arrow will be colored according to the magnitude it represents. Otherwise, the colors used for the annotation arrow will be as specified by *vcLineArrowColor* for line-drawn arrows and wind barbs or *vcFillArrowFillColor* and *vcFillArrowEdgeColor* for filled arrows. On the other hand, if you set the resource *vcRefAnnoArrowUseVecColor* False, you can set the annotation arrow colors explicitly, using *vcRefAnnoArrowLineColor* for line-drawn arrows and wind barbs or *vcRefAnnoArrowFillColor* and *vcRefAnnoArrowEdgeColor* for filled arrows.

Text strings

The reference vector annotation has a complete set of text attribute resources for controlling font height, aspect ratio, color and so forth. These resources apply to both of the annotation's strings.

You control the content of each of the two text strings in the reference vector annotation and whether they should appear at all using the following resources:

- *vcRefAnnoString1On*

- *vcRefAnnoString1*
- *vcRefAnnoString2On*
- *vcRefAnnoString2*

String 1 appears above or to the left of the arrow, and String 2 appears below or to the right of the arrow.

There are a number of special substitution substrings that you can insert anywhere in either of the text strings. They will be replaced with substrings representing numerical values, as follows:

VMG
: The magnitude represented by the annotation's arrow.

RFM
: The value of the reference magnitude used in representing the vector field.

MNM
: The minimum vector magnitude (the value of the **VectorField** resource *vfMagMinV*).

MXM
: The maximum vector magnitude (the value of the **VectorField** resource *vfMagMaxV*).

MNU
: The minimum value of the U component of the vector field (the value of the **VectorField** resource *vfUMinV*).

MXU
: The maximum value of the U component of the vector field (the value of the **VectorField** resource *vfUMaxV*).

MNV
: The minimum value of the V component of the vector field (the value of the **VectorField** resource *vfVMinV*).

MXU
: The maximum value of the V component of the vector field (the value of the **VectorField** resource *vfVMaxV*).

MSF
: The scale factor used for vector magnitudes (value of *vcMagnitudeScaleFactorF*).

MNS
: The minimum value in the scalar dataset used to color the vectors (the value of the **ScalarField** resource *sfDataMinV*).

MXS
: The maximum value in the scalar dataset used to color the vectors (the value of the **ScalarField** resource *sfDataMaxV*).

SSF
: The scale factor used for values in the scalar dataset (value of *vcScalarValueScaleValueF*).

Zero field annotation

VectorPlot provides a zero field annotation that outputs a diagnostic message if you attempt to draw an instance using a **vectorfield** that contains only zero-magnitude vectors or missing values. This annotation also outputs a message if you draw without setting the *vcVectorFieldData* resource to the id of an existing **vectorfield**. You can individually control the contents of each of these messages as well as whether they should appear at all. The applicable resources are:

- *vcZeroFLabelOn*
- *vcZeroFLabelString*

for controlling the zero field message, and

- *vcNoDataLabelOn*
- *vcNoDataLabelString*

for the no data message.

All the remaining resources for the zero field annotation have the prefix *vcZeroFLabel*. These include a complete set of text attribute resources, as well as resources for controlling position according to the locational conventions of the PlotManager Location Control Model.

Numerical formatting and scaling

You can control the format of numerical values that are related to the data when they appear in VectorPlot annotations, including the **labelbar** and the two vector annotations. You can also scale these numbers in a uniform manner, as, for instance, in order to represent the field using units different from the data itself. Formatting is controlled using a format string constructed according to the HLU Floating Point Format Specification scheme. A separate set of resources controls these features for magnitude-related values and **ScalarField** data values. For magnitude-related values, the following resources apply:

- *vcMagnitudeScalingMode*
- *vcMagnitudeScaleValueF*
- *vcMagnitudeScaleFactorF*
- *vcMagnitudeFormat*

whereas **ScalarField** data are affected by the following resources:

- *vcScalarValueScalingMode*
- *vcScalarValueScaleValueF*
- *vcScalarValueScaleFactorF*
- *vcScalarValueFormat*

Status

1. The support for irregular transformations is at a transistional stage. Eventually, **VectorPlot** will be able to perform transformations from irregular coordinates to linear and logarithmic coordinates without using the overlay mechanism. This will eliminate the need for a switch to the index coordinate system.

See also

- NhlCreate function
- NhlDestroy function
- NhlSetValues function
- NhlGetValues function
- NhlAddData function
- VectorField object
- ScalarField object
- PlotManager object
- Title object
- TickMark object
- LabelBar object
- AnnoManager object
- DataComm object
- Transform object
- View object
- Base object

Copyright

Copyright 1987-1999 University Corporation for Atmospheric Research
The use of this Software is governed by a License Agreement.

NCAR Graphics is a registered trademark of the University Corporation for Atmospheric Research.

View class

The **View** class provides its subclasses with the ability to be drawn.

Synopsis

```
Header file:             ncarg/hlu/View.h
Class name:              viewClass
Class pointer:           <Not referenceable>
Fortran class function: <Not referenceable>
Superclass:              Base
Composite classes:       <None>
```

Class-defined types

```
Type name:               NhlTOrientation
Definition:
typedef enum _NhlOrientation {
        NhlHORIZONTAL           = 0,    /* "Horizontal" */
        NhlVERTICAL             = 1     /* "Vertical"   */
} NhlOrientation;
```

```
Type name:              NhlTPosition
Definition:
typedef enum _NhlPosition {
        NhlTOP                  = 0,    /* "Top"        */
        NhlBOTTOM               = 1,    /* "Bottom"     */
        NhlRIGHT                = 2,    /* "Right"      */
        NhlLEFT                 = 3,    /* "Left"       */
        NhlCENTER               = 4     /* "Center"     */
} NhlPosition;

Type name:              NhlTJustification
Definition:
typedef enum _NhlJustification {
        NhlTOPLEFT              = 0,    /* "topLeft"      */
        NhlCENTERLEFT           = 1,    /* "CenterLeft"   */
        NhlBOTTOMLEFT           = 2,    /* "BottomLeft"   */
        NhlTOPCENTER            = 3,    /* "TopCenter"    */
        NhlCENTERCENTER         = 4,    /* "CenterCenter" */
        NhlBOTTOMCENTER         = 5,    /* "BottomCenter" */
        NhlTOPRIGHT             = 6,    /* "TopRight"     */
        NhlCENTERRIGHT          = 7,    /* "CenterRight"  */
        NhlBOTTOMRIGHT          = 8     /* "BottomRight"  */
} NhlJustification;

Type name:              NhlTDrawOrder
Definition:
typedef enum _NhlDrawOrder {
        NhlPREDRAW              = 0,    /* "PreDraw"    */
        NhlDRAW                 = 1,    /* "Draw"       */
        NhlPOSTDRAW             = 2     /* "PostDraw"   */
} NhlDrawOrder;
```

Resources

Local resources

```
+----------------------------------------------------------------+
|                        View Resource Set                       |
|----------------------------------------------------------------|
| NAME                          TYPE                    ACCESS   |
|       CLASS                           DEFAULT                  |
|================================================================|
| vpXF                          NhlTFloat               RCSG     |
|       VpXF                            0.2                      |
|----------------------------------------------------------------|
| vpYF                          NhlTFloat               RCSG     |
|       VpYF                            0.8                      |
|----------------------------------------------------------------|
| vpWidthF                      NhlTFloat               RCSG     |
|       VpWidthF                        0.6                      |
|----------------------------------------------------------------|
| vpHeightF                     NhlTFloat               RCSG     |
|       VpHeightF                       0.6                      |
|----------------------------------------------------------------|
| vpOn                          NhlTBoolean             RCSG     |
```

```
|       VpOn                              True                      |
-------------------------------------------------------------------
| vpKeepAspect                  NhlTBoolean             RCSG       |
|       VpKeepAspect                      False                     |
-------------------------------------------------------------------
| vpUseSegments                 NhlTBoolean             RCG        |
|       VpUseSegments                     False                     |
-------------------------------------------------------------------
| vpAnnoManagerId                NhlTObjId              G          |
|       VpAnnoManagerId                   NullObjId                 |
+-----------------------------------------------------------------+
```

Composite resources

The **View** object class has no composite class resources.

Superclass resources

The superclass of the **View** object class does not define any resources.

Description

The **View** class is the superclass of all viewable objects: that is, all objects that can be drawn within the NDC viewspace. A viewable object is also called simply a view. It contains resources to specify the location and size of each viewable object in NDC units. It allows you to control whether the aspect ratio of the object may change after the object is created. It also allows you to ask any view with segment support to store its image into a segment, in order that it may subsequently redraw itself faster and more efficiently.

A number of **View** subclasses support the use of *segments* in order to speed the task of redrawing themselves when their contents have not changed. A *segment* is a stored record of all the primitives drawn as a result of drawing a particular instance of the class. Segments may be transformed to a different size or shape in NDC space. Therefore when segment drawing is in effect, changes to the **View** resources *vpXF*, *vpYF*, *vpWidthF*, or *vpHeightF* can be handled by redrawing the segment rather than completely redrawing the object. You set the resource *vpUseSegements* True in order to activate the use of segments.

You may make any view into an external annotation of a plot object. In this case, the plot object creates an **AnnoManager** object that you can use to set the position of the view relative to the base plot's viewport. Once a view becomes an annotation, the base plot takes control of positioning, drawing, and usually sizing it. If a view is currently functioning as an annotation, you may retrieve the id of its **AnnoManager** using the resource *vpAnnoManagerId*. If the view is not currently an annotation, the value of *vpAnnoManagerId* will be `NullObjId` (0).

Support functions

The **View** class defines the following support functions:

GetBB
Given the id of a **View** class object, this function returns the coordinates of the left, right, top, and bottom edges of the object's bounding box in NDC units.

NhlIsView
This function just determines if a given object id identifies a **View** class object.

Status

See also

- GetBB function
- Base object
- PlotManager object
- AnnoManager object

Copyright

Copyright 1987-1999 University Corporation for Atmospheric Research
The use of this Software is governed by a License Agreement.

NCAR Graphics is a registered trademark of the University Corporation for Atmospheric Research.

Workspace class

The **Workspace** object manages large blocks of memory on behalf of a number of HLU library objects.

Synopsis

```
Header file:             ncarg/hlu/Workspace.h
Class name:              workspaceClass
Class pointer:           <Not referenceable>
Fortran class function:  <Not referenceable>
Superclass:              Obj
Composite classes:       <None>
```

Resources

Local resources

Workspace Resource Set

NAME	CLASS	TYPE	DEFAULT	ACCESS
wsMaximumSize	WsMaximumSize	NhlTLong	16777216	RCSG
wsThresholdSize	WsThresholdSize	NhlTLong	4194304	RCSG
wsCurrentSize	WsCurrentSize	NhlTLong	0	RCSG

Composite resources

The **Workspace** object class has no composite class objects.

Superclass resources

The superclass of the **Workstation** object class does not define any resources.

Description

The **Workspace** object manages large blocks of memory, known as *workspaces*, that a number of the low-level routines require in order to perform their function. It does *not* manage the dynamic memory needed for the instantiation of the HLU object themselves. The **Workspace** object class is like the **Error** object class in that each HLU program creates one and only one object of this class. This single global **Workstation** object manages the workspace needs of all HLU plot objects. As a user, you are allowed neither to create nor destroy a **Workspace** object. You may, however, set and retrieve several **Workspace** resources.

Most of the time, the **Workspace** object operates transparently, and you do not need to be concerned with how it works. However, based on the memory resources available to your processes, you may wish to tune the performance of your HLU applications by adjusting the resources of the **Workspace** object. Since you do not create the **Workspace**, you cannot get or set any of its resources without first retrieving its object id through a call to the support function **NhlGetWorkspaceObjectId**.

In order to use the resources of the **Workspace** object effectively, you should know a little bit about its operation. Plot objects may create and destroy workspaces. Once created, a workspace may be in one of two states: *in use* or *idle*. The **Workspace** object keeps track of the total number of bytes currently allocated for workspaces. You can retrieve this value at any time (after you get the **Workspace** object id) by getting the value of the resource *wsCurrentSize*. Any time a plot object requests to use a workspace, the **Workspace** object checks to see if the request would cause the number of currently allocated bytes to go over a threshold value, as set by the resource *wsThresholdSize*. If so, the **Workspace** object looks through the list of idle workspaces, freeing space until *either* the size of the requested space added to the remaining space no longer causes *wsThresholdSize* to be exceeded *or* no more idle workspaces remain. Note that the HLU writer can specify that the contents of a workspace be preserved. In this case, the **Workspace** object writes the contents of the workspace to a file before freeing the space.

Notice that *wsThresholdSize* puts no limit on the total memory space available to workspaces that are currently in use. However, depending on the number of separate plot objects created by the application, smaller values will tend to reduce the total amount of space in use at a single time, at a cost of more frequent allocation and deallocation operations on these (potentially very large) memory spaces. The resource *wsMaximumSize*, on the other hand, specifies the maximum amount of memory the **Workspace** object allows before returning a fatal error. You can use this resource as a kind of brake that prevents runaway applications from using all the memory resources available in an environment.

Support functions

The **Workspace** object defines the following support function:

NhlGetWorkspaceObjectId
The **GetWorkspaceObjectId** function returns the object id of the global **Workspace** object.

Status

See also

- GetWorkspaceObjectId function
- Base object

Copyright

Copyright 1987-1999 University Corporation for Atmospheric Research
The use of this Software is governed by a License Agreement.

NCAR Graphics is a registered trademark of the University Corporation for Atmospheric Research.

Workstation class

A **Workstation** object manages an instance of an output device.

Synopsis

```
Header file:             ncarg/hlu/Workstation.h
Class name:              workstationClass
Class pointer:           <Not referenceable>
Fortran class function:  <Not referenceable>
Superclass:              Base
Composite classes:       <None>
```

Class-defined types

```
Type name:               NhlTColorIndex
Definition:              typedef int NhlColorIndex;
                                 Valid Range is -1 to 255
                                 -1:     NhlTRANSPARENT
                                  0:     NhlBACKGROUND
                                  1:     NhlFOREGROUND

Type name:               NhlTColorMap
Definition:              A name of a pre-defined colormap, or an array
                         of a combination of RGB values, named colors,
                         and other color specification formats.

Type name:               NhlTColorIndexGenArray
Definition:              An array of NhlTColorIndex elements.

Type name:               NhlTColorDefinitionGenArray
Definition:              A one-dimensional array defining an RGB triplet,
                         or a string containing a color name from
                         $NCARG_ROOT/lib/ncarg/database/rgb.txt.

Type name:               NhlTDashIndex
Definition:              typedef int NhlDashIndex;
                                 Valid Range is 0 to NhlNwkDashTableLength
                                  0:     NhlSOLIDLINE

Type name:               NhlTDashIndexGenArray
Definition:              An array of NhlTDashIndex elements.

Type name:               NhlTFillIndex
Definition:              typedef int NhlFillIndex;
                                 Valid Range is -1 to NhlNwkFillTableLength
                                 -1:     NhlHOLLOWFILL
                                  0:     NhlSOLIDFILL

Type name:               NhlTFillIndexGenArray
Definition:              An array of NhlTFillIndex elements.

Type name:               NhlTMarkerIndex
Definition:              typedef int NhlMarkerIndex;
                                 Valid Range is 0 to NhlNwkMarkerTableLength
```

```
                         0:     "default"
                         1:     "dot"
                         2:     "+"
                         2:     "plus"
                         3:     "*"
                         3:     "asterisk"
                         4:     "hollow_circle"
                         5:     "cross"
                         5:     "x"
                         6:     "hollow_square"
                         7:     "up_triangle"
                         8:     "down_triangle"
                         9:     "diamond"
                        10:     "left_triangle_filled"
                        11:     "right_triangle_filled"
                        12:     "star_5point"
                        13:     "star_6point"
                        14:     "circle_w_dot"
                        15:     "circle_w_cross"
                        16:     "circle_filled"

Type name:              NhlTMarkerIndexGenArray
Definition:             An array of NhlTMarkerIndex elements.

Type name:              NhlTMarkLineMode
Definition:
typedef enum _NhlMarkLineMode {
        NhlLINES        = 0,    /* "Lines"      */
        NhlMARKERS      = 1,    /* "Markers"    */
        NhlMARKLINES    = 2     /* "MarkLines"  */
} NhlMarkLineMode;

Type name:              NhlTMarkLineModeGenArray
Definition:             An array of NhlTMarkLineMode elements.
```

Resources

Local resources

```
+-----------------------------------------------------------------+
|                       Workstation Resource Set                  |
|-----------------------------------------------------------------|
| NAME                           TYPE                    ACCESS   |
|       CLASS                            DEFAULT                  |
|=================================================================|
| wkColorMap                     NhlTColorMap    RCSG             |
|       WkColorMap                       <Dynamic>                |
|-----------------------------------------------------------------|
| wkColorMapLen                  NhlTInteger             G        |
|       WkColorMapLen                    <Dynamic>                |
|-----------------------------------------------------------------|
| wkBackgroundColor      NhlTColorDefinitionGenArray     RCSG     |
|       WkBackgroundColor                (/0.0,0.0,0.0/)          |
|-----------------------------------------------------------------|
| wkForegroundColor      NhlTColorDefinitionGenArray     RCSG     |
|       WkForegroundColor                <Dynamic>                |
|-----------------------------------------------------------------|
```

wkDashTableLength	WkDashTableLength	NhlTInteger	<Dynamic>	G
wkFillTableLength	WkFillTableLength	NhlTInteger	<Dynamic>	G
wkMarkerTableLength	WkMarkerTableLength	NhlTInteger	<Dynamic>	G
wkGksWorkId	WkGksWorkId	NhlTInteger	<Dynamic>	G
wkDefGraphicStyleId	WkDefGraphicStyleId	NhlTInteger	<Dynamic>	G

Composite resources

The **Workstation** object class has no composite class objects.

Superclass resources

The superclasses of the **Workstation** object class do not define any resources.

Description

In the context of the HLU library, the **Workstation** object class represents the interface to a generalized output device. It is not designed to be instantiated directly. Instead there exist object classes derived from the **Workstation** object that are specialized for particular types of output. You must always create an object belonging to the **Workstation** class for any program using the HLU library. All drawable objects must be created with a **Workstation** parent. Currently there are three object classes subclassed from the **Workstation** class: **XWorkstation** for drawing into a window of a workstation running the X Window System, **PSWorkstation** for creating a PostScript file, and **NcgmWorkstation** for creating an NCAR Computer Graphics Metafile.

The **Workstation** object provides access to a color map and tables of line dash patterns, area fill patterns, and markers. There is a default colormap that is shared by all subclasses of **Workstation** objects, but it is also possible to create an individual colormap for any instance of a **Workstation** object. The dash pattern, fill pattern, and marker tables are shared by all **Workstation** objects and are currently read-only. Eventually, however, there will be methods for creating individual user-defined versions of each of these tables.

The **Workstation** colormap may contain up to 256 different colors, including the background color. You specify the color of features to be drawn using an HLU color index in the range 0 through 255. An *allocated* color index is one for which an RGB triple defining a color has been specified. Color index 0 specifies the background color and color index 1 specifies the foreground color. The **Workstation** colormap is defined to repeat itself. So, if you specify a color index that is greater than any of the currently defined indexes, the index will be mapped into the correct range. If you specify a currently unallocated color index for a feature, the **Workstation** object will use the foreground color. (This refers to color indexes that are unallocated, but are less than another currently defined index. In other words, the holes in the colormap are filled with the foreground color.) You may allocate a complete colormap using the *wkColorMap* resource; there are also support functions for allocating and freeing individual colors.

All **Workstation** class objects share the ability to update and clear the **Workstation** drawing area. The **UpdateWorkstation** and **ClearWorkstation** functions perform different tasks depending on the class of **Workstation** object for which they are invoked. For the **NcgmWorkstation**, the clear and update functions write special CGM commands into the CGM file that end the current frame and start a new one. For the **XWorkstation**, the update function causes all pending draw commands to be drawn, and the clear function clears the entire window. There is also a convenience function, **Frame**, that calls both the update and clear functions. This function operates much like the **Frame** call in the low level NCAR graphics libraries.

Support functions

The **Workstation** object defines the following support functions:

UpdateWorkstation
: The **UpdateWorkstation** function updates the drawing area of an instance of a **Workstation** object.

ClearWorkstation
: The **ClearWorkstation** function clears the drawing area of an instance of a **Workstation** object.

Frame
: The **Frame** function first updates and then clears the drawing area of an instance of a **Workstation** object.

NewColor
: The **NewColor** function adds a color specified as an RGB triple to the colormap of a **Workstation** instance and returns its HLU color index.

SetColor
: The **SetColor** function sets the RGB color values for a specified HLU color index.

FreeColor
: The **FreeColor** function removes a color, specified by its HLU index, from the **Workstation** color map.

IsAllocatedColor
: The boolean function **IsAllocatedColor** returns True if a color has been assigned to a particular HLU color index, and False otherwise.

NhlIsWorkstation
: This function just determines if a given object id identifies a **Workstation** class object.

Status

See also

- NhlCreate function
- NhlDestroy function
- NhlSetValues function
- NhlGetValues function
- XWorkstation object
- PSWorkstation object
- NcgmWorkstation object
- Base object

Copyright

Copyright 1987-1999 University Corporation for Atmospheric Research
The use of this Software is governed by a License Agreement.

NCAR Graphics is a registered trademark of the University Corporation for Atmospheric Research.

XWorkstation class

The **XWorkstation** manages communication with the NCAR Graphcs X Workstation input/output device.

Synopsis

```
Header file:            ncarg/hlu/XWorkstation.h
Class name:             xWorkstationClass
Class pointer:          NhlxWorkstationClass
Fortran class function: NHLFXWORKSTATIONCLASS
Superclass:             Workstation
Composite classes:      <None>
```

Class-defined types

```
Type name:              NhlTXColorMode
Definition:             typedef enum {
                                NhlSHARE        = 0,    /* "Share","Shared"*/
                                NhlPRIVATE      = 1,    /* "Private"    */
                                NhlMIXED        = 2     /* "Mixed"      */
                        } NhlXColorMode;
```

Resources

Local resources

XWorkstation Resource Set

NAME	CLASS	TYPE	DEFAULT	ACCESS
wkWindowId	WkWindowId	NhlTInteger	NULL	RCG
wkXColorMode	WkXColorMode	NhlTXColorMode	<Dynamic>	RCG
wkPause	WkPause	NhlTBoolean	<Dynamic>	RCSG

Composite resources

The **XWorkstation** object class has no composite class objects.

Superclass resources

You can set all resources defined by the superclass of the **XWorkstation** object class, including:

- **Workstation** resources

Description

The **XWorkstation** object is derived from the **Workstation** class and thus inherits all of the color map management and control functions associated with the **Workstation** class.

The **XWorkstation** adds three new resources to the **Workstation** set. These resources enable X Window programmers to seemlessly integrate their GUI's and the HLU libraries. The most significant of these resources is *wkWindowId*. This resource allows users to pass in the X window ID into which all children of a given **XWorkstation** object will draw. This means the users can embed NCAR graphics in their own GUI's. The only restriction to this is that the window id passed in through *wkWindowId* must have been created using the default visual.

The **XWorkstation** object can still be used without having to pass in a window id. If the resource *wkWindowId* is not set at create time then the **XWorkstation** object will create and manage its own X Window. To configure the size of the window that the **XWorkstation** object creates, you need to add a resource entry into your .Xdefaults file. The following is an example of how to set the geometry of the **XWorkstation** window to 800x800 pixels:

```
xgks*geometry: 800x800
```

Eventually the XWorkstation object will accept an X-Window color map id as a resource but currently this feature is not implemented.

The final feature added to the **Workstation** class by the **XWorkstation** object is the ability to pause and wait for user input. This feature only applies when the resource *wkWindowId* is not set at create time. You control this feature using the resource *wkPause*. When set True, the **XWorkstation** will pause during any call to the **ClearWorkstation Frame** functions and wait for the user to press a mouse button or press a key.

Support functions

The **XWorkstation** object does not define any support functions, but inherits all the support functions available to its superclass.

Status

See also

- NhlCreate function
- NhlDestroy function
- NhlSetValues function
- NhlClearWorkstation function
- Frame function
- XWorkstation object
- Base object

Copyright

Copyright 1987-1999 University Corporation for Atmospheric Research
The use of this Software is governed by a License Agreement.

NCAR Graphics is a registered trademark of the University Corporation for Atmospheric Research.

XyPlot class

The **XyPlot** object draws curves made up of X/Y coordinate pairs with tick marks, titles, and a legend.

Synopsis

```
Header file:            ncarg/hlu/XyPlot.h
Class name:             xyPlotClass
Class pointer:          NhlxyPlotClass
Fortran class function: NHLFXYPLOTCLASS
Superclass:             DataComm
Composite classes:      PlotManager,LogLinTransformation,IrregularTransformation
```

```
Data specific class
Class name:             xyDataSpecClass
Class pointer:          <Not referenceable>
Fortran class function: <Not referenceable>
Superclass:             DataSpec
```

Class-defined types

```
Type name:              NhlTAlternatePlace
Definition:
typedef enum _NhlAlternatePlace{
        NhlNONE         = 0,    /* "none"       */
        NhlLEFTAXIS     = 1,    /* "leftAxis"   */
        NhlRIGHTAXIS    = 2,    /* "rightAxis"  */
        NhlTOPAXIS      = 3,    /* "topAxis"    */
        NhlBOTTOMAXIS   = 4     /* "bottomAxis" */
} NhlAlternatePlace;

Type name:              NhlTLineLabelMode
Definition:
typedef enum _NhlLineLabelMode{
        NhlNOLABELS     = 0,    /* "noLabels"   */
        NhlLETTERED     = 1,    /* "lettered"   */
        NhlCUSTOM       = 2     /* "custom"     */
} NhlLineLabelMode;
```

Resources

Local resources

```
+---------------------------------------------------------------+
|                       XyPlot resource set                     |
|---------------------------------------------------------------|
| NAME                          TYPE                    ACCESS  |
|       CLASS                           DEFAULT                 |
|===============================================================|
| xyCoordData                   NhlTObjIdGenArray       CSG     |
|       XyCoordData                     NULL                    |
|---------------------------------------------------------------|
| xyCoordDataSpec               NhlTObjIdGenArray       G       |
|       XyCoordDataSpec                 NULL                    |
|---------------------------------------------------------------|
| xyCurveDrawOrder              NhlTDrawOrder           RCSG    |
|       XyCurveDrawOrder                "Draw"                  |
```

xyXStyle	XyXStyle	NhlTTickMarkStyle	NhlLINEAR	RCSG
xyYStyle	XyYStyle	NhlTTickMarkStyle	NhlLINEAR	RCSG
xyXIrregularPoints	XyXIrregularPoints	NhlTFloatGenArray	NULL	RCSG
xyYIrregularPoints	XyYIrregularPoints	NhlTFloatGenArray	NULL	RCSG
xyXIrrTensionF	XyXIrrTensionF	NhlTFloat	2.0	RCSG
xyYIrrTensionF	XyYIrrTensionF	NhlTFloat	2.0	RCSG
xyComputeXMin	XyComputeExtent	NhlTBoolean	<dynamic>	RCSG
xyComputeXMax	XyComputeExtent	NhlTBoolean	<dynamic>	RCSG
xyComputeYMin	XyComputeExtent	NhlTBoolean	<dynamic>	RCSG
xyComputeYMax	XyComputeExtent	NhlTBoolean	<dynamic>	RCSG

Data-specific resources

The following resources are not set directly to the **XyPlot** object. These resources are used to set the graphical attributes that correspond directly with an individual piece of data that is added to the plot. So the attributes of each individual curve or marker are set this way.

When a **DataItem** object is added to the *xyCoordData* resource, either using NhlAddData or using NhlSetValues, an **XyDataSpec** object is automatically created as a child of the **XyPlot** to hold the resources that describe these graphical attributes. The **XyDataSpec** object is given the same name as the **DataItem** object that was added.

To set these resources in a resource file, you would use resource lines similar to this:

```
*xyplotname.dataspecname.xyDashPattern: solidline
```

xyplotname
: the name you give the **XyPlot** when you create it.

dataspecname
: the name of the **XyDataSpec** object, which will be the same as the name of the **DataItem** object that was added to the *xyCoordData* resource.

To set these resources programmatically, you need to retrieve the **XyDataSpec** object id, and call NhlSetValues on that object.

There are two ways to retrieve the ids for the **XyDataSpec** objects that hold the resources that describe the graphical attributes for a particular **DataItem**:

1. If you used NhlAddData to add the **DataItem** to the *xyCoordData* resource, then the return value from that function is the id of the **XyDataSpec** object that describes the graphical attributes for the added **DataItem**.
2. You can retrieve the value of the xyCoordDataSpec resource of the **XyPlot** after all the data have been added to the **XyPlot**. It is an array containing the ids of all the **XyDataSpec** objects that are currently describing how all the **DataItem** objects in the *xyCoordData* resource are to be visualized.

The **XyDataSpec** class also inherits all the **DataSpec** resources.

XyDataSpec resource set

NAME	CLASS	TYPE	DEFAULT	ACCESS
xyDashPattern	LineDashPattern	NhlTDashIndex	NhlSOLIDLINE	RSG
xyDashPatterns	XyDashPatterns	NhlTDashIndexGenArray	NULL	RSG
xyMonoDashPattern	XyMonoDashPattern	NhlTBoolean	False	RSG
xyMarkLineMode	XyMarkLineMode	NhlTMarkLineMode	NhlLINES	RSG
xyMarkLineModes	XyMarkLineModes	NhlTMarkLineModeGenArray	NULL	RSG
xyMonoMarkLineMode	XyMonoMarkLineMode	NhlTBoolean	False	RSG
xyExplicitLegendLabels	XyExplicitLegendLabels	NhlTStringGenArray	NULL	RSG
xyLineColor	LineColor	NhlTColorIndex	NhlFOREGROUND	RSG
xyLineColors	XyLineColors	NhlTColorIndexGenArray	NULL	RSG
xyMonoLineColor	XyMonoLineColor	NhlTBoolean	False	RSG
xyLineDashSegLenF	LineDashSegLenF	NhlTFloat	<Dynamic>	RCSG
xyLineLabelFontHeightF	FontHeightF	NhlTFloat	<Dynamic>	RCSG
xyLineLabelFontColor	FontColor	NhlTColorIndex	NhlFOREGROUND	RSG

xyLineLabelFontColors XyLineLabelFontColors	NhlTColorIndexGenArray	NULL	RSG
xyMonoLineLabelFontColor XyMonoLineLabelFontColor	NhlTBoolean	False	RSG
xyLabelMode XyLabelMode	NhlTLineLabelMode	NhlNOLABELS	RSG
xyExplicitLabels XyExplicitLabels	NhlTStringGenArray	NULL	RSG
xyLineThicknessF LineThicknessF	NhlTFloat	1.0	RSG
xyLineThicknesses XyLineThicknesses	NhlTFloatGenArray	NULL	RSG
xyMonoLineThickness XyMonoLineThickness	NhlTBoolean	False	RSG
xyMarker MarkerIndex	NhlTMarkerIndex	"default"	RSG
xyMarkers XyMarkers	NhlTMarkerIndexGenArray	NULL	RSG
xyMonoMarker XyMonoMarker	NhlTBoolean	False	RSG
xyMarkerColor MarkerColor	NhlTColorIndex	NhlFOREGROUND	RSG
xyMarkerColors XyMarkerColors	NhlTColorIndexGenArray	NULL	RSG
xyMonoMarkerColor XyMonoMarkerColor	NhlTBoolean	False	RSG
xyMarkerSizeF MarkerSizeF	NhlTFloat	0.01	RSG
xyMarkerSizes XyMarkerSizes	NhlTFloatGenArray	NULL	RSG
xyMonoMarkerSize XyMonoMarkerSize	NhlTBoolean	False	RSG
xyMarkerThicknessF MarkerThicknessF	NhlTFloat	1.0	RSG
xyMarkerThicknesses XyMarkerThicknesses	NhlTFloatGenArray	NULL	RSG
xyMonoMarkerThickness XyMonoMarkerThickness	NhlTBoolean	False	RSG
xyLineLabelFont Font	NhlTFont	"pwritx"	RCSG

```
| xyLineLabelFontAspectF        NhlTFloat              RCSG     |
|       FontAspectF                     1.3125                  |
|---------------------------------------------------------------|
| xyLineLabelFontThicknessF     NhlTFloat              RCSG     |
|       FontThicknessF                  1.0                     |
|---------------------------------------------------------------|
| xyLineLabelFontQuality        NhlTFontQuality        RCSG     |
|       FontQuality                     "High"                  |
|---------------------------------------------------------------|
| xyLineLabelConstantSpacingF   NhlTFloat              RCSG     |
|       TextConstantSpacingF            0.0                     |
|---------------------------------------------------------------|
| xyLineLabelFuncCode           NhlTCharacter          RCSG     |
|       TextFuncCode                    ':'                     |
+---------------------------------------------------------------+
```

Composite resources

Transformation resources

Transformation class resources specify the extent and direction of the data coordinate system. As the superclass of both **LogLinTransformation** and **IrregularTransformation**, you can access all **Transformation** resources. **XyPlot** modifies some of these resources, as follows:

trXMinF
: If this resource is not explicitly set, it is determined based on the value of the *xyComputeXMin* resource and the data extents within the *xyCoordData* resource.

trXMaxF
: If this resource is not explicitly set, it is determined based on the value of the *xyComputeXMax* resource and the data extents within the *xyCoordData* resource.

trYMinF
: If this resource is not explicitly set, it is determined based on the value of the *xyComputeYMin* resource and the data extents within the *xyCoordData* resource.

trYMaxF
: If this resource is not explicitly set, it is determined based on the value of the *xyComputeYMax* resource and the data extents within the *xyCoordData* resource.

LogLinTransformation resources

If the **XyPlot** *xyXStyle* and *xyYStyle* resources indicate the use of a logarithmic or linear coordinate system, it uses a **LogLinTransformation** object to manage the transformation to and from data space.

All **Transformation** superclass resources are accessible. However, **XyPlot** blocks access to all resources specific to the **LogLinTransformation**.

IrregularTransformation resources

If the **XyPlot** *xyXStyle* and *xyYStyle* resources indicate the use of an irregular coordinate system, it uses an **IrregularTransformation** object to manage the transformation to and from data space.

All **Transformation** superclass resources are accessible. However, **XyPlot** blocks access to all resources specific to the **IrregularTransformation** except for:

- trXTensionF
- trYTensionF

PlotManager resources

If *tfPlotManagerOn* is True when an XyPlot object is created, all **PlotManager** resources are available. However, **XyPlot** does modify some of the **PlotManager** resources as follows:

pmTitleDisplayMode
: The default value of *pmTitleDisplayMode* is set to `Conditional`.

pmTickMarkDisplayMode
: The default value of *pmTickMarkDisplayMode* is set to `Conditional`.

pmLegendDisplayMode
: The default value of *pmLegendDisplayMode* is set to `Never`.

pmLabelBarDisplayMode
: The *pmLabelBarDisplayMode* resource is hard-coded to its default value of `NoCreate` in the **XyPlot**.

The resources for the **PlotManager's** *Composite* parts are available as follows:

Title resources
: All the Title resources are available as they are modified by **PlotManager**.

TickMark resources
: All the TickMark resources are available as modified by **PlotManager**.

LabelBar resources
: All LabelBar resources are disabled by **XyPlot**.

Legend resources
: The Legend resources are available as modified by **PlotManager** with the exception of some resources that have been disabled since they are set automatically based on the current values of relevant **XyPlot** and **XyDataSpec** resources. These resources include:
  - lgItemCount
  - lgLabelStrings
  - lgMonoItemType
  - lgItemType
  - lgItemTypes
  - lgMonoDashIndex
  - lgDashIndex
  - lgDashIndexes
  - lgMonoLineColor
  - lgLineColor
  - lgLineColors
  - lgMonoLineThickness
  - lgLineThicknessF
  - lgLineThicknesses
  - lgMonoLineDashSegLen
  - lgLineDashSegLenF

- lgLineDashSegLens
- lgMonoMarkerIndex
- lgMarkerIndex
- lgMarkerIndexes
- lgMonoMarkerColor
- lgMarkerColor
- lgMarkerColors
- lgMonoMarkerThickness
- lgMarkerThicknessF
- lgMarkerThicknesses
- lgMonoMarkerSize
- lgMarkerSizeF
- lgMarkerSizes
- lgMonoLineLabelFontHeight
- lgLineLabelFontHeightF
- lgLineLabelFontHeights
- lgMonoLineLabelFontColor
- lgLineLabelFontColor
- lgLineLabelFontColors
- lgLineLabelStrings
- lgLineLabelFontAspectF
- lgLineLabelFontThicknessF
- lgLineLabelFontQuality
- lgLineLabelConstantSpacingF
- lgLineLabelFuncCode

Superclass resources

All of the **DataComm** resources can be set for the **XyPlot** object.

Description

XyPlot is a composite object. It combines the functionality of the **PlotManager** object, **LogLinTransformation**, and **IrregularTransformation** to create a fully functional **XyPlot** object.

Since the **XyPlot** is a composite object, the resources for the *Composite* classes are also available to the **XyPlot** object through the standard resource-setting mechanisms.

XyPlot is derived from the **DataComm** class, which is derived from the **Transform** class. This means that the functions NhlDataToNDC and NhlNDCToData can be used to obtain transformations of points to and from data coordinates and NDC coordinates. See the Transform class for more information on these functions.

In general, **XyPlot** accepts one or more vectors of X and Y coordinates and displays them by drawing a line through the {X,Y} coordinate pairs, or drawing markers at each point within the coordinate pairs, or both.

XyPlot provides the user with the capability of configuring the line thickness, dash pattern indexes, colors of the lines, line labels, and line label colors for curves. **XyPlot** allows the user to configure Marker indexes, colors, sizes, and thicknesses for displaying markers. **XyPlot** also provides all the same transformation styles that are provided by the **TickMark** object: LOG, LINEAR, and IRREGULAR.

XyPlot accepts **DataItem** objects rather than arrays of values. The *xyCoordData* resource is used to associate **DataItem** objects with the **XyPlot** object. The *xyCoordData* resource is used to specify a list of **DataItem** objects, and the **AddData** function appends **DataItems** to the list.

There are basically two steps to visualizing a set of data in the XyPlot object. First, the programmer has to fully describe the data to the HLU environment. This is done by creating a DataItem object that describes the data's shape. Second, the programmer has to associate the **DataItem** with the *xyCoordData* resource of the **XyPlot**.

You can associate data with **XyPlot** by creating a **DataItem** to describe the data. Then use the NhlAddData function to add that **DataItem** to the list of data referenced by the *xyCoordData* resource. If the **XyPlot** object understands data in the format provided (i.e. the **DataItem** class used), the **NhlAddData** function will return with a value of **NhlNOERROR**.

Additionally, the programmer can just set the *xyCoordData* resource directly using the NhlSetValues function. The *xyCoordData* resource is set with an array of object ids. The difference between setting the resource directly and calling the **NhlAddData** function is that the **NhlAddData** function appends the given data to the current list of data, while setting the *xyCoordData* resource using **NhlSetValues** replaces the current value of that resource.

If programmers want to control the **XyDataSpec** resources programmatically, they will need to add the **DataItem**s to the **XyPlot**'s *xyCoordData* resource first. Then they can retrieve the value of the *xyCoordDataSpec* resource. The *xyCoordDataSpec* resource is an array of **XyDataSpec** ids. The elements of this array correspond to the **DataItem** ids in the *xyCoordData* array resource. To configure exactly how each coordinate pair array of one of the **DataItem**s is going to be displayed, the programmer can set the **XyDataSpec** resources to the corresponding **XyDataSpec** id.

The **XyDataSpec** object is created on behalf of the programmer for each **DataItem** object that is added to the *xyCoordData* resource. The **XyDataSpec** object is created with the same name as the **DataItem** and as a child of the **XyPlot**. Therefore, the data-specific resources (i.e. the resources of the **XyDataSpec** object) can be set in the resource file by simply specifying the name of the **DataItem** as a child of the **XyPlot** with the **XyDataSpec** resources in the resource file.

The **XyDataSpec** resources allow the user to set dash patterns, line colors, line thicknesses, line labels, and line label font colors and heights for each element of the *xyCoordData* resource. The user can also set the other standard text attributes for line labels. However, these attributes are only available as scalar resources, and therefore the **XyDataSpec** object referenced by the first element of the *xyCoordDataSpec* array supplies the values of these resources for all data elements.

The *xyLabelMode* and *xyExplicitLabels* resources control how each curve is labeled. The curve line labels can be any string. However, line labels longer than six or seven characters can cause problems, and spaces are not drawn.

XyPlot also allows markers to be plotted. The *xyMarkLineMode*, *xyMarkLineModes*, and *xyMonoMarkLineMode* resources are used to indicate if markers should be plotted in place of curves, or in addition to them. There are also a number of other resources in the **XyDataSpec** object to configure exactly how the markers should be displayed.

The next step in configuring an **XyPlot** is to select and specify the data transformation of the **XyPlot** object. As is the case with the **TickMark** object, there are currently three styles of data transformations, `Log`, `Linear`, and `Irregular`. There are two style resources for setting the X-Axis and Y-Axis data transformations. This differs from the **TickMark** object, where each side of each axis has its own resource for specifying style. **XyPlot** forces both sides of an axis to have the same style, so there is no need for two separate style resources that set the style for each side of an axis. This has been done to eliminate the possibility that the tick marks don't accurately represent the data transformation. If different styles were allowed for both sides of the X Axis, then the possibility arises that one of the transformations is wrong. The two resources for selecting the transformation style are *xyXStyle* and *xyYStyle*. By default, these are both set to `Linear`.

Once the style is selected, the resources *trXMinF*, *trXMaxF*, *trYMinF*, and *trYMaxF* are used to set the data extents. If these resources are not set, then the **XyPlot** object examines the data and determines for itself what the data extents are. The actual data extents could change if additional data are added to the *xyCoordData* resource or if one of the current **DataItem**s is modified. The *tr* data-extent resources will be automatically recomputed if the *xyComputeXMin*, *xyComputeXMax*, *xyComputeYMin*, and *xyComputeYMax* resources allow it.

`Irregular` style transformations require some additional information. The resources *xyXIrregularPoints* and *xyYIrregularPoints* must be used when `Irregular` style has been selected. These resources apply to both sides of their respective axes. For a more complete discussion of `Irregular` style transformations, see the description of the TickMark object.

Support functions

The **XyPlot** object doesn't define any support functions, but it does inherit all the support functions from its superclass.

Status

See also

- NhlCreate function
- NhlDestroy function
- NhlSetValues function
- NhlGetValues function
- NhlAddData function

* DataComm object
* Title object
* TickMark object

Copyright

Copyright 1987-1999 University Corporation for Atmospheric Research
The use of this Software is governed by a License Agreement.

NCAR Graphics is a registered trademark of the University Corporation for Atmospheric Research.

XyPlot class

The **XyPlot** object draws curves made up of X/Y coordinate pairs with tick marks, titles, and a legend.

Synopsis

```
Header file:             ncarg/hlu/XyPlot.h
Class name:              xyPlotClass
Class pointer:           NhlxyPlotClass
Fortran class function:  NHLFXYPLOTCLASS
Superclass:              DataComm
Composite classes:       PlotManager,LogLinTransformation,IrregularTransformation
```

Data specific class

```
Class name:              xyDataSpecClass
Class pointer:           <Not referenceable>
Fortran class function:  <Not referenceable>
Superclass:              DataSpec
```

Class-defined types

```
Type name:               NhlTAlternatePlace
Definition:
typedef enum _NhlAlternatePlace{
        NhlNONE          = 0,     /* "none"       */
        NhlLEFTAXIS      = 1,     /* "leftAxis"   */
        NhlRIGHTAXIS     = 2,     /* "rightAxis"  */
        NhlTOPAXIS       = 3,     /* "topAxis"    */
        NhlBOTTOMAXIS    = 4      /* "bottomAxis" */
} NhlAlternatePlace;

Type name:               NhlTLineLabelMode
Definition:
typedef enum _NhlLineLabelMode{
        NhlNOLABELS      = 0,     /* "noLabels"   */
        NhlLETTERED      = 1,     /* "lettered"   */
        NhlCUSTOM        = 2      /* "custom"     */
```

} NhlLineLabelMode;

Resources

Local resources

XyPlot resource set

NAME	CLASS	TYPE	DEFAULT	ACCESS
xyCoordData	XyCoordData	NhlTObjIdGenArray	NULL	CSG
xyCoordDataSpec	XyCoordDataSpec	NhlTObjIdGenArray	NULL	G
xyCurveDrawOrder	XyCurveDrawOrder	NhlTDrawOrder	"Draw"	RCSG
xyXStyle	XyXStyle	NhlTTickMarkStyle	NhlLINEAR	RCSG
xyYStyle	XyYStyle	NhlTTickMarkStyle	NhlLINEAR	RCSG
xyXIrregularPoints	XyXIrregularPoints	NhlTFloatGenArray	NULL	RCSG
xyYIrregularPoints	XyYIrregularPoints	NhlTFloatGenArray	NULL	RCSG
xyXIrrTensionF	XyXIrrTensionF	NhlTFloat	2.0	RCSG
xyYIrrTensionF	XyYIrrTensionF	NhlTFloat	2.0	RCSG
xyComputeXMin	XyComputeExtent	NhlTBoolean	<dynamic>	RCSG
xyComputeXMax	XyComputeExtent	NhlTBoolean	<dynamic>	RCSG
xyComputeYMin	XyComputeExtent	NhlTBoolean	<dynamic>	RCSG
xyComputeYMax	XyComputeExtent	NhlTBoolean	<dynamic>	RCSG

Data-specific resources

The following resources are not set directly to the **XyPlot** object. These resources are used to set the graphical attributes that correspond directly with an individual piece of data that is added to the plot. So the attributes of each individual curve or marker are set this way.

When a **DataItem** object is added to the *xyCoordData* resource, either using NhlAddData or using NhlSetValues, an **XyDataSpec** object is automatically created as a child of the **XyPlot** to hold the resources that describe these graphical attributes. The **XyDataSpec** object is given the same name as the **DataItem** object that was added.

To set these resources in a resource file, you would use resource lines similar to this:

```
*xyplotname.dataspecname.xyDashPattern: solidline
```

xyplotname
: the name you give the **XyPlot** when you create it.

dataspecname
: the name of the **XyDataSpec** object, which will be the same as the name of the **DataItem** object that was added to the *xyCoordData* resource.

To set these resources programmatically, you need to retrieve the **XyDataSpec** object id, and call NhlSetValues on that object.

There are two ways to retrieve the ids for the **XyDataSpec** objects that hold the resources that describe the graphical attributes for a particular **DataItem**:

1. If you used NhlAddData to add the **DataItem** to the *xyCoordData* resource, then the return value from that function is the id of the **XyDataSpec** object that describes the graphical attributes for the added **DataItem**.
2. You can retrieve the value of the xyCoordDataSpec resource of the **XyPlot** after all the data have been added to the **XyPlot**. It is an array containing the ids of all the **XyDataSpec** objects that are currently describing how all the **DataItem** objects in the *xyCoordData* resource are to be visualized.

The **XyDataSpec** class also inherits all the **DataSpec** resources.

XyDataSpec resource set

NAME / CLASS	TYPE / DEFAULT	ACCESS
xyDashPattern LineDashPattern	NhlTDashIndex NhlSOLIDLINE	RSG
xyDashPatterns XyDashPatterns	NhlTDashIndexGenArray NULL	RSG
xyMonoDashPattern XyMonoDashPattern	NhlTBoolean False	RSG
xyMarkLineMode XyMarkLineMode	NhlTMarkLineMode NhlLINES	RSG

```
xyMarkLineModes               NhlTMarkLineModeGenArrayRSG
      XyMarkLineModes                 NULL
----------------------------------------------------------------
xyMonoMarkLineMode            NhlTBoolean              RSG
      XyMonoMarkLineMode              False
----------------------------------------------------------------
xyExplicitLegendLabels        NhlTStringGenArray       RSG
      XyExplicitLegendLabels          NULL
----------------------------------------------------------------
xyLineColor                   NhlTColorIndex           RSG
      LineColor                       NhlFOREGROUND
----------------------------------------------------------------
xyLineColors                  NhlTColorIndexGenArray   RSG
      XyLineColors                    NULL
----------------------------------------------------------------
xyMonoLineColor               NhlTBoolean              RSG
      XyMonoLineColor                 False
----------------------------------------------------------------
xyLineDashSegLenF             NhlTFloat                RCSG
      LineDashSegLenF                 <Dynamic>
----------------------------------------------------------------
xyLineLabelFontHeightF        NhlTFloat                RCSG
      FontHeightF                     <Dynamic>
----------------------------------------------------------------
xyLineLabelFontColor          NhlTColorIndex           RSG
      FontColor                       NhlFOREGROUND
----------------------------------------------------------------
xyLineLabelFontColors         NhlTColorIndexGenArray   RSG
      XyLineLabelFontColors           NULL
----------------------------------------------------------------
xyMonoLineLabelFontColor      NhlTBoolean              RSG
      XyMonoLineLabelFontColor        False
----------------------------------------------------------------
xyLabelMode                   NhlTLineLabelMode        RSG
      XyLabelMode                     NhlNOLABELS
----------------------------------------------------------------
xyExplicitLabels              NhlTStringGenArray       RSG
      XyExplicitLabels                NULL
----------------------------------------------------------------
xyLineThicknessF              NhlTFloat                RSG
      LineThicknessF                  1.0
----------------------------------------------------------------
xyLineThicknesses             NhlTFloatGenArray        RSG
      XyLineThicknesses               NULL
----------------------------------------------------------------
xyMonoLineThickness           NhlTBoolean              RSG
      XyMonoLineThickness             False
----------------------------------------------------------------
xyMarker                      NhlTMarkerIndex          RSG
      MarkerIndex                     "default"
----------------------------------------------------------------
xyMarkers                     NhlTMarkerIndexGenArray RSG
      XyMarkers                       NULL
----------------------------------------------------------------
xyMonoMarker                  NhlTBoolean              RSG
      XyMonoMarker                    False
----------------------------------------------------------------
xyMarkerColor                 NhlTColorIndex           RSG
      MarkerColor                     NhlFOREGROUND
----------------------------------------------------------------
xyMarkerColors                NhlTColorIndexGenArray   RSG
```

```
|       XyMarkerColors                    NULL                          |
 ---------------------------------------------------------------------
| xyMonoMarkerColor             NhlTBoolean             RSG             |
|       XyMonoMarkerColor                 False                         |
 ---------------------------------------------------------------------
| xyMarkerSizeF                 NhlTFloat               RSG             |
|       MarkerSizeF                       0.01                          |
 ---------------------------------------------------------------------
| xyMarkerSizes                 NhlTFloatGenArray       RSG             |
|       XyMarkerSizes                     NULL                          |
 ---------------------------------------------------------------------
| xyMonoMarkerSize              NhlTBoolean             RSG             |
|       XyMonoMarkerSize                  False                         |
 ---------------------------------------------------------------------
| xyMarkerThicknessF            NhlTFloat               RSG             |
|       MarkerThicknessF                  1.0                           |
 ---------------------------------------------------------------------
| xyMarkerThicknesses           NhlTFloatGenArray       RSG             |
|       XyMarkerThicknesses               NULL                          |
 ---------------------------------------------------------------------
| xyMonoMarkerThickness         NhlTBoolean             RSG             |
|       XyMonoMarkerThickness             False                         |
 ---------------------------------------------------------------------
| xyLineLabelFont               NhlTFont                RCSG            |
|       Font                              "pwritx"                      |
 ---------------------------------------------------------------------
| xyLineLabelFontAspectF        NhlTFloat               RCSG            |
|       FontAspectF                       1.3125                        |
 ---------------------------------------------------------------------
| xyLineLabelFontThicknessF     NhlTFloat               RCSG            |
|       FontThicknessF                    1.0                           |
 ---------------------------------------------------------------------
| xyLineLabelFontQuality        NhlTFontQuality         RCSG            |
|       FontQuality                       "High"                        |
 ---------------------------------------------------------------------
| xyLineLabelConstantSpacingF   NhlTFloat               RCSG            |
|       TextConstantSpacingF              0.0                           |
 ---------------------------------------------------------------------
| xyLineLabelFuncCode           NhlTCharacter           RCSG            |
|       TextFuncCode                      ':'                           |
+-----------------------------------------------------------------------+
```

Composite resources

Transformation resources

Transformation class resources specify the extent and direction of the data coordinate system. As the superclass of both **LogLinTransformation** and **IrregularTransformation**, you can access all **Transformation** resources. **XyPlot** modifies some of these resources, as follows:

trXMinF
: If this resource is not explicitly set, it is determined based on the value of the *xyComputeXMin* resource and the data extents within the *xyCoordData* resource.

trXMaxF
: If this resource is not explicitly set, it is determined based on the value of the *xyComputeXMax* resource and the data extents within the *xyCoordData* resource.

trYMinF

If this resource is not explicitly set, it is determined based on the value of the *xyComputeYMin* resource and the data extents within the *xyCoordData* resource.

trYMaxF

If this resource is not explicitly set, it is determined based on the value of the *xyComputeYMax* resource and the data extents within the *xyCoordData* resource.

LogLinTransformation resources

If the **XyPlot** *xyXStyle* and *xyYStyle* resources indicate the use of a logarithmic or linear coordinate system, it uses a **LogLinTransformation** object to manage the transformation to and from data space.

All **Transformation** superclass resources are accessible. However, **XyPlot** blocks access to all resources specific to the **LogLinTransformation**.

IrregularTransformation resources

If the **XyPlot** *xyXStyle* and *xyYStyle* resources indicate the use of an irregular coordinate system, it uses an **IrregularTransformation** object to manage the transformation to and from data space.

All **Transformation** superclass resources are accessible. However, **XyPlot** blocks access to all resources specific to the **IrregularTransformation** except for:

- trXTensionF
- trYTensionF

PlotManager resources

If *tfPlotManagerOn* is True when an XyPlot object is created, all **PlotManager** resources are available. However, **XyPlot** does modify some of the **PlotManager** resources as follows:

pmTitleDisplayMode

The default value of *pmTitleDisplayMode* is set to `Conditional`.

pmTickMarkDisplayMode

The default value of *pmTickMarkDisplayMode* is set to `Conditional`.

pmLegendDisplayMode

The default value of *pmLegendDisplayMode* is set to `Never`.

pmLabelBarDisplayMode

The *pmLabelBarDisplayMode* resource is hard-coded to its default value of `NoCreate` in the **XyPlot**.

The resources for the **PlotManager's** *Composite* parts are available as follows:

Title resources

All the Title resources are available as they are modified by **PlotManager**.

TickMark resources

All the TickMark resources are available as modified by **PlotManager**.

LabelBar resources

All LabelBar resources are disabled by **XyPlot**.

Legend resources

The Legend resources are available as modified by **PlotManager** with the exception of some resources that have been disabled since they are set automatically based on the current values of relevant **XyPlot** and **XyDataSpec** resources. These resources include:

- lgItemCount
- lgLabelStrings
- lgMonoItemType
- lgItemType
- lgItemTypes
- lgMonoDashIndex
- lgDashIndex
- lgDashIndexes
- lgMonoLineColor
- lgLineColor
- lgLineColors
- lgMonoLineThickness
- lgLineThicknessF
- lgLineThicknesses
- lgMonoLineDashSegLen
- lgLineDashSegLenF
- lgLineDashSegLens
- lgMonoMarkerIndex
- lgMarkerIndex
- lgMarkerIndexes
- lgMonoMarkerColor
- lgMarkerColor
- lgMarkerColors
- lgMonoMarkerThickness
- lgMarkerThicknessF
- lgMarkerThicknesses
- lgMonoMarkerSize
- lgMarkerSizeF
- lgMarkerSizes
- lgMonoLineLabelFontHeight
- lgLineLabelFontHeightF
- lgLineLabelFontHeights
- lgMonoLineLabelFontColor
- lgLineLabelFontColor
- lgLineLabelFontColors
- lgLineLabelStrings
- lgLineLabelFontAspectF
- lgLineLabelFontThicknessF
- lgLineLabelFontQuality
- lgLineLabelConstantSpacingF
- lgLineLabelFuncCode

Superclass resources

All of the **DataComm** resources can be set for the **XyPlot** object.

Description

XyPlot is a composite object. It combines the functionality of the **PlotManager** object, **LogLinTransformation**, and **IrregularTransformation** to create a fully functional **XyPlot** object.

Since the **XyPlot** is a composite object, the resources for the *Composite* classes are also available to the **XyPlot** object through the standard resource-setting mechanisms.

XyPlot is derived from the **DataComm** class, which is derived from the **Transform** class. This means that the functions NhlDataToNDC and NhlNDCToData can be used to obtain transformations of points to and from data coordinates and NDC coordinates. See the Transform class for more information on these functions.

In general, **XyPlot** accepts one or more vectors of X and Y coordinates and displays them by drawing a line through the {X,Y} coordinate pairs, or drawing markers at each point within the coordinate pairs, or both.

XyPlot provides the user with the capability of configuring the line thickness, dash pattern indexes, colors of the lines, line labels, and line label colors for curves. **XyPlot** allows the user to configure Marker indexes, colors, sizes, and thicknesses for displaying markers. **XyPlot** also provides all the same transformation styles that are provided by the **TickMark** object: LOG, LINEAR, and IRREGULAR.

XyPlot accepts **DataItem** objects rather than arrays of values. The *xyCoordData* resource is used to associate **DataItem** objects with the **XyPlot** object. The *xyCoordData* resource is used to specify a list of **DataItem** objects, and the **AddData** function appends **DataItems** to the list.

There are basically two steps to visualizing a set of data in the XyPlot object. First, the programmer has to fully describe the data to the HLU environment. This is done by creating a DataItem object that describes the data's shape. Second, the programmer has to associate the **DataItem** with the *xyCoordData* resource of the **XyPlot**.

You can associate data with **XyPlot** by creating a **DataItem** to describe the data. Then use the NhlAddData function to add that **DataItem** to the list of data referenced by the *xyCoordData* resource. If the **XyPlot** object understands data in the format provided (i.e. the **DataItem** class used), the **NhlAddData** function will return with a value of **NhlNOERROR**.

Additionally, the programmer can just set the *xyCoordData* resource directly using the NhlSetValues function. The *xyCoordData* resource is set with an array of object ids. The difference between setting the resource directly and calling the **NhlAddData** function is that the **NhlAddData** function appends the given data to the current list of data, while setting the *xyCoordData* resource using **NhlSetValues** replaces the current value of that resource.

If programmers want to control the **XyDataSpec** resources programmatically, they will need to add the

DataItems to the **XyPlot**'s *xyCoordData* resource first. Then they can retrieve the value of the *xyCoordDataSpec* resource. The *xyCoordDataSpec* resource is an array of **XyDataSpec** ids. The elements of this array correspond to the **DataItem** ids in the *xyCoordData* array resource. To configure exactly how each coordinate pair array of one of the **DataItem**s is going to be displayed, the programmer can set the **XyDataSpec** resources to the corresponding **XyDataSpec** id.

The **XyDataSpec** object is created on behalf of the programmer for each **DataItem** object that is added to the *xyCoordData* resource. The **XyDataSpec** object is created with the same name as the **DataItem** and as a child of the **XyPlot**. Therefore, the data-specific resources (i.e. the resources of the **XyDataSpec** object) can be set in the resource file by simply specifying the name of the **DataItem** as a child of the **XyPlot** with the **XyDataSpec** resources in the resource file.

The **XyDataSpec** resources allow the user to set dash patterns, line colors, line thicknesses, line labels, and line label font colors and heights for each element of the *xyCoordData* resource. The user can also set the other standard text attributes for line labels. However, these attributes are only available as scalar resources, and therefore the **XyDataSpec** object referenced by the first element of the *xyCoordDataSpec* array supplies the values of these resources for all data elements.

The *xyLabelMode* and *xyExplicitLabels* resources control how each curve is labeled. The curve line labels can be any string. However, line labels longer than six or seven characters can cause problems, and spaces are not drawn.

XyPlot also allows markers to be plotted. The *xyMarkLineMode*, *xyMarkLineModes*, and *xyMonoMarkLineMode* resources are used to indicate if markers should be plotted in place of curves, or in addition to them. There are also a number of other resources in the **XyDataSpec** object to configure exactly how the markers should be displayed.

The next step in configuring an **XyPlot** is to select and specify the data transformation of the **XyPlot** object. As is the case with the **TickMark** object, there are currently three styles of data transformations, `Log`, `Linear`, and `Irregular`. There are two style resources for setting the X-Axis and Y-Axis data transformations. This differs from the **TickMark** object, where each side of each axis has its own resource for specifying style. **XyPlot** forces both sides of an axis to have the same style, so there is no need for two separate style resources that set the style for each side of an axis. This has been done to eliminate the possibility that the tick marks don't accurately represent the data transformation. If different styles were allowed for both sides of the X Axis, then the possibility arises that one of the transformations is wrong. The two resources for selecting the transformation style are *xyXStyle* and *xyYStyle*. By default, these are both set to `Linear`.

Once the style is selected, the resources *trXMinF*, *trXMaxF*, *trYMinF*, and *trYMaxF* are used to set the data extents. If these resources are not set, then the **XyPlot** object examines the data and determines for itself what the data extents are. The actual data extents could change if additional data are added to the *xyCoordData* resource or if one of the current **DataItem**s is modified. The *tr* data-extent resources will be automatically recomputed if the *xyComputeXMin*, *xyComputeXMax*, *xyComputeYMin*, and *xyComputeYMax* resources allow it.

`Irregular` style transformations require some additional information. The resources *xyXIrregularPoints* and *xyYIrregularPoints* must be used when `Irregular` style has been selected. These resources apply to both sides of their respective axes. For a more complete discussion of `Irregular` style transformations, see the description of the TickMark object.

Support functions

The **XyPlot** object doesn't define any support functions, but it does inherit all the support functions from its superclass.

Status

See also

- NhlCreate function
- NhlDestroy function
- NhlSetValues function
- NhlGetValues function
- NhlAddData function
- DataComm object
- Title object
- TickMark object

Copyright

Copyright 1987-1999 University Corporation for Atmospheric Research
The use of this Software is governed by a License Agreement.

NCAR Graphics is a registered trademark of the University Corporation for Atmospheric Research.

NhlPError

NhlPError, NHLPERROR, NhlErrGetId, NhlErrNumMsgs, NhlErrGetMsg, NhlErrCleanMsgs, NhlErrAddTable, NhlErrSPrintMsg, NhlErrFPrintMsg

The Fortran names of these functions are **NhlFPError**, **NhlFErrGetId**, **NhlFErrNumMsgs**, **NhlFErrGetMsg**, **NhlFErrCleanMsgs**, **NhlFErrSprintMsg**, and **NhlFErrFPrintMsg**. There are no

Fortran bindings for **NHLPERROR** and **NhlErrAddTable**.

Error handling functions for the HLU library.

C synopsis

```
#include <ncarg/hlu/hlu.h>

const char *NhlPError(
                NhlErrorTypes   severity,
                int             errnum,
                char            *format,
                ...
                )

void NHLPERROR((
                NhlErrorTypes   severity,
                int             errnum,
                char            *format,
                ...
                ))

int NhlErrGetId()

int NhlErrNumMsgs()

NhlErrorTypes NhlErrGetMsg(
                int             msgnum,
                const NhlErrMsg **msgptr
                )

NhlErrorTypes NhlErrClearMsgs()

NhlErrorTypes NhlErrAddTable(
                int             start,
                int             tlen,
                const char      **etable
                )

char *NhlErrSPrintMsg(
                char            *buffer,
                const NhlErrMsg *msg
                )

const char *NhlErrFPrintMsg(
                FILE            *fp,
                const NhlErrMsg *msg
                )
```

Fortran synopsis

```
      subroutine NhlFPError(severity, errnum, message)
```

```
      character*(*) severity, message
      integer errnum

      subroutine NhlFErrGetId(id_ret)
      integer id_ret

      subroutine NhlFErrNumMsgs(nummsg)
      integer nummsg

      subroutine NhlFErrGetMsg(msgnum, sev, emsg, errorno, sysmsg
     +                         line, fname, ierr)
      integer msgnum, sev, errorno, line, ierr
      character*(*) emsg, sysmsg, fname

      subroutine NhlFErrClearMsgs(ierr)
      integer ierr

      subroutine NhlFErrSprintMsg(buffer, msgnum)
      character*(*) buffer
      integer imsg

      subroutine NhlFErrFprintMsg(iunit, msgnum)
      integer iunit, imsg
```

Arguments

severity (input)
: Specifies the severity level of the error message being reported.

errnum (input)
: Specifies a unique error number that references an error message from an installed error table.

format (input, C only)
: Message to report with argument replacement like format string for printf.

... (input, C only)
: Arguments for format string aka printf.

msgnum (input)
: Identifies the index of the error message to retrieve.

msgptr (output, C only)
: Pointer to an NhlErrMsg pointer that fully describes an error message that was reported. This arg is used to retrieve a pointer to an NhlErrMsg.

start (input, C only)
: Index to assign to the first error message in the table being installed.

tlen (input, C only)
: Length of the error table being installed.

etable (input, C only)
: Pointer to an error table to install.

buffer (output)
: Character buffer to place the formatted message into.

msg (input, C only)
: Pointer to an NhlErrMsg structure that fully describes an error message.

fp (input, C only)

File pointer to print formatted error message to.

message (input, Fortran only)
: Message to report.

id_ret (output, Fortran only)
: Id for *Error object*.

nummsg (output, Fortran only)
: Number of error messages currently buffered.

sev (output, Fortran only)
: Indicates the severity level of the message being retrieved.

emsg (output, Fortran only)
: Actual text of the error message being retrieved. This corresponds to the *format* argument (after argument replacement) if the error was reported from **C** and the *message* argument if it was reported from **Fortran**.

errorno (output, Fortran only)
: Error number of the error message being retrieved.

sysmsg (output, Fortran only)
: Text from the error table for the message being retrieved.

line (output, Fortran only)
: Line number the error message was reported from, or 0.

fname (output, Fortran only)
: File name the error message was reported from, or NULL.

ierr (output, Fortran only)
: Error code.

iunit (input, Fortran only)
: Unit number to print formatted error message to.

Types

```
Type name:               NhlTErrorTypes
Definition:
typedef enum _NhlErrType{
        NhlFATAL        = -4,   /* "FATAL"      */
        NhlWARNING      = -3,   /* "WARNING"    */
        NhlINFO         = -2,   /* "INFO"       */
        NhlNOERROR      = -1    /* "NOERROR"    */
} NhlErrorTypes;

Type name:               <none>
typedef struct _NhlErrMsg{
                NhlErrorTypes   severity;
                char            *msg;
                int             errorno;
                const char      *sysmsg;
                int             line;
                char            *fname;
} NhlErrMsg, *NhlErrMsgList;
```

severity
: Specifies level of this error message.

msg
: Actual Text of this error message.

errorno
: Error number of this error message.

sysmsg
: Actual Text in the error table indexed by *errorno*.

line
: Line number the error message was reported from, or 0.

fname
: File name the error message was reported from, or NULL.

Description

The error handling module of the HLU library has two purposes. First, it allows the programmer to report errors using the same error mechanism as the HLU library if they want to. Second, it allows the programmer to configure the way those errors are reported.

The **NhlPError** function is used to report errors to the error module. The *fmt* and *arg* parameters are parsed in the same way as **printf** parses its arguments. The **NHLPERROR** macro is used in the same way as the **NhlPError** function except that it prepends File and Line information to the error message. Notice the double parentheses, this was necessary to get the macro to take variable number arguments.

Another function that is useful to programmers--if they plan to use the **NhlPError** function to report errors--is **NhlErrAddTable**. This function allows the programmer to add an additional error table in the same format as the system error table. The values 0 - 1000 are reserved for the system error table. The error module will not allow overlapping error tables to be installed.

The error module was implemented as another object in the Nhl library. This is important because all configuration of the Error module is done by setting resources in this object. The **NhlOpen** function creates the Error object so the resources are not available to the programmer at create time, except by using a resource file. However the programmer does have **NhlSetValues** and **NhlGetValues** access. The **NhlErrGetId** function is provided so the programmer can get the Id that is necessary to set or get resources in an object.

The rest of the functions are provided to allow the programmer to do as much of the error handling as they want. They are really only useful if the error object's *NhlNbufferErrors* resource is set to True.

The **NhlErrNumMsgs** function is provided so the programmer can query the error object to find out how many error messages it currently has buffered. The **NhlErrGetMsg** function is used to retrieve a pointer to the error message indexed by msgnum. The error message is saved internally in a *NhlErrMsg* structure. The index is from 1 to *NhlErrNumMsgs*.

The **NhlErrClearMsgs** function is used to clear the Error object's internal message buffer. The **NhlErrFPrintMsg** and **NhlErrSPrintMsg** functions were provided so the programmer can print messages in the *NhlErrMsg* structure format in the same way the error object prints them out internally.

The error object is destroyed when the **NhlClose** function is called. The programmer can use the **NhlPError** function when the error object doesn't exist (before **NhlOpen** or after **NhlClose**). However, all messages are printed to stderr because no configuration is possible.

Return values

The **NhlPError** function returns a `const char*` that points to the formatted error message.

The **NHLPERROR** macro does not return a value.

The **NhlErrGetId** function returns an int that is a unique id for the Error object.

The **NhlErrNumMsgs** function returns an int that is the number of error messages that have been buffered in the Error object.

The **NhlErrGetMsg**, **NhlErrClearMsgs**, and **NhlErrAddTable** functions return values of type NhlErrorTypes. These error eunctions return **NOERROR** if they were able to complete the function, and **FATAL** otherwise.

The **NhlErrSPrintMsg** function returns a pointer to the *buffer* provided, or *NULL* on error.

The **NhlErrFPrintMsg** function returns a pointer to a `const char` buffer that contains the formatted string that was printed to the *fp* argument, or *NULL* on error.

See also

- Error object

Copyright

Copyright 1987-1999 University Corporation for Atmospheric Research
The use of this Software is governed by a License Agreement.

NCAR Graphics is a registered trademark of the University Corporation for Atmospheric Research.

NhlAddAnnotation

The Fortran name of this function is **NhlFAddAnnotation**.

This function adds an arbitrary **View** object to a plot object as an external annotation, and returns an

AnnoManager object to manage it.

C Synopsis

```
#include <ncarg/hlu/hlu.h>
#include <hfile for Plot Object>

int NhlAddAnnotation(
                int     plot_id,
                int     view_id
                )
```

Fortran Synopsis

```
      subroutine NhlFAddAnnotation(plot_id, view_id, anno_manager_id)
      integer plot_id, view_id, anno_manager_id
```

Arguments

plot_id (input)
Identifier of the plot object to which the **View** object is to be added as an annotation.

view_id (input)
Identifier of the **View** object to be added as an annotation of the plot object.

anno_manager_id (output, Fortran only)
If successful, identifier of new **AnnoManager** object; else error code.

Types

```
Type name:                NhlTErrorTypes
Definition:
typedef enum _NhlErrType{
        NhlFATAL        = -4,   /* "FATAL"      */
        NhlWARNING      = -3,   /* "WARNING"    */
        NhlINFO         = -2,   /* "INFO"       */
        NhlNOERROR      = -1    /* "NOERROR"    */
} NhlErrorTypes;
```

Description

Use this function to add an arbitrary view as an external annotation of a plot object. The view must have the same **Workstation** parent as the plot object to which it is added, and it cannot already be a plot member. Assuming the object identifiers are valid, the the function will return the identifier of an **AnnoManager** object whose resources you can manipulate in order to set the position and usually the size of the view relative to the viewport or the data coordinate space of the base plot.

The view id is appended to an array of user-created annotations accessible through the **PlotManager** resource *pmAnnoViews*. The id of its controlling **AnnoManager** object is in like manner appended to the resource *pmAnnoManagers*.

You can remove the annotation from the plot object using the **NhlRemoveAnnotation** function. You can retrieve the id of the view's controlling **AnnoManager** at any time by getting the value of the **View** resource *vpAnnoManagerId*. Note that you are responsible for destroying the view when you are through using it.

Return Values

If successful the **NhlAddAnnotation** C function returns the identifier of the **AnnoManager** object created to manage the view as an external annotation, and the **NhlFAddAnnotation** Fortran subroutine returns this value in *anno_manager_id*. A return value less than 0 indicates failure; you may cast such a value to type **NhlErrorTypes** in order to determine the error level. The following table indicates what the various return Error Values mean:

```
Value   |                        Meaning
--------+-----------------------------------------------
NOERROR |       not possible
--------+-----------------------------------------------
INFO    |       not possible
--------+-----------------------------------------------
WARNING |       not possible
--------+-----------------------------------------------
FATAL   |       An invalid object id passed in, or internal error.
```

See Also

- **AnnoManager** object
- **View** object
- **PlotManager** object
- **NhlRemoveAnnotation** function

Copyright

Copyright 1987-1999 University Corporation for Atmospheric Research
The use of this Software is governed by a License Agreement.

NCAR Graphics is a registered trademark of the University Corporation for Atmospheric Research.

NhlAddData

The Fortran name of this function is **NhlFAddData**.

This function is used to associate a **DataItem** object with a data resource in a **DataComm** object.

C synopsis

```
#include <ncarg/hlu/DataComm.h>

int NhlAddData(
                int             dcommid,
                NhlString       res_name,
                int             ditemid
                )
```

Fortran synopsis

```
      subroutine NhlFAddData(dcommid, res_name, ditemid, id)
      integer dcommid, ditemid, id
      character*(*) res_name
```

Arguments

dcommid (input)
: Identifies the **DataComm** object with the data resource.

res_name (input)
: Identifies the data resource to set in the **DataComm** object.

ditemid (input)
: Identifies the **DataItem** object id to associate with the *res_name* resource in the *dcommid* **DataComm** object.

id (output, Fortran only)
: The id of the **DataSpec** object that was created to describe the **DataItem** added.

Description

This function is used to associate the given *ditemid* **DataItem** object with the given *res_name* data resource in the given *dcommid* **DataComm** object.

If the given *ditemid* is a valid data type for the given *res_name*, then the *ditemid* will be "added" to that data resource. Different **DataComm** subclasses will use their data resources in different ways. For example, the **XyPlot** object has a data resource *xyCoordData* which is used as a list of **DataItems**. Each **DataItem** in the *xyCoordData* list may actually specify more than one curve in the plot.

Additionally, **DataComm** objects usually have a companion resource to the *res_name* resource that is used to retrieve **DataSpec** objects. The **DataSpec** objects are used to describe *Data Specific* plot attributes. For example, the **XyPlot** objects use the **XyDataSpec** object to describe actual line colors for each coordinate pair array described in the *xyCoordData* resource. The **DataComm** object creates a **DataSpec** object for each individual **DataItem** that is added. The **DataSpec** object is created as a child of the **DataComm** object, and given the same name as the **DataItem** object it is describing. The **XyPlot** object has the *xyCoordDataSpec* resource to allow you to retrieve the array of **XyDataSpec** objects that describe the **DataItem**s in the *xyCoordData* resource. The elements of the *companion* resource correspond to the **DataItem** ids in the *res_name* resource. In the **XyPlot** case, this means that each **DataItem** in the *xyCoordData* resource array is described by the corresponding **XyDataSpec** object in the *xyCoordDataSpec* resource array.

Provided as a shortcut to doing an **NhlGetValues** of the *companion* resource, the **NhlAddData** function returns the id of the **DataSpec** object that was created to describe the **DataItem** added.

Return values

The **NhlAddData** C function returns a value of type **int**, and the **NhlFAddData** returns this value in *id*. The following table indicates what the return values mean:

Value	Meaning
> 0	The **DataItem** was added to the *res_name* data resource without any problems. The integer is the id of the **DataSpec** object being used to visualize that data.
0	The **DataItem** was added, but the function was unable to retrieve the value of the **DataSpec** object id.
< 0 NhlFATAL	Unable to add the given **DataItem** to the given *res_name* data resource. This probably indicates that type of **DataItem** is not understood by the given **DataComm** object.

See also

- NhlRemoveData
- DataComm Object
- DataItem Object

Copyright

Copyright 1987-1999 University Corporation for Atmospheric Research
The use of this Software is governed by a License Agreement.

NCAR Graphics is a registered trademark of the University Corporation for Atmospheric Research.

NhlAddOverlay

The Fortran name of this function is **NhlFAddOverlay**.

This function adds an overlay plot member to a base plot.

C Synopsis

```
#include <ncarg/hlu/hlu.h>
#include <hfile for Base Plot Object>

NhlErrorTypes NhlAddOverlay(
                int             base_id,
                int             transform_id,
                int             after_id
                )
```

Fortran Synopsis

```
      subroutine NhlFAddOverlay(base_id, transform_id, after_id, ierr)
      integer base_id, transform_id, after_id, ierr
```

Arguments

base_id (input)

Identifies the plot object to be used as the base plot.

transform_id (input)

Identifies the **Transform** object to be overlaid on the base plot.

after_id (input)

If the base plot already contains one or more overlays, identifies the existing overlay that the new overlay is to follow in the overlay sequence. A value less than 0 specifies that the new overlay is to be added to the end of the sequence. If the value is the id of the base plot itself, the new overlay is placed just behind the base plot in the sequence.

ierr (output, Fortran only)

Error code.

Types

```
Type name:                NhlTErrorTypes
Definition:
typedef enum _NhlErrType{
        NhlFATAL         = -4,    /* "FATAL"       */
        NhlWARNING       = -3,    /* "WARNING"     */
        NhlINFO          = -2,    /* "INFO"        */
        NhlNOERROR       = -1     /* "NOERROR"     */
} NhlErrorTypes;
```

Description

Use this function to add a transform as an overlay of a base plot. The transform may be either a plot object (with or without plot members of its own) or a simple transform. The transform must have the same **Workstation** parent as the base plot, and it cannot already be a plot member. The transform becomes a plot member of the base plot. The base plot provides the coordinate transformation and data boundaries that together determine where the data coordinate space belonging to the overlay will appear and how much of it will be visible. In addition, if the transform is a plot object the base plot assumes management responsibilities for the plot object's plot members if it has any.

If the base plot already has overlays, you can place new overlay anywhere in the overlay sequence using the *after_id* parameter. The overlay sequence determines the basic drawing order for the overlays.

Note that the **MapPlot** cannot become an overlay, although it can serve as a base plot. Attempts to overlay an object of the **MapPlot** class results in an error.

Return Values

The **NhlAddOverlay** C function returns a value of type **NhlErrorTypes**, and the **NhlFAddOverlay** Fortran subroutine returns the error value in *ierr*. The following table indicates what the various return Error Values mean:

Value	Meaning
NOERROR	Transform object successfully added to the base plot
INFO	minor recoverable error
WARNING	recoverable error - transform object added to the base plot but something was was specified incorrectly.
FATAL	The transform could not be added to the base plot.

See Also

- **PlotManager** object
- **NhlRemoveOverlay** function

Copyright

Copyright 1987-1999 University Corporation for Atmospheric Research
The use of this Software is governed by a License Agreement.

NCAR Graphics is a registered trademark of the University Corporation for Atmospheric Research.

NhlAppGetDefaultParentId

The Fortran name of this function is **NhlFAppGetDefaultParentId**.

This function retrieves the Id of the current *default parent* **App** object.

C Synopsis

```
#include <ncarg/hlu/App.h>

int NhlAppGetDefaultParentId(
                void
                )
```

Fortran Synopsis

```
subroutine NhlFAppGetDefaultParentId(id)
integer id
```

Arguments

id (output, Fortran only)
  On output contains the id of the current default **App** object.

Description

This function returns the Id of the **App** object that will currently be used as the parent of any objects that are created with the constant `NhlDEFAULT_APP` or `0` as the parentid argument of the NhlCreate function.

Return values

If greater than 0, the value is the id of the user created **App** object that is currently being used as the *default parent*. Otherwise, it will return 0 to indicate that the internal HLU **App** object is active.

See also

- **App object**

Copyright

Copyright 1987-1999 University Corporation for Atmospheric Research
The use of this Software is governed by a License Agreement.

NCAR Graphics is a registered trademark of the University Corporation for Atmospheric Research.

NhlChangeWorkstation

The Fortran name of this function is **NhlFChangeWorkstation**.

This function is used to change the Workstation parent of any graphical HLU object. It is provided as an access function to objects that are Base class.

C synopsis

```
#include <ncarg/hlu/hlu.h>

NhlErrorTypes NhlChangeWorkstation(
                int             plot_id,
                int             work_id
                )
```

Fortran synopsis

```
      subroutine NhlFChangeWorkstation(plot_id, work_id, ierr)
      integer plot_id, work_id, ierr
```

Arguments

plot_id (input)
    Id that identifies the graphical object to move to another Workstation class parent.
work_id (input)
    Id that identifies the workstation object to move the graphical object to.
ierr (output, Fortran only)
    Error code.

Types

```
Type name:                NhlTErrorTypes
Definition:
typedef enum _NhlErrType{
        NhlFATAL        = -4,    /* "FATAL"       */
        NhlWARNING      = -3,    /* "WARNING"     */
        NhlINFO         = -2,    /* "INFO"        */
        NhlNOERROR      = -1     /* "NOERROR"     */
} NhlErrorTypes;
```

Description

This function is used to change the output device that the graphical object will be drawn to. It effectively Reparents the given *plot_id* so it behaves as if the given *work_id* was passed to the **Create** function as the *parent_id*. Although, since resource values are determined at **NhlCreate** time, the resource values of the *plot_id* object are not modified.

Return values

The **NhlChangeWorkstation** C function returns a value of type **NhlErrorTypes**, and the **NhlFChangeWorkstation** Fortran subroutine returns the error in *ierr*. The following table indicates what the return error values mean:

Value	Meaning
NOERROR	Object was reparented so it will draw to the new Workstation object with no errors.
INFO	No errors of this type.
WARNING	No errors of this type.
FATAL	Object given was not reparented. Usually indicates that the *plot_id* was invalid, or the *work_id* was invalid.

See also

- NhlCreate function
- NhlDraw
- Workstation class

Copyright

Copyright 1987-1999 University Corporation for Atmospheric Research
The use of this Software is governed by a License Agreement.

NCAR Graphics is a registered trademark of the University Corporation for Atmospheric Research.

NhlClassName

The Fortran name of this function is **NhlFClassName**.

Returns the name of the class of an object.

C synopsis

```
#include <ncarg/hlu/hlu.h>

const char *NhlClassName(
                int     id
                )
```

Fortran synopsis

```
      subroutine NhlFClassName(id, name_ret, ierr)
      integer id, ierr
      character*(*) name_ret
```

Arguments

id (input)
    Specifies the object identifier.

name_ret (output, Fortran only)
    On output, the class name of the object specified by the *id* argument.

ierr (output, Fortran only)
    Error code.

Types

```
Type name:              NhlTErrorTypes
Definition:
typedef enum _NhlErrType{
        NhlFATAL        = -4,   /* "FATAL"      */
        NhlWARNING      = -3,   /* "WARNING"    */
        NhlINFO         = -2,   /* "INFO"       */
        NhlNOERROR      = -1    /* "NOERROR"    */
} NhlErrorTypes;
```

Description

The C function returns the class name of the object specified by the *id* argument, and the Fortran subroutine returns the class name in the variable *name_ret*. This is the name of the class specified during the **NhlCreate** call that created the object.

Return values

The **NhlClassName** C function returns a valid `const char*` or **NULL** on error.

The **NhlFClassName** Fortran subroutine returns the error in *ierr*:

```
Value   |                        Meaning
--------+-----------------------------------------------------
NOERROR |       Class name successfully copied to name_ret
--------+-----------------------------------------------------
INFO    |       not possible
--------+-----------------------------------------------------
WARNING |       name_ret wasn't big enough
--------+-----------------------------------------------------
FATAL   |       An invalid object id passed in.
```

See also

- NhlCreate

Copyright

Copyright 1987-1999 University Corporation for Atmospheric Research
The use of this Software is governed by a License Agreement.

NCAR Graphics is a registered trademark of the University Corporation for Atmospheric Research.

NhlClearWorkstation

The Fortran name of this function is **NhlFClearWorkstation**.

This function clears the drawing area of an instance of a **Workstation** object.

C Synopsis

```
#include <ncarg/hlu/hlu.h>
#include <hfile for Workstation class object>
```

```
NhlErrorTypes NhlClearWorkstation(
                int     workid
                )
```

Fortran Synopsis

```
subroutine NhlFClearWorkstation(workid, ierr)
integer workid, ierr
```

Arguments

workid (input)
Integer identifier of an instance of a **Workstation** class object
ierr (output, Fortran only)
Error code.

Types

```
Type name:                NhlTErrorTypes
Definition:
typedef enum _NhlErrType{
        NhlFATAL        = -4,    /* "FATAL"       */
        NhlWARNING      = -3,    /* "WARNING"     */
        NhlINFO         = -2,    /* "INFO"        */
        NhlNOERROR      = -1     /* "NOERROR"     */
} NhlErrorTypes;
```

Description

This function clears the **Workstation** drawing area. For **XWorkstation** objects **NhlClearWorkstation** clears the window associated with the **XWorkstation** object. If the **XWorkstation** resource *wkPause* is set True, the **NhlClearWorkstation** blocks until the user clicks a mouse button in the window. For **NcgmWorkstation** objects this function causes an `End Picture` command to be written to the output metafile and places the metafile generator into a state such that it is prepared either to begin a new metafile picture if new draw commands are received or to end the metafile if the workstation object is destroyed.

You can replace the usual calling sequence of **NhlUpdateWorkstation** followed by **NhlClearWorkstation** with a single call to **NhlFrame**.

Return Values

The **NhlClearWorkstation** C function returns a value of type **NhlErrorTypes**, and the **NhlFClearWorkstation** Fortran subroutine returns the error in *ierr*. The following table indicates what the various return Error Values mean:

Value	Meaning
NOERROR	function successful
INFO	minor recoverable error
WARNING	recoverable error
FATAL	function failed

See Also

- **NhlDraw**
- **NhlUpdateWorkstation**
- **NhlFrame**
- **Workstation object**
- **XWorkstation object**
- **NcgmWorkstation object**

Copyright

Copyright 1987-1999 University Corporation for Atmospheric Research
The use of this Software is governed by a License Agreement.

NCAR Graphics is a registered trademark of the University Corporation for Atmospheric Research.

NhlClose

The Fortran name of this function is **NhlFClose**.

HLU library clean-up function.

C synopsis

```
#include <ncarg/hlu/hlu.h>

void NhlClose()
```

Fortran synopsis

```
      subroutine NhlFClose
```

Description

This function is used to tell the HLU library that the programmer is done using it. It frees up any memory that it can. The library is unusable after this call, and it is not currently possible to re-open it.

See also

- App object
- Open Function

Copyright

Copyright 1987-1999 University Corporation for Atmospheric Research
The use of this Software is governed by a License Agreement.

NCAR Graphics is a registered trademark of the University Corporation for Atmospheric Research.

NhlCreate

The Fortran name of this function is **NhlFCreate**.

This function is used to create an instance of any HLU object. It is provided as an access function to objects that are Base class.

C synopsis

```
#include <ncarg/hlu/hlu.h>
#include <hfile for class being created>

NhlErrorTypes NhlCreate(
                int             *ret_id,
                NhlString       name,
                NhlClass        lc,
                int             parent_id,
                int             rlist_id
                )
```

Fortran synopsis

```
      external function_for_class_being_created
      subroutine NhlFCreate(ret_id, name, lc, parent_id, rlist_id, ierr)
      integer ret_id, parent_id, rlist_id, ierr
      character*(*) name
      external lc
```

Arguments

ret_id (output)
: Returns the object id that is used to identify the created object. HLU object ids are always positive numbers. If this function is unable to create the object, this value will be set to the return value of the function (`NhlFATAL`).

name (input)
: Sets the name of the object being created.

lc (input)
: Identifies the class of object to create, i.e. the *type* of object to create.

parent_id (input)
: Id of the object that should be the parent of the object being created. The constant NhlDEFAULT_APP (0) can be used to indicate the current *default parent* **App** object.

rlist_id (input)
: Id of a ResList that specifies which resources to initialize, and what values to initialize them to. A value of 0 can be used to indicate a *NULL* ResList. The ResList should be created with a *list_type* of `NhlSETRL` (or 'setrl' in Fortran).

ierr (output, Fortran only)
: Error code.

Types

```
Type name:                NhlTErrorTypes
Definition:
typedef enum _NhlErrType{
        NhlFATAL        = -4,   /* "FATAL"      */
```

```
        NhlWARNING       = -3,    /* "WARNING"     */
        NhlINFO          = -2,    /* "INFO"        */
        NhlNOERROR       = -1     /* "NOERROR"     */
} NhlErrorTypes;
```

Description

This function is used to create any object in the HLU library. The **NhlCreate** function uses the ResList specified by the *rlist_id* parameter to initialize resources that you want to initialize differently than the default. The *rlist_id* is allocated using the NhlRLCreate function. You set the *names* and *values* to initialize using the NhlRLSet, NhlRLSetArray and NhlRLSetMDArray functions.

The *parent_id* argument should be set as follows:

1. If you are creating an **App** object, this argument is ignored.
2. If you are creating a **Workstation** or **DataItem** object, this argument should specify an **App** object.
3. If you are creating a **View** object, this argument should specify a **Workstation** object.

The **NhlCreate** function uses the Resource Database to determine the default values to initialize resources to. The values set in the *rlist_id* ResList are used to override these default values. If defaults cannot be determined from the ResList, or from the resource database, then internal hard-coded defaults will be used.

The **SetValues** functions can be used to change the resource values of an object after it has been created.

Return values

The **NhlCreate** C function returns a value of type **NhlErrorTypes**, and the **NhlFCreate** Fortran subroutine returns the error in *ierr*. The following table indicates what the return error values mean:

Value	Meaning
NOERROR	Object was created with no problems.
INFO	Error was recoverable - usually indicates that the programmer passed an invalid resource name or value in the *args* argument that was ignored.
WARNING	Error was recoverable - usually indicates that there may be some parts of the object that are not configured properly, and that a FATAL error may occur at a later time.
FATAL	The function was unable to create the object.

See also

- Create function overview
- SetValues function overview
- GetValues function overview
- Resource database overview

Copyright

Copyright 1987-1999 University Corporation for Atmospheric Research
The use of this Software is governed by a License Agreement.

NCAR Graphics is a registered trademark of the University Corporation for Atmospheric Research.

NhlDataPolygon

The Fortran name of this function is **NhlFDataPolygon**.

Given arrays containing the X and Y coordinates of points in data coordinate space, this function outputs an immediate-mode polygon connecting each of the points.

C Synopsis

```
#include <ncarg/hlu/hlu.h>
#include <hfile for Transform class object>

NhlErrorTypes NhlDataPolygon(
                int     pid,
                int     gsid,
                float   *x,
                float   *y,
                int     n,
                )
```

Fortran Synopsis

```
      subroutine NhlFDataPolygon(pid, gsid, x, y, n, ierr)
      integer pid, gsid, n, ierr
      real x(n), y(n)
```

Arguments

pid (input)
  Id of a **Transform** or **Workstation** object.
gsid (input)
  Id of a **GraphicStyle** object.
x (input)
  Array of X-Axis coordinate values in data coordinate space.
y (input)
  Array of Y-Axis coordinate values in data coordinate space.
n (input)
  Number of elements in the coordinate arrays *x* and *y*.
ierr (output, Fortran only)
  Error code.

Types

```
Type name:               NhlTErrorTypes
Definition:
typedef enum _NhlErrType{
        NhlFATAL         = -4,    /* "FATAL"       */
        NhlWARNING       = -3,    /* "WARNING"     */
        NhlINFO          = -2,    /* "INFO"        */
        NhlNOERROR       = -1     /* "NOERROR"     */
} NhlErrorTypes;
```

Description

If the parameter *pid* specifies a **transform** (a plot object), this function draws an immediate-mode polygon in the object's current data coordinate space. The polygon is clipped to the object's viewport. If the parameter *pid* specifies a **workstation**, the polygon is drawn in a data coordinate space that is identical to NDC space with boundaries of (0.0,0.0) at the lower left and (1.0,1.0) at the upper right. Clipping occurs only at the viewspace boundaries. An *immediate-mode* polygon appears as soon as the low-level point buffer is flushed after the function is called. It is *not* drawn when the object's **Draw** method is invoked.

You specify the polygon's attributes by passing in a **GraphicStyle** object using the parameter *gsid*. If *gsid* is set to `NhlNULLOBJID` (0) then the polygon attributes are determined by the current settings of the **Workstation** default **GraphicStyle** object. Retrieve the id of the default **GraphicStyle** object using the read-only **Workstation** resource *wkDefGraphicStyleId*.

Return Values

The **NhlDataPolygon** C function returns a value of type **NhlErrorTypes**, and the **NhlFDataPolygon** Fortran subroutine returns the error in *ierr*. The following table indicates what the various return Error Values mean:

Value	Meaning
NOERROR	function successful
INFO	minor recoverable error
WARNING	recoverable error
FATAL	function failed; continuing the program may result in core dump

---

See Also

- Transform class
- GraphicStyle class
- Workstation class
- Nhl(F)NDCPolygon function

Copyright

Copyright 1987-1999 University Corporation for Atmospheric Research
The use of this Software is governed by a License Agreement.

NCAR Graphics is a registered trademark of the University Corporation for Atmospheric Research.

NhlDataPolyline

The Fortran name of this function is **NhlFDataPolyline**.

Given arrays containing the X and Y coordinates of points in data coordinate space, this function outputs an immediate-mode polyline connecting each of the points.

C Synopsis

```
#include <ncarg/hlu/hlu.h>
#include <hfile for Transform class object>

NhlErrorTypes NhlDataPolyline(
                int     pid,
                int     gsid,
                float   *x,
                float   *y,
                int     n,
                )
```

Fortran Synopsis

```
      subroutine NhlFDataPolyline(pid, gsid, x, y, n, ierr)
      integer pid, gsid, n, ierr
      real x(n), y(n)
```

Arguments

pid (input)
: Id of a **Transform** or **Workstation** object.

gsid (input)
: Id of a **GraphicStyle** object.

x (input)
: Array of X-Axis coordinate values in data coordinate space.

y (input)
: Array of Y-Axis coordinate values in data coordinate space.

n (input)
: Number of elements in the coordinate arrays *x* and *y*.

ierr (output, Fortran only)
: Error code.

Types

```
Type name:              NhlTErrorTypes
Definition:
typedef enum _NhlErrType{
        NhlFATAL        = -4,   /* "FATAL"      */
        NhlWARNING      = -3,   /* "WARNING"    */
        NhlINFO         = -2,   /* "INFO"       */
        NhlNOERROR      = -1    /* "NOERROR"    */
} NhlErrorTypes;
```

Description

If the parameter *pid* specifies a **transform** (a plot object), this function draws an immediate-mode polyline in the object's current data coordinate space. The polyline is clipped to the object's viewport. If the parameter *pid* specifies a **workstation**, the polyline is drawn in a data coordinate space that is identical to NDC space with boundaries of (0.0,0.0) at the lower left and (1.0,1.0) at the upper right. Clipping occurs only at the viewspace boundaries. An *immediate-mode* polyline appears as soon as the low-level point buffer is flushed after the function is called. It is *not* drawn when the object's **Draw** method is invoked.

You specify the polyline's attributes by passing in a **GraphicStyle** object using the parameter *gsid*. If *gsid* is set to `NhlNULLOBJID` (0) then the line attributes are determined by the current settings of the **Workstation** default **GraphicStyle** object. Retrieve the id of the default **GraphicStyle** object using the read-only **Workstation** resource *wkDefGraphicStyleId*.

Return Values

The **NhlDataPolyline** C function returns a value of type **NhlErrorTypes**, and the **NhlFDataPolyline** Fortran subroutine returns the error in *ierr*. The following table indicates what the various return Error Values mean:

Value	Meaning
NOERROR	function successful
INFO	minor recoverable error
WARNING	recoverable error
FATAL	function failed; continuing the program may result in core dump

---

See Also

- Transform class
- GraphicStyle class
- Workstation class
- Nhl(F)NDCPolyline function

Copyright

Copyright 1987-1999 University Corporation for Atmospheric Research

The use of this Software is governed by a License Agreement.

NCAR Graphics is a registered trademark of the University Corporation for Atmospheric Research.

NhlDataPolymarker

The Fortran name of this function is **NhlFDataPolymarker**.

Given arrays containing the X and Y coordinates of points in data coordinate space, this function outputs an immediate-mode marker at each of the points.

C Synopsis

```
#include <ncarg/hlu/hlu.h>
#include <hfile for Transform class object>

NhlErrorTypes NhlDataPolymarker(
                int     pid,
                int     gsid,
                float   *x,
                float   *y,
                int     n,
                )
```

Fortran Synopsis

```
      subroutine NhlFDataPolymarker(pid, gsid, x, y, n, ierr)
      integer pid, gsid, n, ierr
      real x(n), y(n)
```

Arguments

pid (input)
: Id of a **Transform** or **Workstation** object.

gsid (input)
: Id of a **GraphicStyle** object.

x (input)
: Array of X-Axis coordinate values in data coordinate space.

y (input)
: Array of Y-Axis coordinate values in data coordinate space.

n (input)

Number of elements in the coordinate arrays *x* and *y*.

ierr (output, Fortran only)

Error code.

Types

```
Type name:                NhlTErrorTypes
Definition:
typedef enum _NhlErrType{
        NhlFATAL        = -4,   /* "FATAL"       */
        NhlWARNING      = -3,   /* "WARNING"     */
        NhlINFO         = -2,   /* "INFO"        */
        NhlNOERROR      = -1    /* "NOERROR"     */
} NhlErrorTypes;
```

Description

If the parameter *pid* specifies a **transform** (a plot object), this function draws an immediate-mode polymarker (a set of glyphs marking particular locations) in the object's current data coordinate space. The polymarker is clipped to the object's viewport. If the parameter *pid* specifies a **workstation**, the polymarker is drawn in a data coordinate space that is identical to NDC space with boundaries of (0.0,0.0) at the lower left and (1.0,1.0) at the upper right. Clipping occurs only at the viewspace boundaries. An *immediate-mode* polymarker appears as soon as the low-level point buffer is flushed after the function is called. It is *not* drawn when the object's **Draw** method is invoked.

You specify the polymarker's attributes by passing in a **GraphicStyle** object using the parameter *gsid*. If *gsid* is set to `NhlNULLOBJID` (0) then the marker attributes are determined by the current settings of the **Workstation** default **GraphicStyle** object. Retrieve the id of the default **GraphicStyle** object using the read-only **Workstation** resource *wkDefGraphicStyleId*.

Return Values

The **NhlDataPolymarker** C function returns a value of type **NhlErrorTypes**, and the **NhlFDataPolymarker** Fortran subroutine returns the error in *ierr*. The following table indicates what the various return Error Values mean:

Value	Meaning
NOERROR	function successful
INFO	minor recoverable error
WARNING	recoverable error
FATAL	function failed; continuing the program

```
|       may result in core dump
```

See Also

- Transform class
- GraphicStyle class
- Workstation class
- Nhl(F)NDCPolymarker function

Copyright

Copyright 1987-1999 University Corporation for Atmospheric Research
The use of this Software is governed by a License Agreement.

NCAR Graphics is a registered trademark of the University Corporation for Atmospheric Research.

NhlDataToNDC

The Fortran name of this function is **NhlFDataToNDC**.

Given arrays containing the X and Y coordinates of points in data coordinate space, this function returns arrays containing the X and Y coordinates of the same points in NDC space.

C Synopsis

```
#include <ncarg/hlu/hlu.h>
#include <hfile for Transform class object>

NhlErrorTypes NhlDataToNDC(
                int     pid,
                float   *x,
                float   *y,
                int     n,
                float   *xout,
                float   *yout,
                float   *xmissing,
                float   *ymissing,
                int     *status,
                float   *out_of_range
                )
```

Fortran Synopsis

```
 subroutine NhlFDataToNDC(pid, x, y, n, xout, yout, xmissing,
+                         ymissing, ixmissing, iymissing, status,
+                         out_of_range, ierr)
 integer pid, n, ixmissing, iymissing, status, ierr
 float x(n), y(n), xout(n), yout(n), xmissing, ymissing, out_of_range
```

Arguments

pid (input)
: Id of a **Transform** class plot object with an established data transformation.

x (input)
: Array of X-Axis coordinates values in data coordinate space.

y (input)
: Array of Y-Axis coordinates values in data coordinate space.

n (input)
: Number of elements in each of the arrays *x*, *y*, *xout* and *yout*.

xout (output)
: Array for X-Axis output values in NDC space. The user is responsible for providing the appropriate amount of storage.

yout (output)
: Array for Y-Axis output values in NDC space. The user is responsible for providing the appropriate amount of storage.

xmissing (input)
: Missing value for the X-Axis input array. In C, it should be a pointer to this value, and if it is NULL then no missing values exist in the input data.

ymissing (input)
: Missing value for the Y-Axis input array. In C, it should be a pointer to this value, and if it is NULL then no missing values exist in the input data.

ixmissing (input, Fortran only)
: Flag for whether or not any missing values exist for the X-Axis input array; if 1, then missing values exists.

iymissing (input, Fortran only)
: Flag for whether or not any missing values exist for the Y-Axis input array; if 1, then missing value exists.

status (output)
: Result code of the transformation: 1 if any input array value maps to an out-of-range value; 0 otherwise (in C, it should be a pointer to this value).

out_of_range (output)
: Value used in the output array for any input coordinate values that map to an out-of-range value (in C, it should be a pointer to this value).

ierr (output, Fortran only)
: Error code.

Types

```
Type name:              NhlTErrorTypes
Definition:
typedef enum _NhlErrType{
        NhlFATAL        = -4,   /* "FATAL"      */
        NhlWARNING      = -3,   /* "WARNING"    */
        NhlINFO         = -2,   /* "INFO"       */
        NhlNOERROR      = -1    /* "NOERROR"    */
} NhlErrorTypes;
```

Description

This function accepts one or more coordinates in NDC space and maps the coordinates into their respective data coordinate values.

All objects that manage a data transformation have the ability to map points in data coordinate space to NDC points. You can use **NhlDataToNDC** function to perform this operation. It accepts one or more coordinate pairs and converts them. You may specify separate separate missing values for each axis. When **NhlDataToNDC** encounters a missing value in the input array, it does not attempt to transform it, but places it unmodified into the output array.

You must provide storage for the parameters *xout* and *yout*. The parameter *n* is the number of elements in the arrays *x*, *y*, *xout* and *yout*. **NhlDataToNDC** is implemented in such a way that you can use the same storage locations for the arrays *x* and *xout*, and for the arrays *y* and *yout* if you do not mind having the input values overwritten. If you do not want **NhlDataToNDC** to filter the input for missing values, then you should be careful to set both *xmissing* and *ymissing* to NULL.

The output parameter *status* is set to the value 1 if any of the input array coordinates are out-of-range across the transformation. It is set to 0 if all points (other than missing value points) were successfully transformed. The output parameter *out_of_range* is set to the value placed in the output array for any point that could not be transformed.

Return Values

The **NhlDataToNDC** C function returns a value of type **NhlErrorTypes**, and the **NhlFDataToNDC** Fortran subroutine returns the error in *ierr*. The following table indicates what the various return Error Values mean:

Value	Meaning
NOERROR	function successful
INFO	minor recoverable error

```
WARNING |       recoverable error
--------+--------------------------------------------------
FATAL   |       function failed; continuing the program
        |       may result in core dump
```

See Also

- **NhlDataToNDC**
- **Transform**

Copyright

Copyright 1987-1999 University Corporation for Atmospheric Research
The use of this Software is governed by a License Agreement.

NCAR Graphics is a registered trademark of the University Corporation for Atmospheric Research.

NhlDestroy

The Fortran name of this function is **NhlFDestroy**.

This function is used to destroy an HLU object.

C synopsis

```
#include <ncarg/hlu/hlu.h>

NhlErrorTypes NhlDestroy(
                int             obj_id
                )
```

Fortran synopsis

```
      subroutine NhlFDestroy(obj_id, ierr)
      integer obj_id, ierr
```

Arguments

obj_id (input)
Identifies the HLU object to destroy. Returned from the NhlCreate function.
ierr (output, Fortran only)
Error code.

Types

```
Type name:              NhlTErrorTypes
Definition:
typedef enum _NhlErrType{
        NhlFATAL        = -4,   /* "FATAL"      */
        NhlWARNING      = -3,   /* "WARNING"    */
        NhlINFO         = -2,   /* "INFO"       */
        NhlNOERROR      = -1    /* "NOERROR"    */
} NhlErrorTypes;
```

Description

This function is used to free all the memory associated with the HLU object specified by the *obj_id* argument. It is important to note that the **NhlDestroy** function is a recursive function that destroys all the children objects of the object being destroyed. Therefore, if you call **NhlDestroy** on a Workstation class object that has children, those children objects will also be destroyed.

Return values

The **NhlDestroy** C function returns a value of type **NhlErrorTypes**, and the **NhlFDestroy** Fortran subroutine returns the error in *ierr*. The following table indicates what the return error values mean:

Value	Meaning
NOERROR	Object was destroyed with no errors.
INFO	Object was destroyed and there was some recoverable error.
WARNING	Object was destroyed and there was some unrecoverable error.
FATAL	Unable to destroy object. Invalid *obj_id*.

NOTE: Since this is a recursive function, these error codes may actually be for a child of the specified object.

See also

- NhlCreate function

Copyright

Copyright 1987-1999 University Corporation for Atmospheric Research
The use of this Software is governed by a License Agreement.

NCAR Graphics is a registered trademark of the University Corporation for Atmospheric Research.

NhlDraw

The Fortran name of this function is **NhlFDraw**.

This function is used to make an HLU graphical object draw.

C synopsis

```
#include <ncarg/hlu/hlu.h>

NhlErrorTypes NhlDraw(
                int             obj_id
                )
```

Fortran synopsis

```
      subroutine NhlFDraw(obj_id, ierr)
      integer obj_id, ierr
```

Arguments

obj_id (input)
: Identifies the HLU object to draw. Returned from the NhlCreate function.

ierr (output, Fortran only)
: Error code.

Types

```
Type name:                NhlTErrorTypes
Definition:
typedef enum _NhlErrType{
        NhlFATAL        = -4,   /* "FATAL"       */
        NhlWARNING      = -3,   /* "WARNING"     */
        NhlINFO         = -2,   /* "INFO"        */
        NhlNOERROR      = -1    /* "NOERROR"     */
} NhlErrorTypes;
```

Description

The **NhlDraw** function simply draws the object specified by the *obj_id* argument. The object draws to the device that is managed by its Workstation parent. If the object is a Workstation class object, then all the Workstation's child objects are drawn on the device managed by the Workstation object.

Return values

The **NhlDraw** C function returns a value of type **NhlErrorTypes**, and the **NhlFDraw** Fortran subroutine returns the error in *ierr*. The following table indicates what the return error values mean:

Value	Meaning
NOERROR	Object was drawn with no errors.
INFO	Object was drawn, but there was some recoverable error.
WARNING	Object was drawn, but the output may not be correct.
FATAL	The function was unable to draw the object. Either the object is not in a drawable state, or the object is not a graphical object.

See also

- NhlCreate function

Copyright

Copyright 1987-1999 University Corporation for Atmospheric Research
The use of this Software is governed by a License Agreement.

NCAR Graphics is a registered trademark of the University Corporation for Atmospheric Research.

NhlPError

NhlPError, NHLPERROR, NhlErrGetId, NhlErrNumMsgs, NhlErrGetMsg, NhlErrCleanMsgs, NhlErrAddTable, NhlErrSPrintMsg, NhlErrFPrintMsg

The Fortran names of these functions are **NhlFPError**, **NhlFErrGetId**, **NhlFErrNumMsgs**, **NhlFErrGetMsg**, **NhlFErrCleanMsgs**, **NhlFErrSprintMsg**, and **NhlFErrFPrintMsg**. There are no Fortran bindings for **NHLPERROR** and **NhlErrAddTable**.

Error handling functions for the HLU library.

C synopsis

```
#include <ncarg/hlu/hlu.h>

const char *NhlPError(
                NhlErrorTypes   severity,
                int             errnum,
                char            *format,
                ...
                )

void NHLPERROR((
                NhlErrorTypes   severity,
                int             errnum,
                char            *format,
                ...
                ))

int NhlErrGetId()

int NhlErrNumMsgs()

NhlErrorTypes NhlErrGetMsg(
                int             msgnum,
                const NhlErrMsg **msgptr
                )

NhlErrorTypes NhlErrClearMsgs()

NhlErrorTypes NhlErrAddTable(
```

```
                int             start,
                int             tlen,
                const char      **etable
                )

char *NhlErrSPrintMsg(
                char            *buffer,
                const NhlErrMsg *msg
                )

const char *NhlErrFPrintMsg(
                FILE            *fp,
                const NhlErrMsg *msg
                )
```

Fortran synopsis

```
      subroutine NhlFPError(severity, errnum, message)
      character*(*) severity, message
      integer errnum

      subroutine NhlFErrGetId(id_ret)
      integer id_ret

      subroutine NhlFErrNumMsgs(nummsg)
      integer nummsg

      subroutine NhlFErrGetMsg(msgnum, sev, emsg, errorno, sysmsg
     +                         line, fname, ierr)
      integer msgnum, sev, errorno, line, ierr
      character*(*) emsg, sysmsg, fname

      subroutine NhlFErrClearMsgs(ierr)
      integer ierr

      subroutine NhlFErrSprintMsg(buffer, msgnum)
      character*(*) buffer
      integer imsg

      subroutine NhlFErrFprintMsg(iunit, msgnum)
      integer iunit, imsg
```

Arguments

severity (input)
: Specifies the severity level of the error message being reported.

errnum (input)
: Specifies a unique error number that references an error message from an installed error table.

format (input, C only)
: Message to report with argument replacement like format string for printf.

... (input, C only)

Arguments for format string aka printf.

msgnum (input)
: Identifies the index of the error message to retrieve.

msgptr (output, C only)
: Pointer to an NhlErrMsg pointer that fully describes an error message that was reported. This arg is used to retrieve a pointer to an NhlErrMsg.

start (input, C only)
: Index to assign to the first error message in the table being installed.

tlen (input, C only)
: Length of the error table being installed.

etable (input, C only)
: Pointer to an error table to install.

buffer (output)
: Character buffer to place the formatted message into.

msg (input, C only)
: Pointer to an NhlErrMsg structure that fully describes an error message.

fp (input, C only)
: File pointer to print formatted error message to.

message (input, Fortran only)
: Message to report.

id_ret (output, Fortran only)
: Id for *Error object*.

nummsg (output, Fortran only)
: Number of error messages currently buffered.

sev (output, Fortran only)
: Indicates the severity level of the message being retrieved.

emsg (output, Fortran only)
: Actual text of the error message being retrieved. This corresponds to the *format* argument (after argument replacement) if the error was reported from **C** and the *message* argument if it was reported from **Fortran**.

errorno (output, Fortran only)
: Error number of the error message being retrieved.

sysmsg (output, Fortran only)
: Text from the error table for the message being retrieved.

line (output, Fortran only)
: Line number the error message was reported from, or 0.

fname (output, Fortran only)
: File name the error message was reported from, or NULL.

ierr (output, Fortran only)
: Error code.

iunit (input, Fortran only)
: Unit number to print formatted error message to.

Types

```
Type name:               NhlTErrorTypes
Definition:
typedef enum _NhlErrType{
        NhlFATAL         = -4,    /* "FATAL"       */
        NhlWARNING       = -3,    /* "WARNING"     */
        NhlINFO          = -2,    /* "INFO"        */
        NhlNOERROR       = -1     /* "NOERROR"     */
} NhlErrorTypes;

Type name:               <none>
typedef struct _NhlErrMsg{
                NhlErrorTypes   severity;
                char            *msg;
                int             errorno;
                const char      *sysmsg;
                int             line;
                char            *fname;
} NhlErrMsg, *NhlErrMsgList;
```

severity
: Specifies level of this error message.

msg
: Actual Text of this error message.

errorno
: Error number of this error message.

sysmsg
: Actual Text in the error table indexed by *errorno*.

line
: Line number the error message was reported from, or 0.

fname
: File name the error message was reported from, or NULL.

Description

The error handling module of the HLU library has two purposes. First, it allows the programmer to report errors using the same error mechanism as the HLU library if they want to. Second, it allows the programmer to configure the way those errors are reported.

The **NhlPError** function is used to report errors to the error module. The *fmt* and *arg* parameters are parsed in the same way as **printf** parses its arguments. The **NHLPERROR** macro is used in the same way as the **NhlPError** function except that it prepends File and Line information to the error message. Notice the double parentheses, this was necessary to get the macro to take variable number arguments.

Another function that is useful to programmers--if they plan to use the **NhlPError** function to report errors--is **NhlErrAddTable**. This function allows the programmer to add an additional error table in the same format as the system error table. The values 0 - 1000 are reserved for the system error table. The error module will not allow overlapping error tables to be installed.

The error module was implemented as another object in the Nhl library. This is important because all configuration of the Error module is done by setting resources in this object. The **NhlOpen** function creates the Error object so the resources are not available to the programmer at create time, except by using a resource file. However the programmer does have **NhlSetValues** and **NhlGetValues** access. The **NhlErrGetId** function is provided so the programmer can get the Id that is necessary to set or get resources in an object.

The rest of the functions are provided to allow the programmer to do as much of the error handling as they want. They are really only useful if the error object's *NhlNbufferErrors* resource is set to True.

The **NhlErrNumMsgs** function is provided so the programmer can query the error object to find out how many error messages it currently has buffered. The **NhlErrGetMsg** function is used to retrieve a pointer to the error message indexed by msgnum. The error message is saved internally in a *NhlErrMsg* structure. The index is from 1 to *NhlErrNumMsgs*.

The **NhlErrClearMsgs** function is used to clear the Error object's internal message buffer. The **NhlErrFPrintMsg** and **NhlErrSPrintMsg** functions were provided so the programmer can print messages in the *NhlErrMsg* structure format in the same way the error object prints them out internally.

The error object is destroyed when the **NhlClose** function is called. The programmer can use the **NhlPError** function when the error object doesn't exist (before **NhlOpen** or after **NhlClose**). However, all messages are printed to stderr because no configuration is possible.

Return values

The **NhlPError** function returns a `const char*` that points to the formatted error message.

The **NHLPERROR** macro does not return a value.

The **NhlErrGetId** function returns an int that is a unique id for the Error object.

The **NhlErrNumMsgs** function returns an int that is the number of error messages that have been buffered in the Error object.

The **NhlErrGetMsg**, **NhlErrClearMsgs**, and **NhlErrAddTable** functions return values of type NhlErrorTypes. These error eunctions return **NOERROR** if they were able to complete the function, and **FATAL** otherwise.

The **NhlErrSPrintMsg** function returns a pointer to the *buffer* provided, or *NULL* on error.

The **NhlErrFPrintMsg** function returns a pointer to a `const char` buffer that contains the formatted string that was printed to the *fp* argument, or *NULL* on error.

See also

- Error object

Copyright

Copyright 1987-1999 University Corporation for Atmospheric Research
The use of this Software is governed by a License Agreement.

NCAR Graphics is a registered trademark of the University Corporation for Atmospheric Research.

NhlPError

NhlPError, NHLPERROR, NhlErrGetId, NhlErrNumMsgs, NhlErrGetMsg, NhlErrCleanMsgs, NhlErrAddTable, NhlErrSPrintMsg, NhlErrFPrintMsg

The Fortran names of these functions are **NhlFPError**, **NhlFErrGetId**, **NhlFErrNumMsgs**, **NhlFErrGetMsg**, **NhlFErrCleanMsgs**, **NhlFErrSprintMsg**, and **NhlFErrFPrintMsg**. There are no Fortran bindings for **NHLPERROR** and **NhlErrAddTable**.

Error handling functions for the HLU library.

C synopsis

```
#include <ncarg/hlu/hlu.h>

const char *NhlPError(
                NhlErrorTypes   severity,
                int             errnum,
                char            *format,
                ...
                )

void NHLPERROR((
                NhlErrorTypes   severity,
                int             errnum,
                char            *format,
                ...
                ))

int NhlErrGetId()

int NhlErrNumMsgs()

NhlErrorTypes NhlErrGetMsg(
```

```
                int             msgnum,
                const NhlErrMsg **msgptr
                )

NhlErrorTypes NhlErrClearMsgs()

NhlErrorTypes NhlErrAddTable(
                int             start,
                int             tlen,
                const char      **etable
                )

char *NhlErrSPrintMsg(
                char            *buffer,
                const NhlErrMsg *msg
                )

const char *NhlErrFPrintMsg(
                FILE            *fp,
                const NhlErrMsg *msg
                )
```

Fortran synopsis

```
      subroutine NhlFPError(severity, errnum, message)
      character*(*) severity, message
      integer errnum

      subroutine NhlFErrGetId(id_ret)
      integer id_ret

      subroutine NhlFErrNumMsgs(nummsg)
      integer nummsg

      subroutine NhlFErrGetMsg(msgnum, sev, emsg, errorno, sysmsg
     +                         line, fname, ierr)
      integer msgnum, sev, errorno, line, ierr
      character*(*) emsg, sysmsg, fname

      subroutine NhlFErrClearMsgs(ierr)
      integer ierr

      subroutine NhlFErrSprintMsg(buffer, msgnum)
      character*(*) buffer
      integer imsg

      subroutine NhlFErrFprintMsg(iunit, msgnum)
      integer iunit, imsg
```

Arguments

severity (input)

Specifies the severity level of the error message being reported.

errnum (input)
: Specifies a unique error number that references an error message from an installed error table.

format (input, C only)
: Message to report with argument replacement like format string for printf.

... (input, C only)
: Arguments for format string aka printf.

msgnum (input)
: Identifies the index of the error message to retrieve.

msgptr (output, C only)
: Pointer to an NhlErrMsg pointer that fully describes an error message that was reported. This arg is used to retrieve a pointer to an NhlErrMsg.

start (input, C only)
: Index to assign to the first error message in the table being installed.

tlen (input, C only)
: Length of the error table being installed.

etable (input, C only)
: Pointer to an error table to install.

buffer (output)
: Character buffer to place the formatted message into.

msg (input, C only)
: Pointer to an NhlErrMsg structure that fully describes an error message.

fp (input, C only)
: File pointer to print formatted error message to.

message (input, Fortran only)
: Message to report.

id_ret (output, Fortran only)
: Id for *Error object*.

nummsg (output, Fortran only)
: Number of error messages currently buffered.

sev (output, Fortran only)
: Indicates the severity level of the message being retrieved.

emsg (output, Fortran only)
: Actual text of the error message being retrieved. This corresponds to the *format* argument (after argument replacement) if the error was reported from **C** and the *message* argument if it was reported from **Fortran**.

errorno (output, Fortran only)
: Error number of the error message being retrieved.

sysmsg (output, Fortran only)
: Text from the error table for the message being retrieved.

line (output, Fortran only)
: Line number the error message was reported from, or 0.

fname (output, Fortran only)
: File name the error message was reported from, or NULL.

ierr (output, Fortran only)
: Error code.

iunit (input, Fortran only)
: Unit number to print formatted error message to.

Types

```
Type name:               NhlTErrorTypes
Definition:
typedef enum _NhlErrType{
        NhlFATAL         = -4,    /* "FATAL"      */
        NhlWARNING       = -3,    /* "WARNING"    */
        NhlINFO          = -2,    /* "INFO"       */
        NhlNOERROR       = -1     /* "NOERROR"    */
} NhlErrorTypes;

Type name:               <none>
typedef struct _NhlErrMsg{
                NhlErrorTypes   severity;
                char            *msg;
                int             errorno;
                const char      *sysmsg;
                int             line;
                char            *fname;
} NhlErrMsg, *NhlErrMsgList;
```

severity
: Specifies level of this error message.

msg
: Actual Text of this error message.

errorno
: Error number of this error message.

sysmsg
: Actual Text in the error table indexed by *errorno*.

line
: Line number the error message was reported from, or 0.

fname
: File name the error message was reported from, or NULL.

Description

The error handling module of the HLU library has two purposes. First, it allows the programmer to report errors using the same error mechanism as the HLU library if they want to. Second, it allows the programmer to configure the way those errors are reported.

The **NhlPError** function is used to report errors to the error module. The *fmt* and *arg* parameters are parsed in the same way as **printf** parses its arguments. The **NHLPERROR** macro is used in the same way as the **NhlPError** function except that it prepends File and Line information to the error message. Notice the double parentheses, this was necessary to get the macro to take variable number arguments.

Another function that is useful to programmers--if they plan to use the **NhlPError** function to report errors--is **NhlErrAddTable**. This function allows the programmer to add an additional error table in the same format as the system error table. The values 0 - 1000 are reserved for the system error table. The error module will not allow overlapping error tables to be installed.

The error module was implemented as another object in the Nhl library. This is important because all configuration of the Error module is done by setting resources in this object. The **NhlOpen** function creates the Error object so the resources are not available to the programmer at create time, except by using a resource file. However the programmer does have **NhlSetValues** and **NhlGetValues** access. The **NhlErrGetId** function is provided so the programmer can get the Id that is necessary to set or get resources in an object.

The rest of the functions are provided to allow the programmer to do as much of the error handling as they want. They are really only useful if the error object's *NhlNbufferErrors* resource is set to True.

The **NhlErrNumMsgs** function is provided so the programmer can query the error object to find out how many error messages it currently has buffered. The **NhlErrGetMsg** function is used to retrieve a pointer to the error message indexed by msgnum. The error message is saved internally in a *NhlErrMsg* structure. The index is from 1 to *NhlErrNumMsgs*.

The **NhlErrClearMsgs** function is used to clear the Error object's internal message buffer. The **NhlErrFPrintMsg** and **NhlErrSPrintMsg** functions were provided so the programmer can print messages in the *NhlErrMsg* structure format in the same way the error object prints them out internally.

The error object is destroyed when the **NhlClose** function is called. The programmer can use the **NhlPError** function when the error object doesn't exist (before **NhlOpen** or after **NhlClose**). However, all messages are printed to stderr because no configuration is possible.

Return values

The **NhlPError** function returns a `const char*` that points to the formatted error message.

The **NHLPERROR** macro does not return a value.

The **NhlErrGetId** function returns an int that is a unique id for the Error object.

The **NhlErrNumMsgs** function returns an int that is the number of error messages that have been buffered in the Error object.

The **NhlErrGetMsg**, **NhlErrClearMsgs**, and **NhlErrAddTable** functions return values of type NhlErrorTypes. These error eunctions return **NOERROR** if they were able to complete the function, and **FATAL** otherwise.

The **NhlErrSPrintMsg** function returns a pointer to the *buffer* provided, or *NULL* on error.

The **NhlErrFPrintMsg** function returns a pointer to a `const char` buffer that contains the formatted string that was printed to the *fp* argument, or *NULL* on error.

See also

- Error object

Copyright

Copyright 1987-1999 University Corporation for Atmospheric Research
The use of this Software is governed by a License Agreement.

NCAR Graphics is a registered trademark of the University Corporation for Atmospheric Research.

NhlPError

NhlPError, NHLPERROR, NhlErrGetId, NhlErrNumMsgs, NhlErrGetMsg, NhlErrCleanMsgs, NhlErrAddTable, NhlErrSPrintMsg, NhlErrFPrintMsg

The Fortran names of these functions are **NhlFPError**, **NhlFErrGetId**, **NhlFErrNumMsgs**, **NhlFErrGetMsg**, **NhlFErrCleanMsgs**, **NhlFErrSprintMsg**, and **NhlFErrFPrintMsg**. There are no Fortran bindings for **NHLPERROR** and **NhlErrAddTable**.

Error handling functions for the HLU library.

C synopsis

```
#include <ncarg/hlu/hlu.h>

const char *NhlPError(
                NhlErrorTypes   severity,
                int             errnum,
                char            *format,
                ...
                )

void NHLPERROR((
                NhlErrorTypes   severity,
                int             errnum,
                char            *format,
                ...
                ))
```

```
int NhlErrGetId()

int NhlErrNumMsgs()

NhlErrorTypes NhlErrGetMsg(
                int             msgnum,
                const NhlErrMsg **msgptr
                )

NhlErrorTypes NhlErrClearMsgs()

NhlErrorTypes NhlErrAddTable(
                int             start,
                int             tlen,
                const char      **etable
                )

char *NhlErrSPrintMsg(
                char            *buffer,
                const NhlErrMsg *msg
                )

const char *NhlErrFPrintMsg(
                FILE            *fp,
                const NhlErrMsg *msg
                )
```

Fortran synopsis

```
      subroutine NhlFPError(severity, errnum, message)
      character*(*) severity, message
      integer errnum

      subroutine NhlFErrGetId(id_ret)
      integer id_ret

      subroutine NhlFErrNumMsgs(nummsg)
      integer nummsg

      subroutine NhlFErrGetMsg(msgnum, sev, emsg, errorno, sysmsg
     +                         line, fname, ierr)
      integer msgnum, sev, errorno, line, ierr
      character*(*) emsg, sysmsg, fname

      subroutine NhlFErrClearMsgs(ierr)
      integer ierr

      subroutine NhlFErrSprintMsg(buffer, msgnum)
      character*(*) buffer
      integer imsg

      subroutine NhlFErrFprintMsg(iunit, msgnum)
      integer iunit, imsg
```

Arguments

severity (input)
Specifies the severity level of the error message being reported.

errnum (input)
Specifies a unique error number that references an error message from an installed error table.

format (input, C only)
Message to report with argument replacement like format string for printf.

... (input, C only)
Arguments for format string aka printf.

msgnum (input)
Identifies the index of the error message to retrieve.

msgptr (output, C only)
Pointer to an NhlErrMsg pointer that fully describes an error message that was reported. This arg is used to retrieve a pointer to an NhlErrMsg.

start (input, C only)
Index to assign to the first error message in the table being installed.

tlen (input, C only)
Length of the error table being installed.

etable (input, C only)
Pointer to an error table to install.

buffer (output)
Character buffer to place the formatted message into.

msg (input, C only)
Pointer to an NhlErrMsg structure that fully describes an error message.

fp (input, C only)
File pointer to print formatted error message to.

message (input, Fortran only)
Message to report.

id_ret (output, Fortran only)
Id for *Error object*.

nummsg (output, Fortran only)
Number of error messages currently buffered.

sev (output, Fortran only)
Indicates the severity level of the message being retrieved.

emsg (output, Fortran only)
Actual text of the error message being retrieved. This corresponds to the *format* argument (after argument replacement) if the error was reported from **C** and the *message* argument if it was reported from **Fortran**.

errorno (output, Fortran only)
Error number of the error message being retrieved.

sysmsg (output, Fortran only)
Text from the error table for the message being retrieved.

line (output, Fortran only)
Line number the error message was reported from, or 0.

fname (output, Fortran only)
File name the error message was reported from, or NULL.

ierr (output, Fortran only)
Error code.

iunit (input, Fortran only)
Unit number to print formatted error message to.

Types

```
Type name:               NhlTErrorTypes
Definition:
typedef enum _NhlErrType{
        NhlFATAL         = -4,    /* "FATAL"       */
        NhlWARNING       = -3,    /* "WARNING"     */
        NhlINFO          = -2,    /* "INFO"        */
        NhlNOERROR       = -1     /* "NOERROR"     */
} NhlErrorTypes;

Type name:               <none>
typedef struct _NhlErrMsg{
                NhlErrorTypes   severity;
                char            *msg;
                int             errorno;
                const char      *sysmsg;
                int             line;
                char            *fname;
} NhlErrMsg, *NhlErrMsgList;
```

severity
Specifies level of this error message.

msg
Actual Text of this error message.

errorno
Error number of this error message.

sysmsg
Actual Text in the error table indexed by *errorno*.

line
Line number the error message was reported from, or 0.

fname
File name the error message was reported from, or NULL.

Description

The error handling module of the HLU library has two purposes. First, it allows the programmer to report errors using the same error mechanism as the HLU library if they want to. Second, it allows the programmer to configure the way those errors are reported.

The **NhlPError** function is used to report errors to the error module. The *fmt* and *arg* parameters are parsed in the same way as **printf** parses its arguments. The **NHLPERROR** macro is used in the same way as the **NhlPError** function except that it prepends File and Line information to the error message. Notice the double parentheses, this was necessary to get the macro to take variable number arguments.

Another function that is useful to programmers--if they plan to use the **NhlPError** function to report errors--is **NhlErrAddTable**. This function allows the programmer to add an additional error table in the same format as the system error table. The values 0 - 1000 are reserved for the system error table. The error module will not allow overlapping error tables to be installed.

The error module was implemented as another object in the Nhl library. This is important because all configuration of the Error module is done by setting resources in this object. The **NhlOpen** function creates the Error object so the resources are not available to the programmer at create time, except by using a resource file. However the programmer does have **NhlSetValues** and **NhlGetValues** access. The **NhlErrGetId** function is provided so the programmer can get the Id that is necessary to set or get resources in an object.

The rest of the functions are provided to allow the programmer to do as much of the error handling as they want. They are really only useful if the error object's *NhlNbufferErrors* resource is set to True.

The **NhlErrNumMsgs** function is provided so the programmer can query the error object to find out how many error messages it currently has buffered. The **NhlErrGetMsg** function is used to retrieve a pointer to the error message indexed by msgnum. The error message is saved internally in a *NhlErrMsg* structure. The index is from 1 to *NhlErrNumMsgs*.

The **NhlErrClearMsgs** function is used to clear the Error object's internal message buffer. The **NhlErrFPrintMsg** and **NhlErrSPrintMsg** functions were provided so the programmer can print messages in the *NhlErrMsg* structure format in the same way the error object prints them out internally.

The error object is destroyed when the **NhlClose** function is called. The programmer can use the **NhlPError** function when the error object doesn't exist (before **NhlOpen** or after **NhlClose**). However, all messages are printed to stderr because no configuration is possible.

Return values

The **NhlPError** function returns a `const char*` that points to the formatted error message.

The **NHLPERROR** macro does not return a value.

The **NhlErrGetId** function returns an int that is a unique id for the Error object.

The **NhlErrNumMsgs** function returns an int that is the number of error messages that have been buffered in the Error object.

The **NhlErrGetMsg**, **NhlErrClearMsgs**, and **NhlErrAddTable** functions return values of type NhlErrorTypes. These error eunctions return **NOERROR** if they were able to complete the function, and **FATAL** otherwise.

The **NhlErrSPrintMsg** function returns a pointer to the *buffer* provided, or *NULL* on error.

The **NhlErrFPrintMsg** function returns a pointer to a `const char` buffer that contains the formatted string that was printed to the *fp* argument, or *NULL* on error.

See also

- Error object

Copyright

Copyright 1987-1999 University Corporation for Atmospheric Research
The use of this Software is governed by a License Agreement.

NCAR Graphics is a registered trademark of the University Corporation for Atmospheric Research.

NhlPError

NhlPError, NHLPERROR, NhlErrGetId, NhlErrNumMsgs, NhlErrGetMsg, NhlErrCleanMsgs, NhlErrAddTable, NhlErrSPrintMsg, NhlErrFPrintMsg

The Fortran names of these functions are **NhlFPError**, **NhlFErrGetId**, **NhlFErrNumMsgs**, **NhlFErrGetMsg**, **NhlFErrCleanMsgs**, **NhlFErrSprintMsg**, and **NhlFErrFPrintMsg**. There are no Fortran bindings for **NHLPERROR** and **NhlErrAddTable**.

Error handling functions for the HLU library.

C synopsis

```
#include <ncarg/hlu/hlu.h>

const char *NhlPError(
                NhlErrorTypes   severity,
                int             errnum,
                char            *format,
                ...
                )

void NHLPERROR((
```

```
                NhlErrorTypes   severity,
                int             errnum,
                char            *format,
                ...
                ))

int NhlErrGetId()

int NhlErrNumMsgs()

NhlErrorTypes NhlErrGetMsg(
                int             msgnum,
                const NhlErrMsg **msgptr
                )

NhlErrorTypes NhlErrClearMsgs()

NhlErrorTypes NhlErrAddTable(
                int             start,
                int             tlen,
                const char      **etable
                )

char *NhlErrSPrintMsg(
                char            *buffer,
                const NhlErrMsg *msg
                )

const char *NhlErrFPrintMsg(
                FILE            *fp,
                const NhlErrMsg *msg
                )
```

Fortran synopsis

```
      subroutine NhlFPError(severity, errnum, message)
      character*(*) severity, message
      integer errnum

      subroutine NhlFErrGetId(id_ret)
      integer id_ret

      subroutine NhlFErrNumMsgs(nummsg)
      integer nummsg

      subroutine NhlFErrGetMsg(msgnum, sev, emsg, errorno, sysmsg
     +                         line, fname, ierr)
      integer msgnum, sev, errorno, line, ierr
      character*(*) emsg, sysmsg, fname

      subroutine NhlFErrClearMsgs(ierr)
      integer ierr

      subroutine NhlFErrSprintMsg(buffer, msgnum)
      character*(*) buffer
      integer imsg
```

```
subroutine NhlFErrFprintMsg(iunit, msgnum)
integer iunit, imsg
```

Arguments

severity (input)
: Specifies the severity level of the error message being reported.

errnum (input)
: Specifies a unique error number that references an error message from an installed error table.

format (input, C only)
: Message to report with argument replacement like format string for printf.

... (input, C only)
: Arguments for format string aka printf.

msgnum (input)
: Identifies the index of the error message to retrieve.

msgptr (output, C only)
: Pointer to an NhlErrMsg pointer that fully describes an error message that was reported. This arg is used to retrieve a pointer to an NhlErrMsg.

start (input, C only)
: Index to assign to the first error message in the table being installed.

tlen (input, C only)
: Length of the error table being installed.

etable (input, C only)
: Pointer to an error table to install.

buffer (output)
: Character buffer to place the formatted message into.

msg (input, C only)
: Pointer to an NhlErrMsg structure that fully describes an error message.

fp (input, C only)
: File pointer to print formatted error message to.

message (input, Fortran only)
: Message to report.

id_ret (output, Fortran only)
: Id for *Error object*.

nummsg (output, Fortran only)
: Number of error messages currently buffered.

sev (output, Fortran only)
: Indicates the severity level of the message being retrieved.

emsg (output, Fortran only)
: Actual text of the error message being retrieved. This corresponds to the *format* argument (after argument replacement) if the error was reported from **C** and the *message* argument if it was reported from **Fortran**.

errorno (output, Fortran only)
: Error number of the error message being retrieved.

sysmsg (output, Fortran only)
: Text from the error table for the message being retrieved.

line (output, Fortran only)
    Line number the error message was reported from, or 0.
fname (output, Fortran only)
    File name the error message was reported from, or NULL.
ierr (output, Fortran only)
    Error code.
iunit (input, Fortran only)
    Unit number to print formatted error message to.

Types

```
Type name:               NhlTErrorTypes
Definition:
typedef enum _NhlErrType{
        NhlFATAL        = -4,   /* "FATAL"      */
        NhlWARNING      = -3,   /* "WARNING"    */
        NhlINFO         = -2,   /* "INFO"       */
        NhlNOERROR      = -1    /* "NOERROR"    */
} NhlErrorTypes;

Type name:               <none>
typedef struct _NhlErrMsg{
                NhlErrorTypes   severity;
                char            *msg;
                int             errorno;
                const char      *sysmsg;
                int             line;
                char            *fname;
} NhlErrMsg, *NhlErrMsgList;
```

severity
    Specifies level of this error message.
msg
    Actual Text of this error message.
errorno
    Error number of this error message.
sysmsg
    Actual Text in the error table indexed by *errorno*.
line
    Line number the error message was reported from, or 0.
fname
    File name the error message was reported from, or NULL.

Description

The error handling module of the HLU library has two purposes. First, it allows the programmer to report errors using the same error mechanism as the HLU library if they want to. Second, it allows the programmer to configure the way those errors are reported.

The **NhlPError** function is used to report errors to the error module. The *fmt* and *arg* parameters are parsed in the same way as **printf** parses its arguments. The **NHLPERROR** macro is used in the same way as the **NhlPError** function except that it prepends File and Line information to the error message. Notice the double parentheses, this was necessary to get the macro to take variable number arguments.

Another function that is useful to programmers--if they plan to use the **NhlPError** function to report errors--is **NhlErrAddTable**. This function allows the programmer to add an additional error table in the same format as the system error table. The values 0 - 1000 are reserved for the system error table. The error module will not allow overlapping error tables to be installed.

The error module was implemented as another object in the Nhl library. This is important because all configuration of the Error module is done by setting resources in this object. The **NhlOpen** function creates the Error object so the resources are not available to the programmer at create time, except by using a resource file. However the programmer does have **NhlSetValues** and **NhlGetValues** access. The **NhlErrGetId** function is provided so the programmer can get the Id that is necessary to set or get resources in an object.

The rest of the functions are provided to allow the programmer to do as much of the error handling as they want. They are really only useful if the error object's *NhlNbufferErrors* resource is set to True.

The **NhlErrNumMsgs** function is provided so the programmer can query the error object to find out how many error messages it currently has buffered. The **NhlErrGetMsg** function is used to retrieve a pointer to the error message indexed by msgnum. The error message is saved internally in a *NhlErrMsg* structure. The index is from 1 to *NhlErrNumMsgs*.

The **NhlErrClearMsgs** function is used to clear the Error object's internal message buffer. The **NhlErrFPrintMsg** and **NhlErrSPrintMsg** functions were provided so the programmer can print messages in the *NhlErrMsg* structure format in the same way the error object prints them out internally.

The error object is destroyed when the **NhlClose** function is called. The programmer can use the **NhlPError** function when the error object doesn't exist (before **NhlOpen** or after **NhlClose**). However, all messages are printed to stderr because no configuration is possible.

Return values

The **NhlPError** function returns a `const char*` that points to the formatted error message.

The **NHLPERROR** macro does not return a value.

The **NhlErrGetId** function returns an int that is a unique id for the Error object.

The **NhlErrNumMsgs** function returns an int that is the number of error messages that have been buffered in the Error object.

The **NhlErrGetMsg**, **NhlErrClearMsgs**, and **NhlErrAddTable** functions return values of type NhlErrorTypes. These error eunctions return **NOERROR** if they were able to complete the function, and **FATAL** otherwise.

The **NhlErrSPrintMsg** function returns a pointer to the *buffer* provided, or *NULL* on error.

The **NhlErrFPrintMsg** function returns a pointer to a `const char` buffer that contains the formatted string that was printed to the *fp* argument, or *NULL* on error.

See also

- Error object

Copyright

Copyright 1987-1999 University Corporation for Atmospheric Research
The use of this Software is governed by a License Agreement.

NCAR Graphics is a registered trademark of the University Corporation for Atmospheric Research.

NhlPError

NhlPError, NHLPERROR, NhlErrGetId, NhlErrNumMsgs, NhlErrGetMsg, NhlErrCleanMsgs, NhlErrAddTable, NhlErrSPrintMsg, NhlErrFPrintMsg

The Fortran names of these functions are **NhlFPError**, **NhlFErrGetId**, **NhlFErrNumMsgs**, **NhlFErrGetMsg**, **NhlFErrCleanMsgs**, **NhlFErrSprintMsg**, and **NhlFErrFPrintMsg**. There are no Fortran bindings for **NHLPERROR** and **NhlErrAddTable**.

Error handling functions for the HLU library.

C synopsis

```
#include <ncarg/hlu/hlu.h>

const char *NhlPError(
                NhlErrorTypes   severity,
                int             errnum,
```

```
                char             *format,
                ...
                )

void NHLPERROR((
                NhlErrorTypes    severity,
                int              errnum,
                char             *format,
                ...
                ))

int NhlErrGetId()

int NhlErrNumMsgs()

NhlErrorTypes NhlErrGetMsg(
                int              msgnum,
                const NhlErrMsg **msgptr
                )

NhlErrorTypes NhlErrClearMsgs()

NhlErrorTypes NhlErrAddTable(
                int              start,
                int              tlen,
                const char       **etable
                )

char *NhlErrSPrintMsg(
                char             *buffer,
                const NhlErrMsg *msg
                )

const char *NhlErrFPrintMsg(
                FILE             *fp,
                const NhlErrMsg *msg
                )
```

Fortran synopsis

```
      subroutine NhlFPError(severity, errnum, message)
      character*(*) severity, message
      integer errnum

      subroutine NhlFErrGetId(id_ret)
      integer id_ret

      subroutine NhlFErrNumMsgs(nummsg)
      integer nummsg

      subroutine NhlFErrGetMsg(msgnum, sev, emsg, errorno, sysmsg
     +                         line, fname, ierr)
      integer msgnum, sev, errorno, line, ierr
      character*(*) emsg, sysmsg, fname

      subroutine NhlFErrClearMsgs(ierr)
```

```
integer ierr

subroutine NhlFErrSprintMsg(buffer, msgnum)
character*(*) buffer
integer imsg

subroutine NhlFErrFprintMsg(iunit, msgnum)
integer iunit, imsg
```

Arguments

severity (input)
: Specifies the severity level of the error message being reported.

errnum (input)
: Specifies a unique error number that references an error message from an installed error table.

format (input, C only)
: Message to report with argument replacement like format string for printf.

... (input, C only)
: Arguments for format string aka printf.

msgnum (input)
: Identifies the index of the error message to retrieve.

msgptr (output, C only)
: Pointer to an NhlErrMsg pointer that fully describes an error message that was reported. This arg is used to retrieve a pointer to an NhlErrMsg.

start (input, C only)
: Index to assign to the first error message in the table being installed.

tlen (input, C only)
: Length of the error table being installed.

etable (input, C only)
: Pointer to an error table to install.

buffer (output)
: Character buffer to place the formatted message into.

msg (input, C only)
: Pointer to an NhlErrMsg structure that fully describes an error message.

fp (input, C only)
: File pointer to print formatted error message to.

message (input, Fortran only)
: Message to report.

id_ret (output, Fortran only)
: Id for *Error object*.

nummsg (output, Fortran only)
: Number of error messages currently buffered.

sev (output, Fortran only)
: Indicates the severity level of the message being retrieved.

emsg (output, Fortran only)
: Actual text of the error message being retrieved. This corresponds to the *format* argument (after argument replacement) if the error was reported from **C** and the *message* argument if it was

reported from **Fortran**.

errorno (output, Fortran only)
    Error number of the error message being retrieved.

sysmsg (output, Fortran only)
    Text from the error table for the message being retrieved.

line (output, Fortran only)
    Line number the error message was reported from, or 0.

fname (output, Fortran only)
    File name the error message was reported from, or NULL.

ierr (output, Fortran only)
    Error code.

iunit (input, Fortran only)
    Unit number to print formatted error message to.

Types

```
Type name:                NhlTErrorTypes
Definition:
typedef enum _NhlErrType{
        NhlFATAL        = -4,   /* "FATAL"      */
        NhlWARNING      = -3,   /* "WARNING"    */
        NhlINFO         = -2,   /* "INFO"       */
        NhlNOERROR      = -1    /* "NOERROR"    */
} NhlErrorTypes;

Type name:                <none>
typedef struct _NhlErrMsg{
                NhlErrorTypes   severity;
                char            *msg;
                int             errorno;
                const char      *sysmsg;
                int             line;
                char            *fname;
} NhlErrMsg, *NhlErrMsgList;
```

severity
    Specifies level of this error message.

msg
    Actual Text of this error message.

errorno
    Error number of this error message.

sysmsg
    Actual Text in the error table indexed by *errorno*.

line
    Line number the error message was reported from, or 0.

fname
    File name the error message was reported from, or NULL.

Description

The error handling module of the HLU library has two purposes. First, it allows the programmer to report errors using the same error mechanism as the HLU library if they want to. Second, it allows the programmer to configure the way those errors are reported.

The **NhlPError** function is used to report errors to the error module. The *fmt* and *arg* parameters are parsed in the same way as **printf** parses its arguments. The **NHLPERROR** macro is used in the same way as the **NhlPError** function except that it prepends File and Line information to the error message. Notice the double parentheses, this was necessary to get the macro to take variable number arguments.

Another function that is useful to programmers--if they plan to use the **NhlPError** function to report errors--is **NhlErrAddTable**. This function allows the programmer to add an additional error table in the same format as the system error table. The values 0 - 1000 are reserved for the system error table. The error module will not allow overlapping error tables to be installed.

The error module was implemented as another object in the Nhl library. This is important because all configuration of the Error module is done by setting resources in this object. The **NhlOpen** function creates the Error object so the resources are not available to the programmer at create time, except by using a resource file. However the programmer does have **NhlSetValues** and **NhlGetValues** access. The **NhlErrGetId** function is provided so the programmer can get the Id that is necessary to set or get resources in an object.

The rest of the functions are provided to allow the programmer to do as much of the error handling as they want. They are really only useful if the error object's *NhlNbufferErrors* resource is set to True.

The **NhlErrNumMsgs** function is provided so the programmer can query the error object to find out how many error messages it currently has buffered. The **NhlErrGetMsg** function is used to retrieve a pointer to the error message indexed by msgnum. The error message is saved internally in a *NhlErrMsg* structure. The index is from 1 to *NhlErrNumMsgs*.

The **NhlErrClearMsgs** function is used to clear the Error object's internal message buffer. The **NhlErrFPrintMsg** and **NhlErrSPrintMsg** functions were provided so the programmer can print messages in the *NhlErrMsg* structure format in the same way the error object prints them out internally.

The error object is destroyed when the **NhlClose** function is called. The programmer can use the **NhlPError** function when the error object doesn't exist (before **NhlOpen** or after **NhlClose**). However, all messages are printed to stderr because no configuration is possible.

Return values

The **NhlPError** function returns a `const char*` that points to the formatted error message.

The **NHLPERROR** macro does not return a value.

The **NhlErrGetId** function returns an int that is a unique id for the Error object.

The **NhlErrNumMsgs** function returns an int that is the number of error messages that have been buffered in the Error object.

The **NhlErrGetMsg**, **NhlErrClearMsgs**, and **NhlErrAddTable** functions return values of type NhlErrorTypes. These error eunctions return **NOERROR** if they were able to complete the function, and **FATAL** otherwise.

The **NhlErrSPrintMsg** function returns a pointer to the *buffer* provided, or *NULL* on error.

The **NhlErrFPrintMsg** function returns a pointer to a `const char` buffer that contains the formatted string that was printed to the *fp* argument, or *NULL* on error.

See also

- Error object

Copyright

Copyright 1987-1999 University Corporation for Atmospheric Research
The use of this Software is governed by a License Agreement.

NCAR Graphics is a registered trademark of the University Corporation for Atmospheric Research.

NhlPError

NhlPError, NHLPERROR, NhlErrGetId, NhlErrNumMsgs, NhlErrGetMsg, NhlErrCleanMsgs, NhlErrAddTable, NhlErrSPrintMsg, NhlErrFPrintMsg

The Fortran names of these functions are **NhlFPError**, **NhlFErrGetId**, **NhlFErrNumMsgs**, **NhlFErrGetMsg**, **NhlFErrCleanMsgs**, **NhlFErrSprintMsg**, and **NhlFErrFPrintMsg**. There are no Fortran bindings for **NHLPERROR** and **NhlErrAddTable**.

Error handling functions for the HLU library.

C synopsis

```
#include <ncarg/hlu/hlu.h>

const char *NhlPError(
                NhlErrorTypes   severity,
                int             errnum,
                char            *format,
                ...
                )

void NHLPERROR((
                NhlErrorTypes   severity,
                int             errnum,
                char            *format,
                ...
                ))

int NhlErrGetId()

int NhlErrNumMsgs()

NhlErrorTypes NhlErrGetMsg(
                int             msgnum,
                const NhlErrMsg **msgptr
                )

NhlErrorTypes NhlErrClearMsgs()

NhlErrorTypes NhlErrAddTable(
                int             start,
                int             tlen,
                const char      **etable
                )

char *NhlErrSPrintMsg(
                char            *buffer,
                const NhlErrMsg *msg
                )

const char *NhlErrFPrintMsg(
                FILE            *fp,
                const NhlErrMsg *msg
                )
```

Fortran synopsis

```
      subroutine NhlFPError(severity, errnum, message)
      character*(*) severity, message
      integer errnum

      subroutine NhlFErrGetId(id_ret)
      integer id_ret

      subroutine NhlFErrNumMsgs(nummsg)
```

```
      integer nummsg

      subroutine NhlFErrGetMsg(msgnum, sev, emsg, errorno, sysmsg
     +                         line, fname, ierr)
      integer msgnum, sev, errorno, line, ierr
      character*(*) emsg, sysmsg, fname

      subroutine NhlFErrClearMsgs(ierr)
      integer ierr

      subroutine NhlFErrSprintMsg(buffer, msgnum)
      character*(*) buffer
      integer imsg

      subroutine NhlFErrFprintMsg(iunit, msgnum)
      integer iunit, imsg
```

Arguments

severity (input)
: Specifies the severity level of the error message being reported.

errnum (input)
: Specifies a unique error number that references an error message from an installed error table.

format (input, C only)
: Message to report with argument replacement like format string for printf.

... (input, C only)
: Arguments for format string aka printf.

msgnum (input)
: Identifies the index of the error message to retrieve.

msgptr (output, C only)
: Pointer to an NhlErrMsg pointer that fully describes an error message that was reported. This arg is used to retrieve a pointer to an NhlErrMsg.

start (input, C only)
: Index to assign to the first error message in the table being installed.

tlen (input, C only)
: Length of the error table being installed.

etable (input, C only)
: Pointer to an error table to install.

buffer (output)
: Character buffer to place the formatted message into.

msg (input, C only)
: Pointer to an NhlErrMsg structure that fully describes an error message.

fp (input, C only)
: File pointer to print formatted error message to.

message (input, Fortran only)
: Message to report.

id_ret (output, Fortran only)
: Id for *Error object*.

nummsg (output, Fortran only)

Number of error messages currently buffered.

sev (output, Fortran only)

Indicates the severity level of the message being retrieved.

emsg (output, Fortran only)

Actual text of the error message being retrieved. This corresponds to the *format* argument (after argument replacement) if the error was reported from **C** and the *message* argument if it was reported from **Fortran**.

errorno (output, Fortran only)

Error number of the error message being retrieved.

sysmsg (output, Fortran only)

Text from the error table for the message being retrieved.

line (output, Fortran only)

Line number the error message was reported from, or 0.

fname (output, Fortran only)

File name the error message was reported from, or NULL.

ierr (output, Fortran only)

Error code.

iunit (input, Fortran only)

Unit number to print formatted error message to.

Types

```
Type name:                NhlTErrorTypes
Definition:
typedef enum _NhlErrType{
        NhlFATAL        = -4,    /* "FATAL"      */
        NhlWARNING      = -3,    /* "WARNING"    */
        NhlINFO         = -2,    /* "INFO"       */
        NhlNOERROR      = -1     /* "NOERROR"    */
} NhlErrorTypes;

Type name:                <none>
typedef struct _NhlErrMsg{
                NhlErrorTypes   severity;
                char            *msg;
                int             errorno;
                const char      *sysmsg;
                int             line;
                char            *fname;
} NhlErrMsg, *NhlErrMsgList;
```

severity

Specifies level of this error message.

msg

Actual Text of this error message.

errorno

Error number of this error message.

sysmsg

Actual Text in the error table indexed by *errorno*.

line

Line number the error message was reported from, or 0.

fname

File name the error message was reported from, or NULL.

Description

The error handling module of the HLU library has two purposes. First, it allows the programmer to report errors using the same error mechanism as the HLU library if they want to. Second, it allows the programmer to configure the way those errors are reported.

The **NhlPError** function is used to report errors to the error module. The *fmt* and *arg* parameters are parsed in the same way as **printf** parses its arguments. The **NHLPERROR** macro is used in the same way as the **NhlPError** function except that it prepends File and Line information to the error message. Notice the double parentheses, this was necessary to get the macro to take variable number arguments.

Another function that is useful to programmers--if they plan to use the **NhlPError** function to report errors--is **NhlErrAddTable**. This function allows the programmer to add an additional error table in the same format as the system error table. The values 0 - 1000 are reserved for the system error table. The error module will not allow overlapping error tables to be installed.

The error module was implemented as another object in the Nhl library. This is important because all configuration of the Error module is done by setting resources in this object. The **NhlOpen** function creates the Error object so the resources are not available to the programmer at create time, except by using a resource file. However the programmer does have **NhlSetValues** and **NhlGetValues** access. The **NhlErrGetId** function is provided so the programmer can get the Id that is necessary to set or get resources in an object.

The rest of the functions are provided to allow the programmer to do as much of the error handling as they want. They are really only useful if the error object's *NhlNbufferErrors* resource is set to True.

The **NhlErrNumMsgs** function is provided so the programmer can query the error object to find out how many error messages it currently has buffered. The **NhlErrGetMsg** function is used to retrieve a pointer to the error message indexed by msgnum. The error message is saved internally in a *NhlErrMsg* structure. The index is from 1 to *NhlErrNumMsgs*.

The **NhlErrClearMsgs** function is used to clear the Error object's internal message buffer. The **NhlErrFPrintMsg** and **NhlErrSPrintMsg** functions were provided so the programmer can print messages in the *NhlErrMsg* structure format in the same way the error object prints them out internally.

The error object is destroyed when the **NhlClose** function is called. The programmer can use the **NhlPError** function when the error object doesn't exist (before **NhlOpen** or after **NhlClose**). However, all messages are printed to stderr because no configuration is possible.

Return values

The **NhlPError** function returns a `const char*` that points to the formatted error message.

The **NHLPERROR** macro does not return a value.

The **NhlErrGetId** function returns an int that is a unique id for the Error object.

The **NhlErrNumMsgs** function returns an int that is the number of error messages that have been buffered in the Error object.

The **NhlErrGetMsg**, **NhlErrClearMsgs**, and **NhlErrAddTable** functions return values of type NhlErrorTypes. These error eunctions return **NOERROR** if they were able to complete the function, and **FATAL** otherwise.

The **NhlErrSPrintMsg** function returns a pointer to the *buffer* provided, or *NULL* on error.

The **NhlErrFPrintMsg** function returns a pointer to a `const char` buffer that contains the formatted string that was printed to the *fp* argument, or *NULL* on error.

See also

- Error object

Copyright

Copyright 1987-1999 University Corporation for Atmospheric Research
The use of this Software is governed by a License Agreement.

NCAR Graphics is a registered trademark of the University Corporation for Atmospheric Research.

NhlPError

NhlPError, NHLPERROR, NhlErrGetId, NhlErrNumMsgs, NhlErrGetMsg, NhlErrCleanMsgs, NhlErrAddTable, NhlErrSPrintMsg, NhlErrFPrintMsg

The Fortran names of these functions are **NhlFPError**, **NhlFErrGetId**, **NhlFErrNumMsgs**, **NhlFErrGetMsg**, **NhlFErrCleanMsgs**, **NhlFErrSprintMsg**, and **NhlFErrFPrintMsg**. There are no

Fortran bindings for **NHLPERROR** and **NhlErrAddTable**.

Error handling functions for the HLU library.

C synopsis

```
#include <ncarg/hlu/hlu.h>

const char *NhlPError(
                NhlErrorTypes   severity,
                int             errnum,
                char            *format,
                ...
                )

void NHLPERROR((
                NhlErrorTypes   severity,
                int             errnum,
                char            *format,
                ...
                ))

int NhlErrGetId()

int NhlErrNumMsgs()

NhlErrorTypes NhlErrGetMsg(
                int             msgnum,
                const NhlErrMsg **msgptr
                )

NhlErrorTypes NhlErrClearMsgs()

NhlErrorTypes NhlErrAddTable(
                int             start,
                int             tlen,
                const char      **etable
                )

char *NhlErrSPrintMsg(
                char            *buffer,
                const NhlErrMsg *msg
                )

const char *NhlErrFPrintMsg(
                FILE            *fp,
                const NhlErrMsg *msg
                )
```

Fortran synopsis

```
      subroutine NhlFPError(severity, errnum, message)
```

```
      character*(*) severity, message
      integer errnum

      subroutine NhlFErrGetId(id_ret)
      integer id_ret

      subroutine NhlFErrNumMsgs(nummsg)
      integer nummsg

      subroutine NhlFErrGetMsg(msgnum, sev, emsg, errorno, sysmsg
     +                         line, fname, ierr)
      integer msgnum, sev, errorno, line, ierr
      character*(*) emsg, sysmsg, fname

      subroutine NhlFErrClearMsgs(ierr)
      integer ierr

      subroutine NhlFErrSprintMsg(buffer, msgnum)
      character*(*) buffer
      integer imsg

      subroutine NhlFErrFprintMsg(iunit, msgnum)
      integer iunit, imsg
```

Arguments

severity (input)
  Specifies the severity level of the error message being reported.

errnum (input)
  Specifies a unique error number that references an error message from an installed error table.

format (input, C only)
  Message to report with argument replacement like format string for printf.

... (input, C only)
  Arguments for format string aka printf.

msgnum (input)
  Identifies the index of the error message to retrieve.

msgptr (output, C only)
  Pointer to an NhlErrMsg pointer that fully describes an error message that was reported. This arg is used to retrieve a pointer to an NhlErrMsg.

start (input, C only)
  Index to assign to the first error message in the table being installed.

tlen (input, C only)
  Length of the error table being installed.

etable (input, C only)
  Pointer to an error table to install.

buffer (output)
  Character buffer to place the formatted message into.

msg (input, C only)
  Pointer to an NhlErrMsg structure that fully describes an error message.

fp (input, C only)

File pointer to print formatted error message to.

message (input, Fortran only)
: Message to report.

id_ret (output, Fortran only)
: Id for *Error object*.

nummsg (output, Fortran only)
: Number of error messages currently buffered.

sev (output, Fortran only)
: Indicates the severity level of the message being retrieved.

emsg (output, Fortran only)
: Actual text of the error message being retrieved. This corresponds to the *format* argument (after argument replacement) if the error was reported from **C** and the *message* argument if it was reported from **Fortran**.

errorno (output, Fortran only)
: Error number of the error message being retrieved.

sysmsg (output, Fortran only)
: Text from the error table for the message being retrieved.

line (output, Fortran only)
: Line number the error message was reported from, or 0.

fname (output, Fortran only)
: File name the error message was reported from, or NULL.

ierr (output, Fortran only)
: Error code.

iunit (input, Fortran only)
: Unit number to print formatted error message to.

Types

```
Type name:                NhlTErrorTypes
Definition:
typedef enum _NhlErrType{
        NhlFATAL        = -4,   /* "FATAL"       */
        NhlWARNING      = -3,   /* "WARNING"     */
        NhlINFO         = -2,   /* "INFO"        */
        NhlNOERROR      = -1    /* "NOERROR"     */
} NhlErrorTypes;

Type name:                <none>
typedef struct _NhlErrMsg{
                NhlErrorTypes   severity;
                char            *msg;
                int             errorno;
                const char      *sysmsg;
                int             line;
                char            *fname;
} NhlErrMsg, *NhlErrMsgList;
```

severity
: Specifies level of this error message.

msg
Actual Text of this error message.
errorno
Error number of this error message.
sysmsg
Actual Text in the error table indexed by *errorno*.
line
Line number the error message was reported from, or 0.
fname
File name the error message was reported from, or NULL.

Description

The error handling module of the HLU library has two purposes. First, it allows the programmer to report errors using the same error mechanism as the HLU library if they want to. Second, it allows the programmer to configure the way those errors are reported.

The **NhlPError** function is used to report errors to the error module. The *fmt* and *arg* parameters are parsed in the same way as **printf** parses its arguments. The **NHLPERROR** macro is used in the same way as the **NhlPError** function except that it prepends File and Line information to the error message. Notice the double parentheses, this was necessary to get the macro to take variable number arguments.

Another function that is useful to programmers--if they plan to use the **NhlPError** function to report errors--is **NhlErrAddTable**. This function allows the programmer to add an additional error table in the same format as the system error table. The values 0 - 1000 are reserved for the system error table. The error module will not allow overlapping error tables to be installed.

The error module was implemented as another object in the Nhl library. This is important because all configuration of the Error module is done by setting resources in this object. The **NhlOpen** function creates the Error object so the resources are not available to the programmer at create time, except by using a resource file. However the programmer does have **NhlSetValues** and **NhlGetValues** access. The **NhlErrGetId** function is provided so the programmer can get the Id that is necessary to set or get resources in an object.

The rest of the functions are provided to allow the programmer to do as much of the error handling as they want. They are really only useful if the error object's *NhlNbufferErrors* resource is set to True.

The **NhlErrNumMsgs** function is provided so the programmer can query the error object to find out how many error messages it currently has buffered. The **NhlErrGetMsg** function is used to retrieve a pointer to the error message indexed by msgnum. The error message is saved internally in a *NhlErrMsg* structure. The index is from 1 to *NhlErrNumMsgs*.

The **NhlErrClearMsgs** function is used to clear the Error object's internal message buffer. The **NhlErrFPrintMsg** and **NhlErrSPrintMsg** functions were provided so the programmer can print messages in the *NhlErrMsg* structure format in the same way the error object prints them out internally.

The error object is destroyed when the **NhlClose** function is called. The programmer can use the **NhlPError** function when the error object doesn't exist (before **NhlOpen** or after **NhlClose**). However, all messages are printed to stderr because no configuration is possible.

Return values

The **NhlPError** function returns a `const char*` that points to the formatted error message.

The **NHLPERROR** macro does not return a value.

The **NhlErrGetId** function returns an int that is a unique id for the Error object.

The **NhlErrNumMsgs** function returns an int that is the number of error messages that have been buffered in the Error object.

The **NhlErrGetMsg**, **NhlErrClearMsgs**, and **NhlErrAddTable** functions return values of type NhlErrorTypes. These error eunctions return **NOERROR** if they were able to complete the function, and **FATAL** otherwise.

The **NhlErrSPrintMsg** function returns a pointer to the *buffer* provided, or *NULL* on error.

The **NhlErrFPrintMsg** function returns a pointer to a `const char` buffer that contains the formatted string that was printed to the *fp* argument, or *NULL* on error.

See also

- Error object

Copyright

Copyright 1987-1999 University Corporation for Atmospheric Research
The use of this Software is governed by a License Agreement.

NCAR Graphics is a registered trademark of the University Corporation for Atmospheric Research.

NhlFrame

The Fortran name of this function is **NhlFFrame**.

This function first updates and then clears the drawing area of an instance of a **Workstation** object.

C Synopsis

```
#include <ncarg/hlu/hlu.h>
#include <hfile for Workstation class object>

NhlErrorTypes NhlFrame(
                int     workid
                )
```

Fortran Synopsis

```
      subroutine NhlFFrame(workid, ierr)
      integer workid, ierr
```

Arguments

workid (input)
Integer identifier of an instance of a **Workstation** class object
ierr (output, Fortran only)
Error code

Types

```
Type name:                NhlTErrorTypes
Definition:
typedef enum _NhlErrType{
        NhlFATAL        = -4,   /* "FATAL"      */
        NhlWARNING      = -3,   /* "WARNING"    */
        NhlINFO         = -2,   /* "INFO"       */
        NhlNOERROR      = -1    /* "NOERROR"    */
} NhlErrorTypes;
```

Description

After executing **NhlDraw** for one or more plot objects, use this function to update the the drawing area of the **Workstation** and clear its drawing area in preparation for a new plot. A call to **NhlFrame** is equivalent to a call to **NhlUpdateWorkstation** followed by a call to **NhlClearWorkstation**.

Return Values

The **NhlFrame** C function returns a value of type **NhlErrorTypes**, and the **NhlFFrame** Fortran subroutine returns the error in *ierr*. The following table indicates what the various return Error Values mean:

```
Value   |                      Meaning
--------+-------------------------------------------------
NOERROR |       function successful
--------+-------------------------------------------------
INFO    |       minor recoverable error
--------+-------------------------------------------------
WARNING |       recoverable error
--------+-------------------------------------------------
FATAL   |       function failed; continuing the program
        |       may result in core dump
```

See Also

- **NhlDraw**
- **NhlUpdateWorkstation**
- **NhlClearWorkstation**
- **Workstation object**
- **XWorkstation object**
- **NcgmWorkstation object**

Copyright

Copyright 1987-1999 University Corporation for Atmospheric Research
The use of this Software is governed by a License Agreement.

NCAR Graphics is a registered trademark of the University Corporation for Atmospheric Research.

NhlMalloc

NhlMalloc, NhlRealloc, NhlFree

No corresponding Fortran routines.

Memory allocation functions.

Synopsis

```
#include <ncarg/hlu/hlu.h>

void *NhlMalloc(
                unsigned int    size
                )

void *NhlRealloc(
                void            *data,
                unsigned int    size
                )

NhlErrorTypes NhlFree(
                void            *data
                )
```

Arguments

size (input)

Specifies the size of the memory to allocate.

data (input)

Specifies a pointer to memory previously returned from NhlMalloc or NhlRealloc.

Types

```
Type name:                NhlTErrorTypes
Definition:
typedef enum _NhlErrType{
        NhlFATAL        = -4,   /* "FATAL"      */
        NhlWARNING      = -3,   /* "WARNING"    */
        NhlINFO         = -2,   /* "INFO"       */
        NhlNOERROR      = -1    /* "NOERROR"    */
} NhlErrorTypes;
```

Description

These functions behave like the stdlib functions **malloc**, **realloc** and **free**.

NhlMalloc returns a pointer to a block of memory of at least *size* bytes. The argument to **NhlFree** is a pointer to a block previously allocated with **NhlMalloc**. If **NhlMalloc** is called with a *size* of 0, it returns **NULL**.

NhlFree frees the memory pointed at by *data* so the memory can be re-used. **NhlFree** must be called on all memory allocated by the HLU library on behalf of the user. This includes all arrays retrieved from objects using **NhlGetValues**. If **NhlFree** is called with a **NULL** pointer, it doesn't work.

NhlRealloc changes the size of the block of memory pointed at by *data*. It may extend the current block or allocate another block, copy the contents pointed at by *data* to the new location, and then free the old block. It returns a pointer to the new block. If *data* is **NULL**, **NhlRealloc** behaves like **NhlMalloc**.

Return values

NhlMalloc
    returns a pointer to a block of memory, or **NULL** if a *size* 0 is specified or if it can't allocate the memory.
NhlRealloc
    returns a pointer to a block of memory, or **NULL** if a *size* 0 is specified or if it can't allocate the memory.
NhlFree
    returns a value of NhlErrorTypes:
    NhlNOERROR if there is no error, or NhlWARNING if there is a detectable error.

Copyright

Copyright 1987-1999 University Corporation for Atmospheric Research
The use of this Software is governed by a License Agreement.

NCAR Graphics is a registered trademark of the University Corporation for Atmospheric Research.

NhlFreeColor

The Fortran name of this function is **NhlFFreeColor**.

This function removes a color, specified by its HLU index, from the **Workstation** color map.

C Synopsis

```
#include <ncarg/hlu/hlu.h>
#include <hfile for Workstation class object>

NhlErrorTypes NhlFreeColor(
                int     workid,
```

```
                int     ci
                )
```

Fortran Synopsis

```
      subroutine NhlFFreeColor(workid, ci, ierr)
      integer workid, ci, ierr
```

Arguments

workid (input)
: Integer identifier of an instance of a **Workstation** class object

ci (input)
: Integer color index in the range 1 through 255 of a currently allocated color.

ierr (output, Fortran only)
: Error code.

Types

```
Type name:              NhlTErrorTypes
Definition:
typedef enum _NhlErrType{
        NhlFATAL        = -4,   /* "FATAL"      */
        NhlWARNING      = -3,   /* "WARNING"    */
        NhlINFO         = -2,   /* "INFO"       */
        NhlNOERROR      = -1    /* "NOERROR"    */
} NhlErrorTypes;
```

Description

This function deallocates a HLU color index from the **Workstation** object's colormap. An error will be generated if the color index is not currently allocated. Once a color index has been freed, it will be eligible for reallocation during a subsequent call to **NhlNewColor**. If you draw a plot object using a freed color index to specify the color of any of its features, those features will be drawn using the foreground color (HLU color index 1, NhlFOREGROUND).

Return Values

The **NhlFreeColor** C function returns a value of type **NhlErrorTypes**, and the **NhlFFreeColor** Fortran subroutine returns the error in *ierr*. The following table indicates what the various return Error Values mean:

```
Value   |                    Meaning
--------+-----------------------------------------------
NOERROR |       function successful
--------+-----------------------------------------------
INFO    |       minor recoverable error
--------+-----------------------------------------------
WARNING |       recoverable error
--------+-----------------------------------------------
FATAL   |       function failed; continuing the program
        |       may result in core dump
```

See Also

- **NhlNewColor**
- **NhlSetColor**
- **NhlIsAllocatedColor**
- **Workstation object**
- **XWorkstation object**
- **NcgmWorkstation object**

Copyright

Copyright 1987-1999 University Corporation for Atmospheric Research
The use of this Software is governed by a License Agreement.

NCAR Graphics is a registered trademark of the University Corporation for Atmospheric Research.

NhlGetBB

The Fortran name of this function is **NhlFGetBB**.

This function returns the coordinates of the left, right, top, and bottom edges of a **View** class object's bounding box in NDC units.

C Synopsis

```
#include <ncarg/hlu/hlu.h>
```

```
#include <hfile for View class object>

NhlErrorTypes NhlGetBB(
                int             pid,
                NhlBoundingBox  *thebox
                )
```

Fortran Synopsis

```
      subroutine NhlFGetBB(pid, top, bottom, left, right, ierr)
      integer pid, ierr
      real top, bottom, left, right
```

C Arguments

pid (input)
: Id of object whose bounding box is requested.

thebox (output)
: Pointer to a struct of type **NhlBoundingBox** containing on output the NDC values of the top, bottom, left, and right edges of the bounding box of the **View** class object.

Fortran Arguments

pid (input)
: Id of object whose bounding box is requested.

top (output)
: On output contains the NDC value of the top edge of the bounding box of the **View** class object.

bottom (output)
: On output contains the NDC value of the bottom edge of the bounding box of the **View** class object.

left (output)
: On output contains the NDC value of the left edge of the bounding box of the **View** class object.

right (output)
: On output contains the NDC value of the right edge of the bounding box of the **View** class object.

ierr (output)
: Error code.

Types

```
Type name:               <none>
```

```
Definition:
typedef struct _NhlBoundingBox {
                int     set;
                float   t;
                float   b;
                float   l;
                float   r;
} NhlBoundingBox;

Type name:              NhlTErrorTypes
Definition:
typedef enum _NhlErrType{
        NhlFATAL        = -4,   /* "FATAL"      */
        NhlWARNING      = -3,   /* "WARNING"    */
        NhlINFO         = -2,   /* "INFO"       */
        NhlNOERROR      = -1    /* "NOERROR"    */
} NhlErrorTypes;
```

Description

View class objects are not restricted to drawing inside their viewport. A number of elements belonging to a **View** object, such as tickmarks and titles (which may be implemented as child objects of the **View** object in question) are most naturally located outside the viewport. This function allows you to determine the smallest rectangle enclosing all plotted elements belonging to a **View** class object. This rectangle is the *bounding box* of the **View** object.

The struct of type **NhlBoundingBox** represented by the variable *thebox* must have defined storage on entrance to this function. The struct member *set* is used internally by the **NhlGetBB** but has no useful meaning at the user level. The members *t*, *b*, *r*, and *l* return the top, bottom, right, and left coordinates respectively of a rectangular bounding box oriented with the viewport.

Return Values

The **NhlGetBB** C function returns a value of type **NhlErrorTypes**, and the **NhlFGetBB** Fortran subroutine returns the error in *ierr*. The following table indicates what the various return Error Values mean:

Value	Meaning
NOERROR	function successful
INFO	minor recoverable error
WARNING	recoverable error
FATAL	function failed; continuing the program may result in core dump

---

See Also

- **View object**

Copyright

Copyright 1987-1999 University Corporation for Atmospheric Research
The use of this Software is governed by a License Agreement.

NCAR Graphics is a registered trademark of the University Corporation for Atmospheric Research.

NhlGetGksCi

The Fortran name of this function is **NhlFGetGksCi**.

This function returns the low-level GKS color index from an HLU color index and an instance of a **Workstation** object.

C synopsis

```
#include <ncarg/hlu/hlu.h>
#include <hfile for Workstation class object>

int NhlGetGksCi(
                int     workid,
                int     hlu_ci
                )
```

Fortran synopsis

```
      subroutine NhlFGetGksCi(workid, hlu_ci, indx_gks_ret)
      integer workid, hlu_ci, indx_gks_ret
```

Arguments

workid (input)
: Integer identifier of an instance of a **Workstation** class object

hlu_ci (input)
Integer color index in the range 1 through 255 of a currently allocated color in the given *workid* **Workstation** object.

indx_gks_ret (output, Fortran only)
On output, returns the GKS workstation color index for the given hlu color index.

Description

This function retrieves the GKS workstation's color index for the given hlu color index. Given the GKS workstation id, retrieved by getting the **Workstation** resource *wkGksWorkId* and the GKS workstation color index, you can mix calls to the low-level NCAR Graphics library and the GKS library with calls to the HLU library.

Using the **NhlGetGksCi** function, you can use the low-level NCAR Graphics library and the GKS library to draw in the same colors that the HLU library is using.

It is important to note that the HLU library has no way of detecting if the GKS colormap has been modified by GKS calls, so all colormap changes should be done through the HLU library to prevent unpredictable results.

Return values

The **NhlGetGksCi** function returns the GKS color index that is used for the given HLU color index for the given HLU workstation; the **NhlFGetGksCi** Fortran subroutine returns the color index in *indx_gks_ret*. If the given HLU color index is not valid, then 1 will be returned; this is the default foreground color.

See also

- **Workstation object**
- **XWorkstation object**
- **NcgmWorkstation object**

Copyright

Copyright 1987-1999 University Corporation for Atmospheric Research
The use of this Software is governed by a License Agreement.

NCAR Graphics is a registered trademark of the University Corporation for Atmospheric Research.

NhlGetValues

The Fortran name of this function is **NhlFGetValues**.

This function is used to retrieve the resource values of an instance of any HLU object. It is provided as an access function to objects that are Base class.

C synopsis

```
#include <ncarg/hlu/hlu.h>

NhlErrorTypes NhlGetValues(
                int             obj_id,
                int             rlist_id
                )
```

Fortran synopsis

```
      subroutine NhlFGetValues(obj_id, rlist_id, ierr)
      integer obj_id, rlist_id, ierr
```

Arguments

obj_id (input)
    Id that identifies the object. Returned from the NhlCreate function.

rlist_id (input)
    Id of a ResList that specifies which resources to retrieve and where put the retrieved values.

ierr (output, Fortran only)
    Error code.

Types

```
Type name:                NhlTErrorTypes
Definition:
typedef enum _NhlErrType{
        NhlFATAL        = -4,   /* "FATAL"       */
        NhlWARNING      = -3,   /* "WARNING"     */
        NhlINFO         = -2,   /* "INFO"        */
        NhlNOERROR      = -1    /* "NOERROR"     */
```

```
} NhlErrorTypes;
```

Description

The **NhlGetValues** function is used to retrieve the resource values of an object. The **NhlGetValues** function uses the ResList specified by the *rlist_id* parameter to determine which resources to retrieve and where to put the retrieved values. The *rlist_id* is allocated using the NhlRLCreate function. The resource name and value pointer combinations are placed in the ResList using the NhlRLGet, NhlRLGetArray, and NhlRLGetMDArray functions.

The NhlSetValues function can be used to set the resource values of an object.

Return values

The **NhlGetValues** C function returns a value of type **NhlErrorTypes**, and the **NhlFGetValues** Fortran subroutine returns the error in *ierr*. The following table indicates what the return error values mean:

Value	Meaning
NOERROR	Values were retrieved with no detectable errors.
INFO	Error was recoverable - usually indicates that the programmer passed an invalid resource name or value in the *args* argument that was ignored.
WARNING	Error was recoverable - usually indicates that there may be some parts of the object that are not configured properly, and that a FATAL error may occur at a later time.
FATAL	The function was unable to retrieve any values from the object. Could mean the *obj_id* was invalid.

See also

- NhlCreate
- NhlSetValues

Copyright

Copyright 1987-1999 University Corporation for Atmospheric Research
The use of this Software is governed by a License Agreement.

NCAR Graphics is a registered trademark of the University Corporation for Atmospheric Research.

NhlGetWorkspaceObjectId

The Fortran name of this function is **NhlFGetWorkspaceObjectId**.

This function retrieves the **Workspace** objects id.

C synopsis

```
#include <ncarg/hlu/Workspace.h>

int NhlGetWorkspaceObjectId()
```

Fortran synopsis

```
subroutine NhlFGetWorkspaceObjectId(id)
integer id
```

Fortran arguments

id (output)
On output, the id of the **Workspace** object.

Description

This function is used to retrive the id of the **Workspace** object. This id can be used to call NhlSetValues, and NhlGetValues on the **Workspace** object.

See also

- Workspace object

Copyright

Copyright 1987-1999 University Corporation for Atmospheric Research
The use of this Software is governed by a License Agreement.

NCAR Graphics is a registered trademark of the University Corporation for Atmospheric Research.

NhlOpen

NhlOpen, NhlInitialize

The Fortran names of these functions are **NhlFOpen** and **NhlFInitialize**.

HLU library initialization functions.

C synopsis

```
#include <ncarg/hlu/hlu.h>

void NhlOpen()

void NhlInitialize()
```

Fortran synopsis

```
      subroutine NhlFOpen

      subroutine NhlFInitialize
```

Description

These functions are used to initialize the HLU library. One of them must be called before any other HLU calls.

NhlOpen is the normal way to initialize the HLU library. It is the simplest interface.

NhlInitialize differs from **NhlOpen** in that the programmer must create an App object after calling it, and before any other objects can be created.

The HLU library can only be initialized using the **Fortran** binding if the *main* program is written in **Fortran** since that is the only time **Fortran** I/O is guaranteed to be initialized.

The HLU library can only be initialized once, even if **NhlClose** is called, so be sure to only call **NhlClose** when you are completely done using the HLU library.

See also

- App object
- Close Function

Copyright

Copyright 1987-1999 University Corporation for Atmospheric Research
The use of this Software is governed by a License Agreement.

NCAR Graphics is a registered trademark of the University Corporation for Atmospheric Research.

NhlIsAllocatedColor

The Fortran name of this function is **NhlFIsAllocatedColor**.

This boolean function return True if a color has been assigned to a particular HLU color index, and False otherwise.

C Synopsis

```
#include <ncarg/hlu/hlu.h>
#include <hfile for Workstation class object>

NhlBoolean NhlIsAllocatedColor(
                int     workid,
                int     ci
                )
```

Fortran Synopsis

```
subroutine NhlFIsAllocatedColor(workid, ci, isalloc)
integer workid, ci, isalloc
```

Arguments

workid (input)
: Integer identifier of an instance of a **Workstation** class object

ci (input)
: Integer color index in the range 0 through 255.

isalloc (output, Fortran only)
: A flag indicating whether or not a color has been assigned to a particular HLU color index.

Description

This boolean function returns True if the HLU color index is currently allocated for the specified **Workstation** object. Otherwise it returns False.

Return Values

The **NhlIsAllocatedColor** C function returns True or False depending on whether the color index is currently allocated, and the **NhlFIsAllocatedColor** Fortran subroutine returns this value in *isalloc*.

See Also

- **NhlNewColor**
- **NhlSetColor**
- **NhlFreeColor**
- **Workstation object**
- **XWorkstation object**
- **NcgmWorkstation object**

Copyright

Copyright 1987-1999 University Corporation for Atmospheric Research
The use of this Software is governed by a License Agreement.

NCAR Graphics is a registered trademark of the University Corporation for Atmospheric Research.

NhlIsApp

The Fortran name of this function is **NhlFIsApp**.

This boolean function returns True if the given id identifies an **App** class object, and False otherwise.

C Synopsis

```
#include </ncarg/hlu/App.h>

NhlBoolean NhlIsApp(
                int     id,
                )
```

Fortran Synopsis

```
      subroutine NhlFIsApp(id, istatus)
      integer id, istatus
```

Arguments

id (input)
: Specifies an integer id to an existing HLU object.

istatus (output, Fortran only)
: On output, the status of the object.

Description

You use this function to determine if the id you supply identifies an **App** class object.

Return values

The **NhlIsApp** C function returns True if the id identifies an **App** class object and False otherwise; the **NhlFIsApp** Fortran subroutine returns the boolean value in *istatus*.

Copyright

Copyright 1987-1999 University Corporation for Atmospheric Research
The use of this Software is governed by a License Agreement.

NCAR Graphics is a registered trademark of the University Corporation for Atmospheric Research.

NhlIsDataComm

The Fortran name of this function is **NhlFIsDataComm**.

This boolean function returns True if the given id identifies a **DataComm** class object, and False otherwise.

C Synopsis

```
#include </ncarg/hlu/DataComm.h>

NhlBoolean NhlIsDataComm(
                int     id,
                )
```

Fortran Synopsis

```
      subroutine NhlFIsDataComm(id, istatus)
      integer id, istatus
```

Arguments

id (input)
: Specifies an integer id to an existing HLU object.

istatus (output, Fortran only)
: On output, the status of the object.

Description

You use this function to determine if the id you supply identifies a **DataComm** class object.

Return values

The **NhlIsDataComm** C function returns True if the id identifies a **DataComm** class object, and False otherwise; the **NhlFIsDataComm** Fortran subroutine returns the status in *istatus*.

Copyright

Copyright 1987-1999 University Corporation for Atmospheric Research
The use of this Software is governed by a License Agreement.

NCAR Graphics is a registered trademark of the University Corporation for Atmospheric Research.

NhlIsDataItem

The Fortran name of this function is **NhlFIsDataItem**.

This boolean function returns True if the given id identifies a **DataItem** class object, and False otherwise.

C Synopsis

```
#include </ncarg/hlu/DataItem.h>

NhlBoolean NhlIsDataItem(
                int     id,
                )
```

Fortran Synopsis

```
      subroutine NhlFIsDataItem(id, istatus)
      integer id, istatus
```

Arguments

id (input)
    Specifies an integer id to an existing HLU object.
istatus (output, Fortran only)
    On output, the status of the object.

Description

You use this function to determine if the id you supply identifies a **DataItem** class object.

Return values

The **NhlIsDataItem** C function returns True if the id identifies a **DataItem** class object, and False otherwise; the **NhlFIsDataItem** Fortran subroutine returns the status in *istatus*.

Copyright

Copyright 1987-1999 University Corporation for Atmospheric Research
The use of this Software is governed by a License Agreement.

NCAR Graphics is a registered trademark of the University Corporation for Atmospheric Research.

NhlIsDataSpec

The Fortran name of this function is **NhlFIsDataSpec**.

This boolean function returns True if the given id identifies a **DataSpec** class object, and False otherwise.

C Synopsis

```
#include </ncarg/hlu/DataSpec.h>

NhlBoolean NhlIsDataSpec(
                int     id,
                )
```

Fortran Synopsis

```
      subroutine NhlFIsDataSpec(id, istatus)
      integer id, istatus
```

Arguments

id (input)
    Specifies an integer id to an existing HLU object.
istatus (output, Fortran only)
    On output, the status of the object.

Description

You use this function to determine if the id you supply identifies a **DataSpec** class object.

Return values

The **NhlIsDataSpec** C function returns True if the id identifies a **DataSpec** class object, and False otherwise; the **NhlFIsDataSpec** Fortran subroutine returns the status in *istatus*.

Copyright

Copyright 1987-1999 University Corporation for Atmospheric Research
The use of this Software is governed by a License Agreement.

NCAR Graphics is a registered trademark of the University Corporation for Atmospheric Research.

NhlIsTransform

The Fortran name of this function is **NhlFIsTransform**.

This boolean function returns True if the given id identifies a **Transform** class object, and False otherwise.

C Synopsis

```
#include </ncarg/hlu/Transform.h>

NhlBoolean NhlIsTransform(
                 int     id,
                 )
```

Fortran Synopsis

```
subroutine NhlFIsTransform(id, istatus)
integer id, istatus
```

Arguments

id (input)
: Specifies an integer id to an existing HLU object.

istatus (output, Fortran only)
: On output, the status of the object.

Description

You use this function to determine if the id you supply identifies a **Transform** class object.

Return values

The **NhlIsTransform** C function returns True if the id identifies a **Transform** class object, and False otherwise; the **NhlFIsTransform** Fortran subroutine returns the status in *istatus*.

Copyright

Copyright 1987-1999 University Corporation for Atmospheric Research
The use of this Software is governed by a License Agreement.

NCAR Graphics is a registered trademark of the University Corporation for Atmospheric Research.

NhlIsView

The Fortran name of this function is **NhlFIsView**.

This boolean function returns True if the given id identifies a **View** class object, and False otherwise.

C Synopsis

```
#include </ncarg/hlu/View.h>

NhlBoolean NhlIsView(
                int     id,
                )
```

Fortran Synopsis

```
      subroutine NhlFIsView(id, istatus)
      integer id, istatus
```

Arguments

id (input)
Specifies an integer id to an existing HLU object.
istatus (output, Fortran only)
On output, the status of the object.

Description

You use this function to determine if the id you supply identifies a **View** class object.

Return values

The **NhlIsView** C function returns True if the id identifies a **View** class object, and False otherwise; the **NhlFIsView** Fortran subroutine returns the status in *istatus*.

Copyright

Copyright 1987-1999 University Corporation for Atmospheric Research
The use of this Software is governed by a License Agreement.

NCAR Graphics is a registered trademark of the University Corporation for Atmospheric Research.

NhlIsWorkstation

The Fortran name of this function is **NhlFIsWorkstation**.

This boolean function returns True if the given id identifies a **Workstation** class object, and False otherwise.

C Synopsis

```
#include </ncarg/hlu/Workstation.h>

NhlBoolean NhlIsWorkstation(
                int     id,
                )
```

Fortran Synopsis

```
      subroutine NhlFIsWorkstation(id, istatus)
      integer id, istatus
```

Arguments

id (input)
    Specifies an integer id to an existing HLU object.
istatus (output, Fortran only)
    On output, the status of the object.

Description

You use this function to determine if the id you supply identifies a **Workstation** class object.

Return values

The **NhlIsWorkstation** C function returns True if the id identifies a **Workstation** class object, and False otherwise; the **NhlFIsWorkstation** Fortran subroutine returns the status in *istatus*.

Copyright

Copyright 1987-1999 University Corporation for Atmospheric Research
The use of this Software is governed by a License Agreement.

NCAR Graphics is a registered trademark of the University Corporation for Atmospheric Research.

NhlMalloc

NhlMalloc, NhlRealloc, NhlFree

No corresponding Fortran routines.

Memory allocation functions.

Synopsis

```
#include <ncarg/hlu/hlu.h>

void *NhlMalloc(
                 unsigned int    size
                 )

void *NhlRealloc(
                 void            *data,
                 unsigned int    size
                 )

NhlErrorTypes NhlFree(
                 void            *data
                 )
```

Arguments

size (input)
: Specifies the size of the memory to allocate.

data (input)
: Specifies a pointer to memory previously returned from NhlMalloc or NhlRealloc.

Types

```
Type name:                NhlTErrorTypes
Definition:
typedef enum _NhlErrType{
        NhlFATAL        = -4,    /* "FATAL"       */
        NhlWARNING      = -3,    /* "WARNING"     */
        NhlINFO         = -2,    /* "INFO"        */
        NhlNOERROR      = -1     /* "NOERROR"     */
```

```
} NhlErrorTypes;
```

Description

These functions behave like the stdlib functions **malloc**, **realloc** and **free**.

NhlMalloc returns a pointer to a block of memory of at least *size* bytes. The argument to **NhlFree** is a pointer to a block previously allocated with **NhlMalloc**. If **NhlMalloc** is called with a *size* of 0, it returns **NULL**.

NhlFree frees the memory pointed at by *data* so the memory can be re-used. **NhlFree** must be called on all memory allocated by the HLU library on behalf of the user. This includes all arrays retrieved from objects using **NhlGetValues**. If **NhlFree** is called with a **NULL** pointer, it doesn't work.

NhlRealloc changes the size of the block of memory pointed at by *data*. It may extend the current block or allocate another block, copy the contents pointed at by *data* to the new location, and then free the old block. It returns a pointer to the new block. If *data* is **NULL**, **NhlRealloc** behaves like **NhlMalloc**.

Return values

NhlMalloc
: returns a pointer to a block of memory, or **NULL** if a *size* 0 is specified or if it can't allocate the memory.

NhlRealloc
: returns a pointer to a block of memory, or **NULL** if a *size* 0 is specified or if it can't allocate the memory.

NhlFree
: returns a value of NhlErrorTypes:
NhlNOERROR if there is no error, or NhlWARNING if there is a detectable error.

Copyright

Copyright 1987-1999 University Corporation for Atmospheric Research
The use of this Software is governed by a License Agreement.

NCAR Graphics is a registered trademark of the University Corporation for Atmospheric Research.

NhlNDCPolygon

The Fortran name of this function is **NhlFNDCPolygon**.

Given arrays containing the X and Y coordinates of points in NDC space, this function outputs an immediate-mode polygon connecting each of the points.

C Synopsis

```
#include <ncarg/hlu/hlu.h>
#include <hfile for Transform class object>

NhlErrorTypes NhlNDCPolygon(
                int     pid,
                int     gsid,
                float   *x,
                float   *y,
                int     n,
                )
```

Fortran Synopsis

```
      subroutine NhlFNDCPolygon(pid, gsid, x, y, n, ierr)
      integer pid, gsid, n, ierr
      float x(n), y(n)
```

Arguments

pid (input)
: Id of a **Transform** or **Workstation** object.

gsid (input)
: Id of a **GraphicStyle** object.

x (input)
: Array of X-Axis coordinate values in NDC space.

y (input)
: Array of Y-Axis coordinate values in NDC space.

n (input)
: Number of elements in the coordinate arrays *x* and *y*.

ierr (output, Fortran only)
: Error code.

Types

```
Type name:                NhlTErrorTypes
Definition:
typedef enum _NhlErrType{
        NhlFATAL        = -4,   /* "FATAL"       */
        NhlWARNING      = -3,   /* "WARNING"     */
        NhlINFO         = -2,   /* "INFO"        */
        NhlNOERROR      = -1    /* "NOERROR"     */
} NhlErrorTypes;
```

Description

This function draws an immediate-mode polygon in NDC space. If the parameter *pid* specifies a **transform** (a plot object) the polygon is clipped to the object's viewport. If the parameter *pid* specifies a **workstation**, the polygon is clipped only at the boundaries of the NDC viewspace. An *immediate-mode* polygon appears as soon as the low-level point buffer is flushed after the function is called. It is *not* drawn when the object's **Draw** method is invoked.

You specify the polygon's attributes by passing in a **GraphicStyle** object using the parameter *gsid*. If *gsid* is set to `NhlNULLOBJID` (0) then the polygon attributes are determined by the current settings of the **Workstation** default **GraphicStyle** object. Retrieve the id of the default **GraphicStyle** object by getting the value of the read-only **Workstation** resource *wkDefGraphicStyleId*.

Return Values

The **NhlNDCPolygon** C function returns a value of type **NhlErrorTypes**, and the **NhlFNDCPolygon** Fortran subroutine returns the error in *ierr*. The following table indicates what the various return Error Values mean:

Value	Meaning
NOERROR	function successful
INFO	minor recoverable error
WARNING	recoverable error
FATAL	function failed; continuing the program may result in core dump

See Also

- Transform class
- GraphicStyle class
- Workstation class
- Nhl(F)DataPolygon function

Copyright

Copyright 1987-1999 University Corporation for Atmospheric Research
The use of this Software is governed by a License Agreement.

NCAR Graphics is a registered trademark of the University Corporation for Atmospheric Research.

NhlNDCPolyline

The Fortran name of this function is **NhlFNDCPolyline**.

Given arrays containing the X and Y coordinates of points in NDC space, this function outputs an immediate-mode polyline connecting each of the points.

C Synopsis

```
#include <ncarg/hlu/hlu.h>
#include <hfile for Transform class object>

NhlErrorTypes NhlNDCPolyline(
                int     pid,
                int     gsid,
                float   *x,
                float   *y,
                int     n,
                )
```

Fortran Synopsis

```
      subroutine NhlFNDCPolyline(pid, gsid, x, y, n, ierr)
      integer pid, gsid, n, ierr
      float x(n), y(n)
```

Arguments

pid (input)
: Id of a **Transform** or **Workstation** object.

gsid (input)

Id of a **GraphicStyle** object.

x (input)

Array of X-Axis coordinate values in NDC space.

y (input)

Array of Y-Axis coordinate values in NDC space.

n (input)

Number of elements in the coordinate arrays *x* and *y*.

ierr (output, Fortran only)

Error code.

Types

```
Type name:               NhlTErrorTypes
Definition:
typedef enum _NhlErrType{
        NhlFATAL         = -4,    /* "FATAL"       */
        NhlWARNING       = -3,    /* "WARNING"     */
        NhlINFO          = -2,    /* "INFO"        */
        NhlNOERROR       = -1     /* "NOERROR"     */
} NhlErrorTypes;
```

Description

This function draws an immediate-mode polyline in NDC space. If the parameter *pid* specifies a **transform** (a plot object) the polyline is clipped to the object's viewport. If the parameter *pid* specifies a **workstation**, the polyline is clipped only at the boundaries of the NDC viewspace. An *immediate-mode* polyline appears as soon as the low-level point buffer is flushed after the function is called. It is *not* drawn when the object's **Draw** method is invoked.

You specify the polyline's attributes by passing in a **GraphicStyle** object using the parameter *gsid*. If *gsid* is set to `NhlNULLOBJID` (0) then the line attributes are determined by the current settings of the **Workstation** default **GraphicStyle** object. Retrieve the id of the default **GraphicStyle** object using the read-only **Workstation** resource *wkDefGraphicStyleId*.

Return Values

The **NhlNDCPolyline** C function returns a value of type **NhlErrorTypes**, and the **NhlFNDCPolyline** Fortran subroutine returns the error in *ierr*. The following table indicates what the various return Error Values mean:

```
Value   |                      Meaning
--------+----------------------------------------------
NOERROR |       function successful
--------+----------------------------------------------
```

```
INFO    |       minor recoverable error
--------+-------------------------------------------------
WARNING |       recoverable error
--------+-------------------------------------------------
FATAL   |       function failed; continuing the program
        |       may result in core dump
```

See Also

- Transform class
- GraphicStyle class
- Workstation class
- Nhl(F)DataPolyline function

Copyright

Copyright 1987-1999 University Corporation for Atmospheric Research
The use of this Software is governed by a License Agreement.

NCAR Graphics is a registered trademark of the University Corporation for Atmospheric Research.

NhlNDCPolymarker

The Fortran name of this function is **NhlFNDCPolymarker**.

Given arrays containing the X and Y coordinates of points in NDC space, this function outputs an immediate-mode marker at each of the points.

C Synopsis

```
#include <ncarg/hlu/hlu.h>
#include <hfile for Transform class object>

NhlErrorTypes NhlNDCPolymarker(
                int     pid,
                int     gsid,
                float   *x,
                float   *y,
                int     n,
                )
```

Fortran Synopsis

```
subroutine NhlFNDCPolymarker(pid, gsid, x, y, n, ierr)
integer pid, gsid, n, ierr
float x(n), y(n)
```

Arguments

pid (input)
: Id of a **Transform** or **Workstation** object.

gsid (input)
: Id of a **GraphicStyle** object.

x (input)
: Array of X-Axis coordinate values in NDC space.

y (input)
: Array of Y-Axis coordinate values in NDC space.

n (input)
: Number of elements in the coordinate arrays *x* and *y*.

ierr (output, Fortran only)
: Error code.

Types

```
Type name:              NhlTErrorTypes
Definition:
typedef enum _NhlErrType{
        NhlFATAL        = -4,   /* "FATAL"      */
        NhlWARNING      = -3,   /* "WARNING"    */
        NhlINFO         = -2,   /* "INFO"       */
        NhlNOERROR      = -1    /* "NOERROR"    */
} NhlErrorTypes;
```

Description

This function draws an immediate-mode polymarker (a set of glyphs marking particular locations) in NDC space. If the parameter *pid* specifies a **transform** (a plot object) the polymarker is clipped to the object's viewport. If the parameter *pid* specifies a **workstation**, the polymarker is clipped only at the boundaries of the NDC viewspace. An *immediate-mode* polymarker appears as soon as the low-level point buffer is flushed after the function is called. It is *not* drawn when the object's **Draw** method is invoked.

You specify the polymarker's attributes by passing in a **GraphicStyle** object using the parameter *gsid*. If *gsid* is set to `NhlNULLOBJID` (0) then the marker attributes are determined by the current settings of the **Workstation** default **GraphicStyle** object. Retrieve the id of the default **GraphicStyle** object using the read-only **Workstation** resource *wkDefGraphicStyleId*.

Return Values

The **NhlNDCPolymarker** C function returns a value of type **NhlErrorTypes**, and the **NhlFNDCPolymarker** Fortran subroutine returns the error in *ierr*. The following table indicates what the various return Error Values mean:

Value	Meaning
NOERROR	function successful
INFO	minor recoverable error
WARNING	recoverable error
FATAL	function failed; continuing the program may result in core dump

See Also

- Transform class
- GraphicStyle class
- Workstation class
- Nhl(F)DataPolymarker function

Copyright

Copyright 1987-1999 University Corporation for Atmospheric Research
The use of this Software is governed by a License Agreement.

NCAR Graphics is a registered trademark of the University Corporation for Atmospheric Research.

NhlNDCToData

The Fortran name of this function is **NhlFNDCToData**.

Given arrays containing the X and Y coordinates of points in NDC space, this function returns arrays containing the X and Y coordinates of the same points in data coordinate space.

C Synopsis

```
#include <ncarg/hlu/hlu.h>
#include <hfile for Transform class object>

NhlErrorTypes NhlNDCToData(
                int     pid,
                float   *x,
                float   *y,
                int     n,
                float   *xout,
                float   *yout,
                float   *xmissing,
                float   *ymissing,
                int     *status,
                float   *out_of_range
                )
```

Fortran Synopsis

```
      subroutine NhlFNDCToData(pid, x, y, n, xout, yout, xmissing,
     +                         ymissing, ixmissing, iymissing, status,
     +                         out_of_range, ierr)
          integer pid, n, ixmissing, iymissing, status, ierr
          real x(n), y(n), xout(n), yout(n), xmissing, ymissing, out_of_range
```

Arguments

pid (input)
: Id of a **Transform** class plot object with an established data transformation.

x (input)
: Array of X-Axis coordinates values in NDC space.

y (input)
: Array of Y-Axis coordinates values in NDC space.

n (input)
: Number of elements in each of the arrays *x*, *y*, *xout* and *yout*.

xout (output)
: Array for X-Axis output values in data coordinate space. The user is responsible for providing the appropriate amount of storage.

yout (output)
: Array for Y-Axis output values in data coordinate space. The user is responsible for providing the appropriate amount of storage.

xmissing (input)

Missing value for the X-Axis input array. In C, it should be a pointer to this value, and if it is NULL then no missing values exist in input data.

ymissing (input)
: Missing value for the Y-Axis input array. In C, it should be a pointer to this value, and if it is NULL then no missing values exist in input data.

ixmissing (input, Fortran only)
: Flag for whether or not any missing values exist for the X-Axis input array; if 1, then missing values exists.

iymissing (input, Fortran only)
: Flag for whether or not any missing values exist for the Y-Axis input array; if 1, then missing value exists..

status (output)
: Result code of the transformation: 1 if any input array value maps to an out-of-range value; 0 otherwise (in C, it should be a pointer to this value).

out_of_range (output)
: Value used in the output array for any input coordinate values that map to an out-of-range value (in C, it should be a pointer to this value).

ierr (output, Fortran only)
: Error code.

Types

```
Type name:               NhlTErrorTypes
Definition:
typedef enum _NhlErrType{
        NhlFATAL        = -4,    /* "FATAL"      */
        NhlWARNING      = -3,    /* "WARNING"    */
        NhlINFO         = -2,    /* "INFO"       */
        NhlNOERROR      = -1     /* "NOERROR"    */
} NhlErrorTypes;
```

Description

This function accepts one or more coordinates in data coordinates and maps the coordinates into their respective NDC values.

All objects that manage a data transformation have the ability to map points in NDC space to data coordinate points. You can use **NhlNDCToData** function to perform this operation. It accepts one or more coordinate pairs and converts them. You may specify separate separate missing values for each axis. When **NhlNDCToData** encounters a missing value in the input array, it does not attempt to transform it, but places it unmodified into the output array.

You must provide storage for the parameters *xout* and *yout*. The parameter *n* is the number of elements in the arrays *x*, *y*, *xout* and *yout*. **NhlNDCToData** is implemented in such a way that you can use the same storage locations for the arrays *x* and *xout*, and for the arrays *y* and *yout* if you do not mind having the input values overwritten. If you do not want **NhlNDCToData** to filter the input for missing values, then you should be careful to set both *xmissing* and *ymissing* to NULL.

The output parameter *status* is set to the value 1 if any of the input array coordinates are out-of-range across the transformation. It is set to 0 if all points (other than missing value points) were successfully transformed. The output parameter *out_of_range* is set to the value placed in the output array for any point that could not be transformed.

Return Values

The **NhlNDCToData** C function returns a value of type **NhlErrorTypes**, and the **NhlFNDCToData** Fortran subroutine returns the error in *ierr*. The following table indicates what the various return Error Values mean:

```
Value   |                          Meaning
--------+-------------------------------------------------
NOERROR |       function successful
--------+-------------------------------------------------
INFO    |       minor recoverable error
--------+-------------------------------------------------
WARNING |       recoverable error
--------+-------------------------------------------------
FATAL   |       function failed; continuing the program
        |       may result in core dump
```

See Also

- **NhlDataToNDC**
- **Transform object**

Copyright

Copyright 1987-1999 University Corporation for Atmospheric Research
The use of this Software is governed by a License Agreement.

NCAR Graphics is a registered trademark of the University Corporation for Atmospheric Research.

NhlName

The Fortran name of this function is **NhlFName**.

This function returns the "name" of an object.

C synopsis

```
#include <ncarg/hlu/hlu.h>

const char *NhlName(
                int     id
                )
```

Fortran synopsis

```
      subroutine NhlFName(id, name_ret, ierr)
      integer id, ierr
      character*(*) name_ret
```

Arguments

id (input)
: Specifies the object identifier.

name_ret (output, Fortran only)
: The name of the object specified by the *id* argument.

ierr (output, Fortran only)
: Error code.

Types

```
Type name:                NhlTErrorTypes
Definition:
typedef enum _NhlErrType{
        NhlFATAL        = -4,   /* "FATAL"      */
        NhlWARNING      = -3,   /* "WARNING"    */
        NhlINFO         = -2,   /* "INFO"       */
        NhlNOERROR      = -1    /* "NOERROR"    */
} NhlErrorTypes;
```

Description

This function returns the name of the object specified by the *id* argument. This is the name specified during the **NhlCreate** call that created the object.

Return values

The **NhlName** C function returns a valid `const char*` or **NULL** on error.

The **NhlFName** Fortran subroutine returns the error in *ierr*:

Value	Meaning
NOERROR	Object name successfully copied to name_ret
INFO	not possible
WARNING	name_ret wasn’t big enough
FATAL	An invalid object id passed in.

See also

- NhlCreate

Copyright

Copyright 1987-1999 University Corporation for Atmospheric Research
The use of this Software is governed by a License Agreement.

NCAR Graphics is a registered trademark of the University Corporation for Atmospheric Research.

NhlNewColor

The Fortran name of this function is **NhlFNewColor**.

This function adds a color specified as an RGB triple to the colormap of a **Workstation** instance and returns its HLU color index.

C Synopsis

```
#include <ncarg/hlu/hlu.h>
#include <hfile for Workstation class object>

int NhlNewColor(
                int     workid,
                float   red,
                float   green,
                float   blue
                )
```

Fortran Synopsis

```
      subroutine NhlFNewColor(workid, red, green, blue, indx)
      integer workid, indx
      real red, green, blue
```

Arguments

workid (input)
  Integer identifier of an instance of a **Workstation** class object
red (input)
  Value of the red component of the desired color in the range 0.0 to 1.0 inclusive.
green (input)
  Value of the green component of the desired color in the range 0.0 to 1.0 inclusive.
blue (input)
  Value of the blue component of the desired color in the range 0.0 to 1.0 inclusive.
indx (output, Fortran only)
  The HLU color index of a newly allocated color with the specified RGB values.

Types

```
Type name:              NhlTErrorTypes
Definition:
typedef enum _NhlErrType{
        NhlFATAL        = -4,   /* "FATAL"      */
        NhlWARNING      = -3,   /* "WARNING"    */
        NhlINFO         = -2,   /* "INFO"       */
        NhlNOERROR      = -1    /* "NOERROR"    */
} NhlErrorTypes;
```

Description

This function is used to add a new color to a **Workstation** object's colormap. **NhlNewColor** returns a an HLU color index of its own choosing from among those currently unallocated by the **Workstation** object. To set a specific HLU color index you must use the function **NhlSetColor**. If the **Workstation** colormap is full an error message is generated, and the negative return value may be cast to the type **NhlErrorTypes**.

Return Values

If greater than or equal to 0, the integer return value of **NhlNewColor** is the HLU color index of a newly allocated color with RGB values as specified by the function parameters. The Fortran routine, **NhlFNewColor**, returns this value in *indx*. If the return value is less than 0, you may cast the value to the type **NhlErrorTypes**, to determine the error status, as follows:

Value	Meaning
NOERROR	function successful
INFO	minor recoverable error
WARNING	recoverable error
FATAL	function failed; continuing the program may result in core dump

See Also

- **NhlSetColor**
- **NhlFreeColor**
- **NhlIsAllocatedColor**
- **Workstation object**
- **XWorkstation object**
- **NcgmWorkstation object**

Copyright

Copyright 1987-1999 University Corporation for Atmospheric Research
The use of this Software is governed by a License Agreement.

NCAR Graphics is a registered trademark of the University Corporation for Atmospheric Research.

NhlNumber

No corresponding Fortran routine.

This cpp macro determines the number of elements in a fixed-size array.

C Synopsis

```
#include <ncarg/hlu/hlu.h>

unsigned int NhlNumber(
                Array variable
                )
```

Arguments

Array variable (input)
Specifies a fixed-size array variable.

Description

This cpp macro is used to determine the number of elements in a fixed-size array. Since it is a cpp macro, it is only available from the C interface.

Return values

The number of elements in the given array.

Copyright

Copyright 1987-1999 University Corporation for Atmospheric Research
The use of this Software is governed by a License Agreement.

NCAR Graphics is a registered trademark of the University Corporation for Atmospheric Research.

NhlOpen

NhlOpen, NhlInitialize

The Fortran names of these functions are **NhlFOpen** and **NhlFInitialize**.

HLU library initialization functions.

C synopsis

```
#include <ncarg/hlu/hlu.h>

void NhlOpen()

void NhlInitialize()
```

Fortran synopsis

```
      subroutine NhlFOpen

      subroutine NhlFInitialize
```

Description

These functions are used to initialize the HLU library. One of them must be called before any other HLU calls.

NhlOpen is the normal way to initialize the HLU library. It is the simplest interface.

NhlInitialize differs from **NhlOpen** in that the programmer must create an App object after calling it, and before any other objects can be created.

The HLU library can only be initialized using the **Fortran** binding if the *main* program is written in **Fortran** since that is the only time **Fortran** I/O is guaranteed to be initialized.

The HLU library can only be initialized once, even if **NhlClose** is called, so be sure to only call **NhlClose** when you are completely done using the HLU library.

See also

- App object
- Close Function

Copyright

Copyright 1987-1999 University Corporation for Atmospheric Research
The use of this Software is governed by a License Agreement.

NCAR Graphics is a registered trademark of the University Corporation for Atmospheric Research.

NhlPError

NhlPError, NHLPERROR, NhlErrGetId, NhlErrNumMsgs, NhlErrGetMsg, NhlErrCleanMsgs, NhlErrAddTable, NhlErrSPrintMsg, NhlErrFPrintMsg

The Fortran names of these functions are **NhlFPError**, **NhlFErrGetId**, **NhlFErrNumMsgs**, **NhlFErrGetMsg**, **NhlFErrCleanMsgs**, **NhlFErrSprintMsg**, and **NhlFErrFPrintMsg**. There are no Fortran bindings for **NHLPERROR** and **NhlErrAddTable**.

Error handling functions for the HLU library.

C synopsis

```
#include <ncarg/hlu/hlu.h>

const char *NhlPError(
                NhlErrorTypes   severity,
                int             errnum,
                char            *format,
                ...
                )

void NHLPERROR((
                NhlErrorTypes   severity,
                int             errnum,
                char            *format,
                ...
                ))

int NhlErrGetId()

int NhlErrNumMsgs()

NhlErrorTypes NhlErrGetMsg(
                int             msgnum,
                const NhlErrMsg **msgptr
                )

NhlErrorTypes NhlErrClearMsgs()
```

```
NhlErrorTypes NhlErrAddTable(
                int             start,
                int             tlen,
                const char      **etable
                )

char *NhlErrSPrintMsg(
                char            *buffer,
                const NhlErrMsg *msg
                )

const char *NhlErrFPrintMsg(
                FILE            *fp,
                const NhlErrMsg *msg
                )
```

Fortran synopsis

```
      subroutine NhlFPError(severity, errnum, message)
      character*(*) severity, message
      integer errnum

      subroutine NhlFErrGetId(id_ret)
      integer id_ret

      subroutine NhlFErrNumMsgs(nummsg)
      integer nummsg

      subroutine NhlFErrGetMsg(msgnum, sev, emsg, errorno, sysmsg
     +                         line, fname, ierr)
      integer msgnum, sev, errorno, line, ierr
      character*(*) emsg, sysmsg, fname

      subroutine NhlFErrClearMsgs(ierr)
      integer ierr

      subroutine NhlFErrSprintMsg(buffer, msgnum)
      character*(*) buffer
      integer imsg

      subroutine NhlFErrFprintMsg(iunit, msgnum)
      integer iunit, imsg
```

Arguments

severity (input)
: Specifies the severity level of the error message being reported.

errnum (input)
: Specifies a unique error number that references an error message from an installed error table.

format (input, C only)

Message to report with argument replacement like format string for printf.

... (input, C only)
: Arguments for format string aka printf.

msgnum (input)
: Identifies the index of the error message to retrieve.

msgptr (output, C only)
: Pointer to an NhlErrMsg pointer that fully describes an error message that was reported. This arg is used to retrieve a pointer to an NhlErrMsg.

start (input, C only)
: Index to assign to the first error message in the table being installed.

tlen (input, C only)
: Length of the error table being installed.

etable (input, C only)
: Pointer to an error table to install.

buffer (output)
: Character buffer to place the formatted message into.

msg (input, C only)
: Pointer to an NhlErrMsg structure that fully describes an error message.

fp (input, C only)
: File pointer to print formatted error message to.

message (input, Fortran only)
: Message to report.

id_ret (output, Fortran only)
: Id for *Error object*.

nummsg (output, Fortran only)
: Number of error messages currently buffered.

sev (output, Fortran only)
: Indicates the severity level of the message being retrieved.

emsg (output, Fortran only)
: Actual text of the error message being retrieved. This corresponds to the *format* argument (after argument replacement) if the error was reported from **C** and the *message* argument if it was reported from **Fortran**.

errorno (output, Fortran only)
: Error number of the error message being retrieved.

sysmsg (output, Fortran only)
: Text from the error table for the message being retrieved.

line (output, Fortran only)
: Line number the error message was reported from, or 0.

fname (output, Fortran only)
: File name the error message was reported from, or NULL.

ierr (output, Fortran only)
: Error code.

iunit (input, Fortran only)
: Unit number to print formatted error message to.

Types

```
Type name:               NhlTErrorTypes
Definition:
typedef enum _NhlErrType{
        NhlFATAL         = -4,    /* "FATAL"       */
        NhlWARNING       = -3,    /* "WARNING"     */
        NhlINFO          = -2,    /* "INFO"        */
        NhlNOERROR       = -1     /* "NOERROR"     */
} NhlErrorTypes;

Type name:               <none>
typedef struct _NhlErrMsg{
                NhlErrorTypes   severity;
                char            *msg;
                int             errorno;
                const char      *sysmsg;
                int             line;
                char            *fname;
} NhlErrMsg, *NhlErrMsgList;
```

severity
: Specifies level of this error message.

msg
: Actual Text of this error message.

errorno
: Error number of this error message.

sysmsg
: Actual Text in the error table indexed by *errorno*.

line
: Line number the error message was reported from, or 0.

fname
: File name the error message was reported from, or NULL.

Description

The error handling module of the HLU library has two purposes. First, it allows the programmer to report errors using the same error mechanism as the HLU library if they want to. Second, it allows the programmer to configure the way those errors are reported.

The **NhlPError** function is used to report errors to the error module. The *fmt* and *arg* parameters are parsed in the same way as **printf** parses its arguments. The **NHLPERROR** macro is used in the same way as the **NhlPError** function except that it prepends File and Line information to the error message. Notice the double parentheses, this was necessary to get the macro to take variable number arguments.

Another function that is useful to programmers--if they plan to use the **NhlPError** function to report errors--is **NhlErrAddTable**. This function allows the programmer to add an additional error table in the same format as the system error table. The values 0 - 1000 are reserved for the system error table. The error module will not allow overlapping error tables to be installed.

The error module was implemented as another object in the Nhl library. This is important because all configuration of the Error module is done by setting resources in this object. The **NhlOpen** function creates the Error object so the resources are not available to the programmer at create time, except by using a resource file. However the programmer does have **NhlSetValues** and **NhlGetValues** access. The **NhlErrGetId** function is provided so the programmer can get the Id that is necessary to set or get resources in an object.

The rest of the functions are provided to allow the programmer to do as much of the error handling as they want. They are really only useful if the error object's *NhlNbufferErrors* resource is set to True.

The **NhlErrNumMsgs** function is provided so the programmer can query the error object to find out how many error messages it currently has buffered. The **NhlErrGetMsg** function is used to retrieve a pointer to the error message indexed by msgnum. The error message is saved internally in a *NhlErrMsg* structure. The index is from 1 to *NhlErrNumMsgs*.

The **NhlErrClearMsgs** function is used to clear the Error object's internal message buffer. The **NhlErrFPrintMsg** and **NhlErrSPrintMsg** functions were provided so the programmer can print messages in the *NhlErrMsg* structure format in the same way the error object prints them out internally.

The error object is destroyed when the **NhlClose** function is called. The programmer can use the **NhlPError** function when the error object doesn't exist (before **NhlOpen** or after **NhlClose**). However, all messages are printed to stderr because no configuration is possible.

Return values

The **NhlPError** function returns a `const char*` that points to the formatted error message.

The **NHLPERROR** macro does not return a value.

The **NhlErrGetId** function returns an int that is a unique id for the Error object.

The **NhlErrNumMsgs** function returns an int that is the number of error messages that have been buffered in the Error object.

The **NhlErrGetMsg**, **NhlErrClearMsgs**, and **NhlErrAddTable** functions return values of type NhlErrorTypes. These error eunctions return **NOERROR** if they were able to complete the function, and **FATAL** otherwise.

The **NhlErrSPrintMsg** function returns a pointer to the *buffer* provided, or *NULL* on error.

The **NhlErrFPrintMsg** function returns a pointer to a `const char` buffer that contains the formatted string that was printed to the *fp* argument, or *NULL* on error.

See also

- Error object

Copyright

Copyright 1987-1999 University Corporation for Atmospheric Research
The use of this Software is governed by a License Agreement.

NCAR Graphics is a registered trademark of the University Corporation for Atmospheric Research.

NhlRLCreate

NhlRLCreate, NhlRLDestroy, NhlRLClear, NhlRLUnSet, NhlRLIsSet

The Fortran names of these functions are **NhlFRLCreate**, **NhlFRLDestroy**, **NhlFRLClear**, **NhlFRLUnSet**, and **NhlFRLIsSet**. *ResList* support functions.

C synopsis

```
#include <ncarg/hlu/hlu.h>

int NhlRLCreate(
                NhlRLType       list_type
                )

void NhlRLDestroy(
                int             list_id
                )

void NhlRLClear(
                int             list_id
                )

void NhlRLUnSet(
                int             list_id,
                NhlString       resname
                )

NhlBoolean NhlRLIsSet(
                int             list_id,
                NhlString       resname
                )
```

Fortran synopsis

```
subroutine NhlFRLCreate(list_id, list_type)
integer list_id
character*(*) list_type

subroutine nhlfrldestroy(list_id)
integer list_id

subroutine nhlfrlclear(list_id)
integer list_id

subroutine nhlfrlunset(list_id, resname)
integer list_id
character*(*) resname

subroutine nhlfrlisset(list_id, resname, ival)
integer list_id, ival
character*(*) resname
```

Arguments

list_type (input)
: Specifies the type of ResList to create. Use NhlSETRL (or 'setrl' in Fortran) for a list being used to set values in an object (**NhlCreate** and **NhlSetValues**). Use NhlGETRL (or 'getrl' in Fortran) for a list being used to retrieve values from an object. (**NhlGetValues**)

list_id (input/output)
: Integer identifier for the *ResList*. It is returned from the **NhlRLCreate** function.

resname (input)
: Resource name. In the **NhlRLIsSet** function, this is used to determine if the given resource name has been set in the *ResList*. In the **NhlRLUnSet** function, it is used to remove the given resource name from the *ResList*.

ival (input, Fortran only)
: Flag indicating whether or not a resource specified by *resname* is in the *ResList*.

Types

```
Type name:              <none>
typedef enum NhlRLType_{
        NhlSETRL,       /* "setrl" */
        NhlGETRL        /* "getrl" */
} NhlRlType;
```

Description

The **NhlRLCreate** function is used to create a ResList. A ResList is a dynamic list that stores *name/value* pairs that are assigned to it. The **NhlRLCreate** function returns a *list_id* that is used as an argument to the Create, SetValues, and GetValues *ResList* functions, as well as the other *ResList* support functions.

The **NhlRLDestroy** function is used to *free* an existing *ResList*. The *list_id* number may be reused the next time a **NhlRLCreate** is called.

The **NhlRLClear** function is used to clear all the resource names and values out of the *ResList*. It basically empties out the list.

The **NhlRLUnSet** function is used to remove the resource name specified by the *resname* argument from the *ResList*.

The **NhlRLIsSet** function is used to find out if the resource name specified by the *resname* argument is currently in the *ResList*.

Return values

The **NhlRLCreate** C function returns a positive integer that is used to identify the *ResList*, and the **NhlFRLCreate** Fortran subroutine returns this value in *list_id*. If it returns a non-positive number, then it was unable to create a *ResList*.

The **NhlRLIsSet** function returns True if *resname* is currently contained in the *ResList* and False if it is not; the **NhlFRLIsSet** Fortran subroutine returns this value in *ival*.

See also

- Create function
- SetValues function
- GetValues function

Copyright

Copyright 1987-1999 University Corporation for Atmospheric Research
The use of this Software is governed by a License Agreement.

NCAR Graphics is a registered trademark of the University Corporation for Atmospheric Research.

NhlRLCreate

NhlRLCreate, NhlRLDestroy, NhlRLClear, NhlRLUnSet, NhlRLIsSet

The Fortran names of these functions are **NhlFRLCreate**, **NhlFRLDestroy**, **NhlFRLClear**, **NhlFRLUnSet**, and **NhlFRLIsSet**. *ResList* support functions.

C synopsis

```
#include <ncarg/hlu/hlu.h>

int NhlRLCreate(
                NhlRLType       list_type
                )

void NhlRLDestroy(
                int             list_id
                )

void NhlRLClear(
                int             list_id
                )

void NhlRLUnSet(
                int             list_id,
                NhlString       resname
                )

NhlBoolean NhlRLIsSet(
                int             list_id,
                NhlString       resname
                )
```

Fortran synopsis

```
      subroutine NhlFRLCreate(list_id, list_type)
      integer list_id
      character*(*) list_type

      subroutine nhlfrldestroy(list_id)
      integer list_id

      subroutine nhlfrlclear(list_id)
      integer list_id

      subroutine nhlfrlunset(list_id, resname)
      integer list_id
```

```
character*(*) resname

subroutine nhlfrlisset(list_id, resname, ival)
integer list_id, ival
character*(*) resname
```

Arguments

list_type (input)
: Specifies the type of ResList to create. Use NhlSETRL (or 'setrl' in Fortran) for a list being used to set values in an object (**NhlCreate** and **NhlSetValues**). Use NhlGETRL (or 'getrl' in Fortran) for a list being used to retrieve values from an object. (**NhlGetValues**)

list_id (input/output)
: Integer identifier for the *ResList*. It is returned from the **NhlRLCreate** function.

resname (input)
: Resource name. In the **NhlRLIsSet** function, this is used to determine if the given resource name has been set in the *ResList*. In the **NhlRLUnSet** function, it is used to remove the given resource name from the *ResList*.

ival (input, Fortran only)
: Flag indicating whether or not a resource specified by *resname* is in the *ResList*.

Types

```
Type name:              <none>
typedef enum NhlRLType_{
        NhlSETRL,       /* "setrl" */
        NhlGETRL        /* "getrl" */
} NhlRlType;
```

Description

The **NhlRLCreate** function is used to create a ResList. A ResList is a dynamic list that stores *name/value* pairs that are assigned to it. The **NhlRLCreate** function returns a *list_id* that is used as an argument to the Create, SetValues, and GetValues *ResList* functions, as well as the other *ResList* support functions.

The **NhlRLDestroy** function is used to *free* an existing *ResList*. The *list_id* number may be reused the next time a **NhlRLCreate** is called.

The **NhlRLClear** function is used to clear all the resource names and values out of the *ResList*. It basically empties out the list.

The **NhlRLUnSet** function is used to remove the resource name specified by the *resname* argument from the *ResList*.

The **NhlRLIsSet** function is used to find out if the resource name specified by the *resname* argument is currently in the *ResList*.

Return values

The **NhlRLCreate** C function returns a positive integer that is used to identify the *ResList*, and the **NhlFRLCreate** Fortran subroutine returns this value in *list_id*. If it returns a non-positive number, then it was unable to create a *ResList*.

The **NhlRLIsSet** function returns True if *resname* is currently contained in the *ResList* and False if it is not; the **NhlFRLIsSet** Fortran subroutine returns this value in *ival*.

See also

- Create function
- SetValues function
- GetValues function

Copyright

Copyright 1987-1999 University Corporation for Atmospheric Research
The use of this Software is governed by a License Agreement.

NCAR Graphics is a registered trademark of the University Corporation for Atmospheric Research.

NhlRLCreate

NhlRLCreate, NhlRLDestroy, NhlRLClear, NhlRLUnSet, NhlRLIsSet

The Fortran names of these functions are **NhlFRLCreate**, **NhlFRLDestroy**, **NhlFRLClear**, **NhlFRLUnSet**, and **NhlFRLIsSet**. *ResList* support functions.

C synopsis

```
#include <ncarg/hlu/hlu.h>

int NhlRLCreate(
                NhlRLType       list_type
                )

void NhlRLDestroy(
                int             list_id
                )

void NhlRLClear(
                int             list_id
                )

void NhlRLUnSet(
                int             list_id,
                NhlString       resname
                )

NhlBoolean NhlRLIsSet(
                int             list_id,
                NhlString       resname
                )
```

Fortran synopsis

```
subroutine NhlFRLCreate(list_id, list_type)
integer list_id
character*(*) list_type

subroutine nhlfrldestroy(list_id)
integer list_id

subroutine nhlfrlclear(list_id)
integer list_id

subroutine nhlfrlunset(list_id, resname)
integer list_id
character*(*) resname

subroutine nhlfrlisset(list_id, resname, ival)
integer list_id, ival
character*(*) resname
```

Arguments

list_type (input)

Specifies the type of ResList to create. Use NhlSETRL (or 'setrl' in Fortran) for a list being used to set values in an object (**NhlCreate** and **NhlSetValues**). Use NhlGETRL (or 'getrl' in Fortran) for a list being used to retrieve values from an object. (**NhlGetValues**)

list_id (input/output)

Integer identifier for the *ResList*. It is returned from the **NhlRLCreate** function.

resname (input)

Resource name. In the **NhlRLIsSet** function, this is used to determine if the given resource name has been set in the *ResList*. In the **NhlRLUnSet** function, it is used to remove the given resource name from the *ResList*.

ival (input, Fortran only)

Flag indicating whether or not a resource specified by *resname* is in the *ResList*.

Types

```
Type name:              <none>
typedef enum NhlRLType_{
        NhlSETRL,       /* "setrl" */
        NhlGETRL        /* "getrl" */
} NhlRlType;
```

Description

The **NhlRLCreate** function is used to create a ResList. A ResList is a dynamic list that stores *name/value* pairs that are assigned to it. The **NhlRLCreate** function returns a *list_id* that is used as an argument to the Create, SetValues, and GetValues *ResList* functions, as well as the other *ResList* support functions.

The **NhlRLDestroy** function is used to *free* an existing *ResList*. The *list_id* number may be reused the next time a **NhlRLCreate** is called.

The **NhlRLClear** function is used to clear all the resource names and values out of the *ResList*. It basically empties out the list.

The **NhlRLUnSet** function is used to remove the resource name specified by the *resname* argument from the *ResList*.

The **NhlRLIsSet** function is used to find out if the resource name specified by the *resname* argument is currently in the *ResList*.

Return values

The **NhlRLCreate** C function returns a positive integer that is used to identify the *ResList*, and the **NhlFRLCreate** Fortran subroutine returns this value in *list_id*. If it returns a non-positive number, then it was unable to create a *ResList*.

The **NhlRLIsSet** function returns True if *resname* is currently contained in the *ResList* and False if it is not; the **NhlFRLIsSet** Fortran subroutine returns this value in *ival*.

See also

- Create function
- SetValues function
- GetValues function

Copyright

Copyright 1987-1999 University Corporation for Atmospheric Research
The use of this Software is governed by a License Agreement.

NCAR Graphics is a registered trademark of the University Corporation for Atmospheric Research.

NhlRLGet

NhlRLGet, NhlRLGetInteger, NhlRLGetFloat, NhlRLGetDouble, NhlRLGetString

The Fortran names of these functions are **NhlFRLGetInteger**, **NhlFRLGetFloat**, **NhlFRLGetDouble**, **NhlFRLGetString**. There is no Fortran binding for **NhlRLGet**.

Used to add the given *name* and *val_addr* to the *ResList*.

C synopsis

```
#include <ncarg/hlu/hlu.h>

NhlErrorTypes NhlRLGet(
                int             list_id,
                NhlString       resname,
                NhlString       type,
                <AnyType>       *val_addr
                )

NhlErrorTypes NhlRLGetInteger(
                int             list_id,
                NhlString       resname,
                int             *val_addr
                )
```

```
NhlErrorTypes NhlRLGetFloat(
                int             list_id,
                NhlString       resname,
                float           *val_addr
                )

NhlErrorTypes NhlRLGetDouble(
                int             list_id,
                NhlString       resname,
                double          *val_addr
                )

NhlErrorTypes NhlRLGetString(
                int             list_id,
                NhlString       resname,
                NhlString       *val_addr
                )
```

Fortran synopsis

```
subroutine NhlFRLGetInteger(list_id, resname, ival, ierr)
integer list_id, ival, ierr
character*(*) resname

subroutine NhlFRLGetFloat(list_id, rename, rval, ierr)
integer list_id, ierr
character*(*) resname
real rval

subroutine NhlFRLGetDouble(list_id, rename, dval, ierr)
integer list_id, ierr
character*(*) resname
double precision dval

subroutine NhlFRLGetString(list_id, resname, sval, ierr)
integer list_id, ierr
character*(*) resname, sval
```

Arguments

list_id (input)
: Id of the *ResList* to retrieve the *resname/val_addr* pair from.

resname (input)
: Specifies the resource name to retrieve the value of.

type (input, C only)
: Specifies the type *val_addr* points to.

val_addr (output, C only)
: Specifies the address to place the value retrieved from the *resname* resource when the list is applied to an HLU object using the **NhlGetValues** function.

ival (output, Fortran only)

The integer value retrieved from the *resname* resource when the list is applied to an HLU object using the **NhlFGetValues** function.

rval (output, Fortran only)

The real value retrieved from the *resname* resource when the list is applied to an HLU object using the **NhlFGetValues** function.

dval (output, Fortran only)

The double precision value retrieved from the *resname* resource when the list is applied to an HLU object using the **NhlFGetValues** function.

sval (output, Fortran only)

The string retrieved from the *resname* resource when the list is applied to an HLU object using the **NhlFGetValues** function.

ierr (output, Fortran only)

Error code.

Types

```
Type name:                NhlTErrorTypes
Definition:
typedef enum _NhlErrType{
        NhlFATAL          = -4,    /* "FATAL"        */
        NhlWARNING        = -3,    /* "WARNING"      */
        NhlINFO           = -2,    /* "INFO"         */
        NhlNOERROR        = -1     /* "NOERROR"      */
} NhlErrorTypes;
```

Description

These functions can only be used on a **NhlGETRL** *ResList*. They are used to retrieve *name/val_addr* pairs from the *ResList* that will be passed to the NhlGetValues function.

Return values

The C functions return a value of type NhlErrorTypes, and the Fortran subroutines return the error value in *ierr*. If they are unable to retrive the *name/val_addr* pair from the list, they will return **NhlFATAL**, otherwise they will return **NhlNOERROR**.

See also

- NhlGetValues function

Copyright

Copyright 1987-1999 University Corporation for Atmospheric Research
The use of this Software is governed by a License Agreement.

NCAR Graphics is a registered trademark of the University Corporation for Atmospheric Research.

NhlRLGetArray

NhlRLGetArray, NhlRLGetIntegerArray, NhlRLGetFloatArray, NhlRLGetDoubleArray, NhlRLGetStringArray

The Fortran names of these functions are **NhlFRLGetIntegerArray**, **NhlFRLGetFloatArray**, **NhlFRLGetDoubleArray**, **NhlFRLGetStringArray**. There is no Fortran binding for **NhlRLGetArray**.

Used to add the given *name* and *val_addr* to the *ResList*.

C synopsis

```
#include <ncarg/hlu/hlu.h>

NhlErrorTypes NhlRLGetArray(
                int             list_id,
                NhlString       resname,
                NhlPointer      *data_addr,
                NhlString       *type_element,
                unsigned int    *size_element,
                int             *num_elements
                )

NhlErrorTypes NhlRLGetIntegerArray(
                int             list_id,
                NhlString       resname,
                int             **data_addr,
                int             *num_elements
                )

NhlErrorTypes NhlRLGetFloatArray(
                int             list_id,
                NhlString       resname,
                float           **data_addr,
                int             *num_elements
                )

NhlErrorTypes NhlRLGetDoubleArray(
                int             list_id,
                NhlString       resname,
```

```
                double          **data_addr,
                int             *num_elements
                )

NhlErrorTypes NhlRLGetStringArray(
                int             list_id,
                NhlString       resname,
                NhlString       **data_addr,
                int             *num_elements
                )
```

Fortran synopsis

```
      subroutine NhlFRLGetIntegerArray(list_id, resname, iarray,
     +                                 num_elements, ierr)
      integer list_id, iarray(num_elements), num_elements, ierr
      character*(*) resname

      subroutine NhlFRLGetFloatArray(list_id, resname, rarray,
     +                               num_elements, ierr)
      integer list_id, num_elements, ierr
      real rarray(num_elements)
      character*(*) resname

      subroutine NhlFRLGetDoubleArray(list_id, resname, darray,
     +                               num_elements, ierr)
      integer list_id, num_elements, ierr
      double precision darray(num_elements)
      character*(*) resname

      subroutine NhlFRLGetStringArray(list_id, resname, sarray,
     +                                num_elements, ierr)
      integer list_id, num_elements, ierr
      character*(*) resname, sarray(num_elements)
```

Arguments

list_id (input)
: Id of the *ResList* to retrieve the *name/val_addr* pair from.

resname (input)
: Specifies the resource name to retrieve the value of.

data_addr (output, C only)
: Specifies the address of the pointer to the array retrieved from the *resname* resource when the ResList is applied to an HLU object using the **NhlGetValues** function. The programmers are responsible for freeing this array when they finish with it.

type_element (output, C only)
: Returns the name of the type of each element in the *data* array. This parameter must be a valid pointer to an NhlString. The NhlString will be assigned during the call to **NhlGetValues**, and the programmer is responsible for freeing the NhlString.

size_element (output, C only)

Returns the size, in bytes, of each element in the *data* array. This parameter must be a valid pointer to an unsigned integer.

num_elements (input)
Returns the number of elements in the *data* array. For a C program, this parameter must be a valid pointer to an integer. For a Fortran program, this parameter must contain the number of elements in the *data* array as it is declared in the program.

iarray (output, Fortran only)
The integer array retrieved from the *resname* resource when the list is applied to an HLU object using the **NhlFGetValues** function.

rarray (output, Fortran only)
The real array retrieved from the *resname* resource when the list is applied to an HLU object using the **NhlFGetValues** function.

darray (output, Fortran only)
The double precision array retrieved from the *resname* resource when the list is applied to an HLU object using the **NhlFGetValues** function.

sarray (output, Fortran only)
The string array retrieved from the *resname* resource when the list is applied to an HLU object using the **NhlFGetValues** function.

ierr (output, Fortran only)
Error code.

Types

```
Type name:                NhlTErrorTypes
Definition:
typedef enum _NhlErrType{
        NhlFATAL          = -4,    /* "FATAL"        */
        NhlWARNING        = -3,    /* "WARNING"      */
        NhlINFO           = -2,    /* "INFO"         */
        NhlNOERROR        = -1     /* "NOERROR"      */
} NhlErrorTypes;
```

Description

These functions can only be used on a **NhlGETRL** *ResList*. They are used to retrieve *name/data_addr* pairs from the *ResList* that will be passed to the NhlGetValues function. These functions need to be used to retrieve an array resource from an HLU object. The programmer is responsible for freeing the memory returned to the *data_addr* and *type_element* parameters after the call to **NhlGetValues**.

Return values

The C functions return a value of type NhlErrorTypes, and the Fortran subroutines return the error value in *ierr*. If they are unable to retrieve the *name/value* pair to the list, they will return **NhlFATAL**;

otherwise they will return **NhlNOERROR**.

See also

- NhlGetValues function

Copyright

Copyright 1987-1999 University Corporation for Atmospheric Research
The use of this Software is governed by a License Agreement.

NCAR Graphics is a registered trademark of the University Corporation for Atmospheric Research.

NhlRLGet

NhlRLGet, NhlRLGetInteger, NhlRLGetFloat, NhlRLGetDouble, NhlRLGetString

The Fortran names of these functions are **NhlFRLGetInteger**, **NhlFRLGetFloat**, **NhlFRLGetDouble**, **NhlFRLGetString**. There is no Fortran binding for **NhlRLGet**.

Used to add the given *name* and *val_addr* to the *ResList*.

C synopsis

```
#include <ncarg/hlu/hlu.h>

NhlErrorTypes NhlRLGet(
                 int             list_id,
                 NhlString       resname,
                 NhlString       type,
                 <AnyType>       *val_addr
                 )

NhlErrorTypes NhlRLGetInteger(
                 int             list_id,
                 NhlString       resname,
                 int             *val_addr
                 )

NhlErrorTypes NhlRLGetFloat(
                 int             list_id,
```

```
                NhlString       resname,
                float           *val_addr
                )

NhlErrorTypes NhlRLGetDouble(
                int             list_id,
                NhlString       resname,
                double          *val_addr
                )

NhlErrorTypes NhlRLGetString(
                int             list_id,
                NhlString       resname,
                NhlString       *val_addr
                )
```

Fortran synopsis

```
      subroutine NhlFRLGetInteger(list_id, resname, ival, ierr)
      integer list_id, ival, ierr
      character*(*) resname

      subroutine NhlFRLGetFloat(list_id, rename, rval, ierr)
      integer list_id, ierr
      character*(*) resname
      real rval

      subroutine NhlFRLGetDouble(list_id, rename, dval, ierr)
      integer list_id, ierr
      character*(*) resname
      double precision dval

      subroutine NhlFRLGetString(list_id, resname, sval, ierr)
      integer list_id, ierr
      character*(*) resname, sval
```

Arguments

list_id (input)
: Id of the *ResList* to retrieve the *resname/val_addr* pair from.

resname (input)
: Specifies the resource name to retrieve the value of.

type (input, C only)
: Specifies the type *val_addr* points to.

val_addr (output, C only)
: Specifies the address to place the value retrieved from the *resname* resource when the list is applied to an HLU object using the **NhlGetValues** function.

ival (output, Fortran only)
: The integer value retrieved from the *resname* resource when the list is applied to an HLU object using the **NhlFGetValues** function.

rval (output, Fortran only)
: The real value retrieved from the *resname* resource when the list is applied to an HLU object using the **NhlFGetValues** function.

dval (output, Fortran only)
: The double precision value retrieved from the *resname* resource when the list is applied to an HLU object using the **NhlFGetValues** function.

sval (output, Fortran only)
: The string retrieved from the *resname* resource when the list is applied to an HLU object using the **NhlFGetValues** function.

ierr (output, Fortran only)
: Error code.

Types

```
Type name:                NhlTErrorTypes
Definition:
typedef enum _NhlErrType{
        NhlFATAL        = -4,    /* "FATAL"       */
        NhlWARNING      = -3,    /* "WARNING"     */
        NhlINFO         = -2,    /* "INFO"        */
        NhlNOERROR      = -1     /* "NOERROR"     */
} NhlErrorTypes;
```

Description

These functions can only be used on a **NhlGETRL** *ResList*. They are used to retrieve *name/val_addr* pairs from the *ResList* that will be passed to the NhlGetValues function.

Return values

The C functions return a value of type NhlErrorTypes, and the Fortran subroutines return the error value in *ierr*. If they are unable to retrive the *name/val_addr* pair from the list, they will return **NhlFATAL**, otherwise they will return **NhlNOERROR**.

See also

- NhlGetValues function

Copyright

Copyright 1987-1999 University Corporation for Atmospheric Research
The use of this Software is governed by a License Agreement.

NCAR Graphics is a registered trademark of the University Corporation for Atmospheric Research.

NhlRLGetArray

NhlRLGetArray, NhlRLGetIntegerArray, NhlRLGetFloatArray, NhlRLGetDoubleArray, NhlRLGetStringArray

The Fortran names of these functions are **NhlFRLGetIntegerArray**, **NhlFRLGetFloatArray**, **NhlFRLGetDoubleArray**, **NhlFRLGetStringArray**. There is no Fortran binding for **NhlRLGetArray**.

Used to add the given *name* and *val_addr* to the *ResList*.

C synopsis

```
#include <ncarg/hlu/hlu.h>

NhlErrorTypes NhlRLGetArray(
                int             list_id,
                NhlString       resname,
                NhlPointer      *data_addr,
                NhlString       *type_element,
                unsigned int    *size_element,
                int             *num_elements
                )

NhlErrorTypes NhlRLGetIntegerArray(
                int             list_id,
                NhlString       resname,
                int             **data_addr,
                int             *num_elements
                )

NhlErrorTypes NhlRLGetFloatArray(
                int             list_id,
                NhlString       resname,
                float           **data_addr,
                int             *num_elements
                )

NhlErrorTypes NhlRLGetDoubleArray(
                int             list_id,
                NhlString       resname,
                double          **data_addr,
                int             *num_elements
```

```
                )

NhlErrorTypes NhlRLGetStringArray(
                int             list_id,
                NhlString       resname,
                NhlString       **data_addr,
                int             *num_elements
                )
```

Fortran synopsis

```
      subroutine NhlFRLGetIntegerArray(list_id, resname, iarray,
     +                                 num_elements, ierr)
      integer list_id, iarray(num_elements), num_elements, ierr
      character*(*) resname

      subroutine NhlFRLGetFloatArray(list_id, resname, rarray,
     +                               num_elements, ierr)
      integer list_id, num_elements, ierr
      real rarray(num_elements)
      character*(*) resname

      subroutine NhlFRLGetDoubleArray(list_id, resname, darray,
     +                               num_elements, ierr)
      integer list_id, num_elements, ierr
      double precision darray(num_elements)
      character*(*) resname

      subroutine NhlFRLGetStringArray(list_id, resname, sarray,
     +                                num_elements, ierr)
      integer list_id, num_elements, ierr
      character*(*) resname, sarray(num_elements)
```

Arguments

list_id (input)
: Id of the *ResList* to retrieve the *name/val_addr* pair from.

resname (input)
: Specifies the resource name to retrieve the value of.

data_addr (output, C only)
: Specifies the address of the pointer to the array retrieved from the *resname* resource when the ResList is applied to an HLU object using the **NhlGetValues** function. The programmers are responsible for freeing this array when they finish with it.

type_element (output, C only)
: Returns the name of the type of each element in the *data* array. This parameter must be a valid pointer to an NhlString. The NhlString will be assigned during the call to **NhlGetValues**, and the programmer is responsible for freeing the NhlString.

size_element (output, C only)
: Returns the size, in bytes, of each element in the *data* array. This parameter must be a valid pointer

to an unsigned integer.

num_elements (input)

Returns the number of elements in the *data* array. For a C program, this parameter must be a valid pointer to an integer. For a Fortran program, this parameter must contain the number of elements in the *data* array as it is declared in the program.

iarray (output, Fortran only)

The integer array retrieved from the *resname* resource when the list is applied to an HLU object using the **NhlFGetValues** function.

rarray (output, Fortran only)

The real array retrieved from the *resname* resource when the list is applied to an HLU object using the **NhlFGetValues** function.

darray (output, Fortran only)

The double precision array retrieved from the *resname* resource when the list is applied to an HLU object using the **NhlFGetValues** function.

sarray (output, Fortran only)

The string array retrieved from the *resname* resource when the list is applied to an HLU object using the **NhlFGetValues** function.

ierr (output, Fortran only)

Error code.

Types

```
Type name:                NhlTErrorTypes
Definition:
typedef enum _NhlErrType{
        NhlFATAL        = -4,    /* "FATAL"       */
        NhlWARNING      = -3,    /* "WARNING"     */
        NhlINFO         = -2,    /* "INFO"        */
        NhlNOERROR      = -1     /* "NOERROR"     */
} NhlErrorTypes;
```

Description

These functions can only be used on a **NhlGETRL** *ResList*. They are used to retrieve *name/data_addr* pairs from the *ResList* that will be passed to the NhlGetValues function. These functions need to be used to retrieve an array resource from an HLU object. The programmer is responsible for freeing the memory returned to the *data_addr* and *type_element* parameters after the call to **NhlGetValues**.

Return values

The C functions return a value of type NhlErrorTypes, and the Fortran subroutines return the error value in *ierr*. If they are unable to retrieve the *name/value* pair to the list, they will return **NhlFATAL**; otherwise they will return **NhlNOERROR**.

See also

- NhlGetValues function

Copyright

Copyright 1987-1999 University Corporation for Atmospheric Research
The use of this Software is governed by a License Agreement.

NCAR Graphics is a registered trademark of the University Corporation for Atmospheric Research.

NhlRLGet

NhlRLGet, NhlRLGetInteger, NhlRLGetFloat, NhlRLGetDouble, NhlRLGetString

The Fortran names of these functions are **NhlFRLGetInteger**, **NhlFRLGetFloat**, **NhlFRLGetDouble**, **NhlFRLGetString**. There is no Fortran binding for **NhlRLGet**.

Used to add the given *name* and *val_addr* to the *ResList*.

C synopsis

```
#include <ncarg/hlu/hlu.h>

NhlErrorTypes NhlRLGet(
                int             list_id,
                NhlString       resname,
                NhlString       type,
                <AnyType>       *val_addr
                )

NhlErrorTypes NhlRLGetInteger(
                int             list_id,
                NhlString       resname,
                int             *val_addr
                )

NhlErrorTypes NhlRLGetFloat(
                int             list_id,
                NhlString       resname,
                float           *val_addr
```

```
                 )

NhlErrorTypes NhlRLGetDouble(
                 int              list_id,
                 NhlString        resname,
                 double           *val_addr
                 )

NhlErrorTypes NhlRLGetString(
                 int              list_id,
                 NhlString        resname,
                 NhlString        *val_addr
                 )
```

Fortran synopsis

```
subroutine NhlFRLGetInteger(list_id, resname, ival, ierr)
integer list_id, ival, ierr
character*(*) resname

subroutine NhlFRLGetFloat(list_id, rename, rval, ierr)
integer list_id, ierr
character*(*) resname
real rval

subroutine NhlFRLGetDouble(list_id, rename, dval, ierr)
integer list_id, ierr
character*(*) resname
double precision dval

subroutine NhlFRLGetString(list_id, resname, sval, ierr)
integer list_id, ierr
character*(*) resname, sval
```

Arguments

list_id (input)
: Id of the *ResList* to retrieve the *resname/val_addr* pair from.

resname (input)
: Specifies the resource name to retrieve the value of.

type (input, C only)
: Specifies the type *val_addr* points to.

val_addr (output, C only)
: Specifies the address to place the value retrieved from the *resname* resource when the list is applied to an HLU object using the **NhlGetValues** function.

ival (output, Fortran only)
: The integer value retrieved from the *resname* resource when the list is applied to an HLU object using the **NhlFGetValues** function.

rval (output, Fortran only)
: The real value retrieved from the *resname* resource when the list is applied to an HLU object using

the **NhlFGetValues** function.

dval (output, Fortran only)

The double precision value retrieved from the *resname* resource when the list is applied to an HLU object using the **NhlFGetValues** function.

sval (output, Fortran only)

The string retrieved from the *resname* resource when the list is applied to an HLU object using the **NhlFGetValues** function.

ierr (output, Fortran only)

Error code.

Types

```
Type name:              NhlTErrorTypes
Definition:
typedef enum _NhlErrType{
        NhlFATAL        = -4,   /* "FATAL"      */
        NhlWARNING      = -3,   /* "WARNING"    */
        NhlINFO         = -2,   /* "INFO"       */
        NhlNOERROR      = -1    /* "NOERROR"    */
} NhlErrorTypes;
```

Description

These functions can only be used on a **NhlGETRL** *ResList*. They are used to retrieve *name/val_addr* pairs from the *ResList* that will be passed to the NhlGetValues function.

Return values

The C functions return a value of type NhlErrorTypes, and the Fortran subroutines return the error value in *ierr*. If they are unable to retrive the *name/val_addr* pair from the list, they will return **NhlFATAL**, otherwise they will return **NhlNOERROR**.

See also

- NhlGetValues function

Copyright

Copyright 1987-1999 University Corporation for Atmospheric Research

The use of this Software is governed by a License Agreement.

NCAR Graphics is a registered trademark of the University Corporation for Atmospheric Research.

NhlRLGetArray

NhlRLGetArray, NhlRLGetIntegerArray, NhlRLGetFloatArray, NhlRLGetDoubleArray, NhlRLGetStringArray

The Fortran names of these functions are **NhlFRLGetIntegerArray**, **NhlFRLGetFloatArray**, **NhlFRLGetDoubleArray**, **NhlFRLGetStringArray**. There is no Fortran binding for **NhlRLGetArray**.

Used to add the given *name* and *val_addr* to the *ResList*.

C synopsis

```
#include <ncarg/hlu/hlu.h>

NhlErrorTypes NhlRLGetArray(
                int              list_id,
                NhlString        resname,
                NhlPointer       *data_addr,
                NhlString        *type_element,
                unsigned int     *size_element,
                int              *num_elements
                )

NhlErrorTypes NhlRLGetIntegerArray(
                int              list_id,
                NhlString        resname,
                int              **data_addr,
                int              *num_elements
                )

NhlErrorTypes NhlRLGetFloatArray(
                int              list_id,
                NhlString        resname,
                float            **data_addr,
                int              *num_elements
                )

NhlErrorTypes NhlRLGetDoubleArray(
                int              list_id,
                NhlString        resname,
                double           **data_addr,
                int              *num_elements
                )
```

```
NhlErrorTypes NhlRLGetStringArray(
                int             list_id,
                NhlString       resname,
                NhlString       **data_addr,
                int             *num_elements
                )
```

Fortran synopsis

```
      subroutine NhlFRLGetIntegerArray(list_id, resname, iarray,
     +                                 num_elements, ierr)
      integer list_id, iarray(num_elements), num_elements, ierr
      character*(*) resname

      subroutine NhlFRLGetFloatArray(list_id, resname, rarray,
     +                               num_elements, ierr)
      integer list_id, num_elements, ierr
      real rarray(num_elements)
      character*(*) resname

      subroutine NhlFRLGetDoubleArray(list_id, resname, darray,
     +                               num_elements, ierr)
      integer list_id, num_elements, ierr
      double precision darray(num_elements)
      character*(*) resname

      subroutine NhlFRLGetStringArray(list_id, resname, sarray,
     +                                num_elements, ierr)
      integer list_id, num_elements, ierr
      character*(*) resname, sarray(num_elements)
```

Arguments

list_id (input)
: Id of the *ResList* to retrieve the *name/val_addr* pair from.

resname (input)
: Specifies the resource name to retrieve the value of.

data_addr (output, C only)
: Specifies the address of the pointer to the array retrieved from the *resname* resource when the ResList is applied to an HLU object using the **NhlGetValues** function. The programmers are responsible for freeing this array when they finish with it.

type_element (output, C only)
: Returns the name of the type of each element in the *data* array. This parameter must be a valid pointer to an NhlString. The NhlString will be assigned during the call to **NhlGetValues**, and the programmer is responsible for freeing the NhlString.

size_element (output, C only)
: Returns the size, in bytes, of each element in the *data* array. This parameter must be a valid pointer to an unsigned integer.

num_elements (input)

Returns the number of elements in the *data* array. For a C program, this parameter must be a valid pointer to an integer. For a Fortran program, this parameter must contain the number of elements in the *data* array as it is declared in the program.

iarray (output, Fortran only)
The integer array retrieved from the *resname* resource when the list is applied to an HLU object using the **NhlFGetValues** function.

rarray (output, Fortran only)
The real array retrieved from the *resname* resource when the list is applied to an HLU object using the **NhlFGetValues** function.

darray (output, Fortran only)
The double precision array retrieved from the *resname* resource when the list is applied to an HLU object using the **NhlFGetValues** function.

sarray (output, Fortran only)
The string array retrieved from the *resname* resource when the list is applied to an HLU object using the **NhlFGetValues** function.

ierr (output, Fortran only)
Error code.

Types

```
Type name:                NhlTErrorTypes
Definition:
typedef enum _NhlErrType{
        NhlFATAL        = -4,    /* "FATAL"       */
        NhlWARNING      = -3,    /* "WARNING"     */
        NhlINFO         = -2,    /* "INFO"        */
        NhlNOERROR      = -1     /* "NOERROR"     */
} NhlErrorTypes;
```

Description

These functions can only be used on a **NhlGETRL** *ResList*. They are used to retrieve *name/data_addr* pairs from the *ResList* that will be passed to the NhlGetValues function. These functions need to be used to retrieve an array resource from an HLU object. The programmer is responsible for freeing the memory returned to the *data_addr* and *type_element* parameters after the call to **NhlGetValues**.

Return values

The C functions return a value of type NhlErrorTypes, and the Fortran subroutines return the error value in *ierr*. If they are unable to retrieve the *name/value* pair to the list, they will return **NhlFATAL**; otherwise they will return **NhlNOERROR**.

See also

- NhlGetValues function

Copyright

Copyright 1987-1999 University Corporation for Atmospheric Research
The use of this Software is governed by a License Agreement.

NCAR Graphics is a registered trademark of the University Corporation for Atmospheric Research.

NhlRLGet

NhlRLGet, NhlRLGetInteger, NhlRLGetFloat, NhlRLGetDouble, NhlRLGetString

The Fortran names of these functions are **NhlFRLGetInteger**, **NhlFRLGetFloat**, **NhlFRLGetDouble**, **NhlFRLGetString**. There is no Fortran binding for **NhlRLGet**.

Used to add the given *name* and *val_addr* to the *ResList*.

C synopsis

```
#include <ncarg/hlu/hlu.h>

NhlErrorTypes NhlRLGet(
                int             list_id,
                NhlString       resname,
                NhlString       type,
                <AnyType>       *val_addr
                )

NhlErrorTypes NhlRLGetInteger(
                int             list_id,
                NhlString       resname,
                int             *val_addr
                )

NhlErrorTypes NhlRLGetFloat(
                int             list_id,
                NhlString       resname,
                float           *val_addr
                )
```

```
NhlErrorTypes NhlRLGetDouble(
                int             list_id,
                NhlString       resname,
                double          *val_addr
                )

NhlErrorTypes NhlRLGetString(
                int             list_id,
                NhlString       resname,
                NhlString       *val_addr
                )
```

Fortran synopsis

```
subroutine NhlFRLGetInteger(list_id, resname, ival, ierr)
integer list_id, ival, ierr
character*(*) resname

subroutine NhlFRLGetFloat(list_id, rename, rval, ierr)
integer list_id, ierr
character*(*) resname
real rval

subroutine NhlFRLGetDouble(list_id, rename, dval, ierr)
integer list_id, ierr
character*(*) resname
double precision dval

subroutine NhlFRLGetString(list_id, resname, sval, ierr)
integer list_id, ierr
character*(*) resname, sval
```

Arguments

list_id (input)
: Id of the *ResList* to retrieve the *resname/val_addr* pair from.

resname (input)
: Specifies the resource name to retrieve the value of.

type (input, C only)
: Specifies the type *val_addr* points to.

val_addr (output, C only)
: Specifies the address to place the value retrieved from the *resname* resource when the list is applied to an HLU object using the **NhlGetValues** function.

ival (output, Fortran only)
: The integer value retrieved from the *resname* resource when the list is applied to an HLU object using the **NhlFGetValues** function.

rval (output, Fortran only)
: The real value retrieved from the *resname* resource when the list is applied to an HLU object using the **NhlFGetValues** function.

dval (output, Fortran only)
The double precision value retrieved from the *resname* resource when the list is applied to an HLU object using the **NhlFGetValues** function.

sval (output, Fortran only)
The string retrieved from the *resname* resource when the list is applied to an HLU object using the **NhlFGetValues** function.

ierr (output, Fortran only)
Error code.

Types

```
Type name:                NhlTErrorTypes
Definition:
typedef enum _NhlErrType{
        NhlFATAL        = -4,    /* "FATAL"      */
        NhlWARNING      = -3,    /* "WARNING"    */
        NhlINFO         = -2,    /* "INFO"       */
        NhlNOERROR      = -1     /* "NOERROR"    */
} NhlErrorTypes;
```

Description

These functions can only be used on a **NhlGETRL** *ResList*. They are used to retrieve *name/val_addr* pairs from the *ResList* that will be passed to the NhlGetValues function.

Return values

The C functions return a value of type NhlErrorTypes, and the Fortran subroutines return the error value in *ierr*. If they are unable to retrive the *name/val_addr* pair from the list, they will return **NhlFATAL**, otherwise they will return **NhlNOERROR**.

See also

- NhlGetValues function

Copyright

Copyright 1987-1999 University Corporation for Atmospheric Research
The use of this Software is governed by a License Agreement.

NCAR Graphics is a registered trademark of the University Corporation for Atmospheric Research.

NhlRLGetArray

NhlRLGetArray, NhlRLGetIntegerArray, NhlRLGetFloatArray, NhlRLGetDoubleArray, NhlRLGetStringArray

The Fortran names of these functions are **NhlFRLGetIntegerArray**, **NhlFRLGetFloatArray**, **NhlFRLGetDoubleArray**, **NhlFRLGetStringArray**. There is no Fortran binding for **NhlRLGetArray**.

Used to add the given *name* and *val_addr* to the *ResList*.

C synopsis

```
#include <ncarg/hlu/hlu.h>

NhlErrorTypes NhlRLGetArray(
                int             list_id,
                NhlString       resname,
                NhlPointer      *data_addr,
                NhlString       *type_element,
                unsigned int    *size_element,
                int             *num_elements
                )

NhlErrorTypes NhlRLGetIntegerArray(
                int             list_id,
                NhlString       resname,
                int             **data_addr,
                int             *num_elements
                )

NhlErrorTypes NhlRLGetFloatArray(
                int             list_id,
                NhlString       resname,
                float           **data_addr,
                int             *num_elements
                )

NhlErrorTypes NhlRLGetDoubleArray(
                int             list_id,
                NhlString       resname,
                double          **data_addr,
                int             *num_elements
                )

NhlErrorTypes NhlRLGetStringArray(
                int             list_id,
```

```
                NhlString       resname,
                NhlString       **data_addr,
                int             *num_elements
                )
```

Fortran synopsis

```
      subroutine NhlFRLGetIntegerArray(list_id, resname, iarray,
     +                                  num_elements, ierr)
      integer list_id, iarray(num_elements), num_elements, ierr
      character*(*) resname

      subroutine NhlFRLGetFloatArray(list_id, resname, rarray,
     +                                num_elements, ierr)
      integer list_id, num_elements, ierr
      real rarray(num_elements)
      character*(*) resname

      subroutine NhlFRLGetDoubleArray(list_id, resname, darray,
     +                                num_elements, ierr)
      integer list_id, num_elements, ierr
      double precision darray(num_elements)
      character*(*) resname

      subroutine NhlFRLGetStringArray(list_id, resname, sarray,
     +                                 num_elements, ierr)
      integer list_id, num_elements, ierr
      character*(*) resname, sarray(num_elements)
```

Arguments

list_id (input)
: Id of the *ResList* to retrieve the *name/val_addr* pair from.

resname (input)
: Specifies the resource name to retrieve the value of.

data_addr (output, C only)
: Specifies the address of the pointer to the array retrieved from the *resname* resource when the ResList is applied to an HLU object using the **NhlGetValues** function. The programmers are responsible for freeing this array when they finish with it.

type_element (output, C only)
: Returns the name of the type of each element in the *data* array. This parameter must be a valid pointer to an NhlString. The NhlString will be assigned during the call to **NhlGetValues**, and the programmer is responsible for freeing the NhlString.

size_element (output, C only)
: Returns the size, in bytes, of each element in the *data* array. This parameter must be a valid pointer to an unsigned integer.

num_elements (input)
: Returns the number of elements in the *data* array. For a C program, this parameter must be a valid pointer to an integer. For a Fortran program, this parameter must contain the number of elements

in the *data* array as it is declared in the program.

iarray (output, Fortran only)
: The integer array retrieved from the *resname* resource when the list is applied to an HLU object using the **NhlFGetValues** function.

rarray (output, Fortran only)
: The real array retrieved from the *resname* resource when the list is applied to an HLU object using the **NhlFGetValues** function.

darray (output, Fortran only)
: The double precision array retrieved from the *resname* resource when the list is applied to an HLU object using the **NhlFGetValues** function.

sarray (output, Fortran only)
: The string array retrieved from the *resname* resource when the list is applied to an HLU object using the **NhlFGetValues** function.

ierr (output, Fortran only)
: Error code.

Types

```
Type name:                NhlTErrorTypes
Definition:
typedef enum _NhlErrType{
        NhlFATAL          = -4,    /* "FATAL"       */
        NhlWARNING        = -3,    /* "WARNING"     */
        NhlINFO           = -2,    /* "INFO"        */
        NhlNOERROR        = -1     /* "NOERROR"     */
} NhlErrorTypes;
```

Description

These functions can only be used on a **NhlGETRL** *ResList*. They are used to retrieve *name/data_addr* pairs from the *ResList* that will be passed to the NhlGetValues function. These functions need to be used to retrieve an array resource from an HLU object. The programmer is responsible for freeing the memory returned to the *data_addr* and *type_element* parameters after the call to **NhlGetValues**.

Return values

The C functions return a value of type NhlErrorTypes, and the Fortran subroutines return the error value in *ierr*. If they are unable to retrieve the *name/value* pair to the list, they will return **NhlFATAL**; otherwise they will return **NhlNOERROR**.

See also

- NhlGetValues function

Copyright

Copyright 1987-1999 University Corporation for Atmospheric Research
The use of this Software is governed by a License Agreement.

NCAR Graphics is a registered trademark of the University Corporation for Atmospheric Research.

NhlRLGetMDArray

NhlRLGetMDArray, NhlRLGetMDIntegerArray, NhlRLGetMDFloatArray, NhlRLGetMDDoubleArray

The Fortran names of these functions are **NhlFRLGetMDIntegerArray**, **NhlFRLGetMDFloatArray**, **NhlFRLGetMDDoubleArray**. There is no Fortran binding for **NhlRLGetMDArray**.

Used to add the given *name* and *val_addr* to the *ResList*.

C synopsis

```
#include <ncarg/hlu/hlu.h>

NhlErrorTypes NhlRLGetMDArray(
                int             list_id,
                NhlString       resname,
                NhlPointer      *data_addr,
                NhlString       *type_element,
                unsigned int    *size_element,
                int             *num_dimensions,
                int             **len_dimensions
                )

NhlErrorTypes NhlRLGetMDIntegerArray(
                int             list_id,
                NhlString       resname,
                int             **data_addr,
                int             *num_dimensions,
                int             **len_dimensions
                )

NhlErrorTypes NhlRLGetMDFloatArray(
                int             list_id,
                NhlString       resname,
                float           **data_addr,
```

```
                int             *num_dimensions,
                int             **len_dimensions
                )

NhlErrorTypes NhlRLGetMDDoubleArray(
                int             list_id,
                NhlString       resname,
                double          **data_addr,
                int             *num_dimensions,
                int             **len_dimensions
                )
```

Fortran synopsis

```
      subroutine NhlFRLGetMDIntegerArray(list_id, resname, iarray,
     +                   num_dim, len_dim, ierr)
      integer list_id, iarray(*), num_dim, len_dim, ierr
      character*(*) resname

      subroutine NhlFRLGetMDFloatArray(list_id, resname, rarray,
     +                   num_dim, len_dim, ierr)
      integer list_id, num_dim, len_dim, ierr
      real rarray(*)
      character*(*) resname

      subroutine NhlFRLGetMDDoubleArray(list_id, resname, darray,
     +                   num_dim, len_dim, ierr)
      integer list_id, num_dim, len_dim, ierr
      double precision darray(*)
      character*(*) resname
```

Arguments

list_id (input)
: Id of the *ResList* to retrieve the *name/val_addr* pair from.

resname (input)
: Specifies the resource name to retrieve the value of.

data_addr (output, C only)
: Specifies the address of the pointer to the array retrieved from the *resname* resource when the ResList is applied to an HLU object using the **NhlGetValues** function. The programmers are responsible for freeing this array when they finish with it.

type_element (output, C only)
: Returns the name of the type of each element in the *data* array. This parameter must be a valid pointer to an NhlString. The NhlString will be assigned during the call to **NhlGetValues**, and the programmer is responsible for freeing the NhlString returned.

size_element (output, C only)
: Returns the size, in bytes, of each element in the *data* array. This parameter must be a valid pointer to an unsigned integer.

num_dimensions (output, C only)

Returns the number of dimensions in the *data* array. This parameter must be a valid pointer to an integer.

len_dimensions (output, C only)

Returns an array of length *num_dimensions*. Specifies the length of each dimension in the *data* array. The array will be allocated during the call to **NhlGetValues**, and the programmer is responsible for freeing the array returned.

num_dim (input, Fortran only)

This parameter must contain the number of dimensions in the data array as it is declared in the program (*iarray* or *rarray*). It is also the number of elements in the *len_dim* array. If the resource that is being returned has fewer dimensions than the data array, this value will be changed to indicate the number of dimensions that were returned.

len_dim (input, Fortran only)

This array must contain the length of each dimension in the data array as it is declared in the program (*iarray* or *rarray*). If the resource that is being returned has fewer dimensions than the data array, the *num_dim* variable will indicate how many dimensions were returned, and therefore the number of elements in this array that have been changed to indicate the length of each dimension that was returned. The array will be modified during the call to **NhlFGetValues**.

iarray (output, Fortran only)

The multi-dimensioned integer array retrieved from the *resname* resource when the list is applied to an HLU object using the **NhlFGetValues** function.

rarray (output, Fortran only)

The multi-dimensioned real array retrieved from the *resname* resource when the list is applied to an HLU object using the **NhlFGetValues** function.

darray (output, Fortran only)

The multi-dimensioned double precision array retrieved from the *resname* resource when the list is applied to an HLU object using the **NhlFGetValues** function.

ierr (output, Fortran only)

Error code.

Types

```
Type name:                NhlTErrorTypes
Definition:
typedef enum _NhlErrType{
        NhlFATAL        = -4,    /* "FATAL"       */
        NhlWARNING      = -3,    /* "WARNING"     */
        NhlINFO         = -2,    /* "INFO"        */
        NhlNOERROR      = -1     /* "NOERROR"     */
} NhlErrorTypes;
```

Description

These functions can only be used on a **NhlGETRL** *ResList*. They are used to retrieve *name/data_addr* pairs from the *ResList* that will be passed to the NhlGetValues function. These functions need to be used to retrieve an Array resource from an HLU object. The programmer is responsible for freeing the memory returned to the *data_addr*, *type_element*, and *len_dimensions* parameters after the call to **NhlGetValues**.

Return values

The C functions return a value of type NhlErrorTypes, and the Fortran subroutines return the error value in *ierr*. If they are unable to retrieve the *name/value* pair to the list, they will return **NhlFATAL**; otherwise they will return **NhlNOERROR**.

See also

- NhlGetValues function

Copyright

Copyright 1987-1999 University Corporation for Atmospheric Research
The use of this Software is governed by a License Agreement.

NCAR Graphics is a registered trademark of the University Corporation for Atmospheric Research.

NhlRLGetMDArray

NhlRLGetMDArray, NhlRLGetMDIntegerArray, NhlRLGetMDFloatArray, NhlRLGetMDDoubleArray

The Fortran names of these functions are **NhlFRLGetMDIntegerArray**, **NhlFRLGetMDFloatArray**, **NhlFRLGetMDDoubleArray**. There is no Fortran binding for **NhlRLGetMDArray**.

Used to add the given *name* and *val_addr* to the *ResList*.

C synopsis

```
#include <ncarg/hlu/hlu.h>

NhlErrorTypes NhlRLGetMDArray(
                int             list_id,
                NhlString       resname,
                NhlPointer      *data_addr,
```

```
                NhlString       *type_element,
                unsigned int    *size_element,
                int             *num_dimensions,
                int             **len_dimensions
                )

NhlErrorTypes NhlRLGetMDIntegerArray(
                int             list_id,
                NhlString       resname,
                int             **data_addr,
                int             *num_dimensions,
                int             **len_dimensions
                )

NhlErrorTypes NhlRLGetMDFloatArray(
                int             list_id,
                NhlString       resname,
                float           **data_addr,
                int             *num_dimensions,
                int             **len_dimensions
                )

NhlErrorTypes NhlRLGetMDDoubleArray(
                int             list_id,
                NhlString       resname,
                double          **data_addr,
                int             *num_dimensions,
                int             **len_dimensions
                )
```

Fortran synopsis

```
      subroutine NhlFRLGetMDIntegerArray(list_id, resname, iarray,
     +                   num_dim, len_dim, ierr)
      integer list_id, iarray(*), num_dim, len_dim, ierr
      character*(*) resname

      subroutine NhlFRLGetMDFloatArray(list_id, resname, rarray,
     +                   num_dim, len_dim, ierr)
      integer list_id, num_dim, len_dim, ierr
      real rarray(*)
      character*(*) resname

      subroutine NhlFRLGetMDDoubleArray(list_id, resname, darray,
     +                   num_dim, len_dim, ierr)
      integer list_id, num_dim, len_dim, ierr
      double precision darray(*)
      character*(*) resname
```

Arguments

list_id (input)
: Id of the *ResList* to retrieve the *name/val_addr* pair from.

resname (input)
Specifies the resource name to retrieve the value of.

data_addr (output, C only)
Specifies the address of the pointer to the array retrieved from the *resname* resource when the ResList is applied to an HLU object using the **NhlGetValues** function. The programmers are responsible for freeing this array when they finish with it.

type_element (output, C only)
Returns the name of the type of each element in the *data* array. This parameter must be a valid pointer to an NhlString. The NhlString will be assigned during the call to **NhlGetValues**, and the programmer is responsible for freeing the NhlString returned.

size_element (output, C only)
Returns the size, in bytes, of each element in the *data* array. This parameter must be a valid pointer to an unsigned integer.

num_dimensions (output, C only)
Returns the number of dimensions in the *data* array. This parameter must be a valid pointer to an integer.

len_dimensions (output, C only)
Returns an array of length *num_dimensions*. Specifies the length of each dimension in the *data* array. The array will be allocated during the call to **NhlGetValues**, and the programmer is responsible for freeing the array returned.

num_dim (input, Fortran only)
This parameter must contain the number of dimensions in the data array as it is declared in the program (*iarray* or *rarray*). It is also the number of elements in the *len_dim* array. If the resource that is being returned has fewer dimensions than the data array, this value will be changed to indicate the number of dimensions that were returned.

len_dim (input, Fortran only)
This array must contain the length of each dimension in the data array as it is declared in the program (*iarray* or *rarray*). If the resource that is being returned has fewer dimensions than the data array, the *num_dim* variable will indicate how many dimensions were returned, and therefore the number of elements in this array that have been changed to indicate the length of each dimension that was returned. The array will be modified during the call to **NhlFGetValues**.

iarray (output, Fortran only)
The multi-dimensioned integer array retrieved from the *resname* resource when the list is applied to an HLU object using the **NhlFGetValues** function.

rarray (output, Fortran only)
The multi-dimensioned real array retrieved from the *resname* resource when the list is applied to an HLU object using the **NhlFGetValues** function.

darray (output, Fortran only)
The multi-dimensioned double precision array retrieved from the *resname* resource when the list is applied to an HLU object using the **NhlFGetValues** function.

ierr (output, Fortran only)
Error code.

Types

```
Type name:                NhlTErrorTypes
Definition:
typedef enum _NhlErrType{
        NhlFATAL          = -4,    /* "FATAL"       */
        NhlWARNING        = -3,    /* "WARNING"     */
        NhlINFO           = -2,    /* "INFO"        */
        NhlNOERROR        = -1     /* "NOERROR"     */
} NhlErrorTypes;
```

Description

These functions can only be used on a **NhlGETRL** *ResList*. They are used to retrieve *name/data_addr* pairs from the *ResList* that will be passed to the NhlGetValues function. These functions need to be used to retrieve an Array resource from an HLU object. The programmer is responsible for freeing the memory returned to the *data_addr*, *type_element*, and *len_dimensions* parameters after the call to **NhlGetValues**.

Return values

The C functions return a value of type NhlErrorTypes, and the Fortran subroutines return the error value in *ierr*. If they are unable to retrieve the *name/value* pair to the list, they will return **NhlFATAL**; otherwise they will return **NhlNOERROR**.

See also

- NhlGetValues function

Copyright

Copyright 1987-1999 University Corporation for Atmospheric Research
The use of this Software is governed by a License Agreement.

NCAR Graphics is a registered trademark of the University Corporation for Atmospheric Research.

NhlRLGetMDArray

NhlRLGetMDArray, NhlRLGetMDIntegerArray, NhlRLGetMDFloatArray, NhlRLGetMDDoubleArray

The Fortran names of these functions are **NhlFRLGetMDIntegerArray**, **NhlFRLGetMDFloatArray**, **NhlFRLGetMDDoubleArray**. There is no Fortran binding for **NhlRLGetMDArray**.

Used to add the given *name* and *val_addr* to the *ResList*.

C synopsis

```
#include <ncarg/hlu/hlu.h>

NhlErrorTypes NhlRLGetMDArray(
                int             list_id,
                NhlString       resname,
                NhlPointer      *data_addr,
                NhlString       *type_element,
                unsigned int    *size_element,
                int             *num_dimensions,
                int             **len_dimensions
                )

NhlErrorTypes NhlRLGetMDIntegerArray(
                int             list_id,
                NhlString       resname,
                int             **data_addr,
                int             *num_dimensions,
                int             **len_dimensions
                )

NhlErrorTypes NhlRLGetMDFloatArray(
                int             list_id,
                NhlString       resname,
                float           **data_addr,
                int             *num_dimensions,
                int             **len_dimensions
                )

NhlErrorTypes NhlRLGetMDDoubleArray(
                int             list_id,
                NhlString       resname,
                double          **data_addr,
                int             *num_dimensions,
                int             **len_dimensions
                )
```

Fortran synopsis

```
      subroutine NhlFRLGetMDIntegerArray(list_id, resname, iarray,
     +                   num_dim, len_dim, ierr)
      integer list_id, iarray(*), num_dim, len_dim, ierr
      character*(*) resname

      subroutine NhlFRLGetMDFloatArray(list_id, resname, rarray,
     +                   num_dim, len_dim, ierr)
```

```
      integer list_id, num_dim, len_dim, ierr
      real rarray(*)
      character*(*) resname

      subroutine NhlFRLGetMDDoubleArray(list_id, resname, darray,
     +                   num_dim, len_dim, ierr)
      integer list_id, num_dim, len_dim, ierr
      double precision darray(*)
      character*(*) resname
```

Arguments

list_id (input)
: Id of the *ResList* to retrieve the *name/val_addr* pair from.

resname (input)
: Specifies the resource name to retrieve the value of.

data_addr (output, C only)
: Specifies the address of the pointer to the array retrieved from the *resname* resource when the ResList is applied to an HLU object using the **NhlGetValues** function. The programmers are responsible for freeing this array when they finish with it.

type_element (output, C only)
: Returns the name of the type of each element in the *data* array. This parameter must be a valid pointer to an NhlString. The NhlString will be assigned during the call to **NhlGetValues**, and the programmer is responsible for freeing the NhlString returned.

size_element (output, C only)
: Returns the size, in bytes, of each element in the *data* array. This parameter must be a valid pointer to an unsigned integer.

num_dimensions (output, C only)
: Returns the number of dimensions in the *data* array. This parameter must be a valid pointer to an integer.

len_dimensions (output, C only)
: Returns an array of length *num_dimensions*. Specifies the length of each dimension in the *data* array. The array will be allocated during the call to **NhlGetValues**, and the programmer is responsible for freeing the array returned.

num_dim (input, Fortran only)
: This parameter must contain the number of dimensions in the data array as it is declared in the program (*iarray* or *rarray*). It is also the number of elements in the *len_dim* array. If the resource that is being returned has fewer dimensions than the data array, this value will be changed to indicate the number of dimensions that were returned.

len_dim (input, Fortran only)
: This array must contain the length of each dimension in the data array as it is declared in the program (*iarray* or *rarray*). If the resource that is being returned has fewer dimensions than the data array, the *num_dim* variable will indicate how many dimensions were returned, and therefore the number of elements in this array that have been changed to indicate the length of each dimension that was returned. The array will be modified during the call to **NhlFGetValues**.

iarray (output, Fortran only)
: The multi-dimensioned integer array retrieved from the *resname* resource when the list is applied

to an HLU object using the **NhlFGetValues** function.

rarray (output, Fortran only)

The multi-dimensioned real array retrieved from the *resname* resource when the list is applied to an HLU object using the **NhlFGetValues** function.

darray (output, Fortran only)

The multi-dimensioned double precision array retrieved from the *resname* resource when the list is applied to an HLU object using the **NhlFGetValues** function.

ierr (output, Fortran only)

Error code.

Types

```
Type name:              NhlTErrorTypes
Definition:
typedef enum _NhlErrType{
        NhlFATAL        = -4,   /* "FATAL"      */
        NhlWARNING      = -3,   /* "WARNING"    */
        NhlINFO         = -2,   /* "INFO"       */
        NhlNOERROR      = -1    /* "NOERROR"    */
} NhlErrorTypes;
```

Description

These functions can only be used on a **NhlGETRL** *ResList*. They are used to retrieve *name/data_addr* pairs from the *ResList* that will be passed to the NhlGetValues function. These functions need to be used to retrieve an Array resource from an HLU object. The programmer is responsible for freeing the memory returned to the *data_addr*, *type_element*, and *len_dimensions* parameters after the call to **NhlGetValues**.

Return values

The C functions return a value of type NhlErrorTypes, and the Fortran subroutines return the error value in *ierr*. If they are unable to retrieve the *name/value* pair to the list, they will return **NhlFATAL**; otherwise they will return **NhlNOERROR**.

See also

- NhlGetValues function

Copyright

Copyright 1987-1999 University Corporation for Atmospheric Research
The use of this Software is governed by a License Agreement.

NCAR Graphics is a registered trademark of the University Corporation for Atmospheric Research.

NhlRLGetMDArray

NhlRLGetMDArray, NhlRLGetMDIntegerArray, NhlRLGetMDFloatArray, NhlRLGetMDDoubleArray

The Fortran names of these functions are **NhlFRLGetMDIntegerArray**, **NhlFRLGetMDFloatArray**, **NhlFRLGetMDDoubleArray**. There is no Fortran binding for **NhlRLGetMDArray**.

Used to add the given *name* and *val_addr* to the *ResList*.

C synopsis

```
#include <ncarg/hlu/hlu.h>

NhlErrorTypes NhlRLGetMDArray(
                int             list_id,
                NhlString       resname,
                NhlPointer      *data_addr,
                NhlString       *type_element,
                unsigned int    *size_element,
                int             *num_dimensions,
                int             **len_dimensions
                )

NhlErrorTypes NhlRLGetMDIntegerArray(
                int             list_id,
                NhlString       resname,
                int             **data_addr,
                int             *num_dimensions,
                int             **len_dimensions
                )

NhlErrorTypes NhlRLGetMDFloatArray(
                int             list_id,
                NhlString       resname,
                float           **data_addr,
                int             *num_dimensions,
                int             **len_dimensions
                )

NhlErrorTypes NhlRLGetMDDoubleArray(
```

```
int             list_id,
NhlString       resname,
double          **data_addr,
int             *num_dimensions,
int             **len_dimensions
)
```

Fortran synopsis

```
      subroutine NhlFRLGetMDIntegerArray(list_id, resname, iarray,
     +                   num_dim, len_dim, ierr)
      integer list_id, iarray(*), num_dim, len_dim, ierr
      character*(*) resname

      subroutine NhlFRLGetMDFloatArray(list_id, resname, rarray,
     +                   num_dim, len_dim, ierr)
      integer list_id, num_dim, len_dim, ierr
      real rarray(*)
      character*(*) resname

      subroutine NhlFRLGetMDDoubleArray(list_id, resname, darray,
     +                   num_dim, len_dim, ierr)
      integer list_id, num_dim, len_dim, ierr
      double precision darray(*)
      character*(*) resname
```

Arguments

list_id (input)
: Id of the *ResList* to retrieve the *name/val_addr* pair from.

resname (input)
: Specifies the resource name to retrieve the value of.

data_addr (output, C only)
: Specifies the address of the pointer to the array retrieved from the *resname* resource when the ResList is applied to an HLU object using the **NhlGetValues** function. The programmers are responsible for freeing this array when they finish with it.

type_element (output, C only)
: Returns the name of the type of each element in the *data* array. This parameter must be a valid pointer to an NhlString. The NhlString will be assigned during the call to **NhlGetValues**, and the programmer is responsible for freeing the NhlString returned.

size_element (output, C only)
: Returns the size, in bytes, of each element in the *data* array. This parameter must be a valid pointer to an unsigned integer.

num_dimensions (output, C only)
: Returns the number of dimensions in the *data* array. This parameter must be a valid pointer to an integer.

len_dimensions (output, C only)
: Returns an array of length *num_dimensions*. Specifies the length of each dimension in the *data*

array. The array will be allocated during the call to **NhlGetValues**, and the programmer is responsible for freeing the array returned.

num_dim (input, Fortran only)
: This parameter must contain the number of dimensions in the data array as it is declared in the program (*iarray* or *rarray*). It is also the number of elements in the *len_dim* array. If the resource that is being returned has fewer dimensions than the data array, this value will be changed to indicate the number of dimensions that were returned.

len_dim (input, Fortran only)
: This array must contain the length of each dimension in the data array as it is declared in the program (*iarray* or *rarray*). If the resource that is being returned has fewer dimensions than the data array, the *num_dim* variable will indicate how many dimensions were returned, and therefore the number of elements in this array that have been changed to indicate the length of each dimension that was returned. The array will be modified during the call to **NhlFGetValues**.

iarray (output, Fortran only)
: The multi-dimensioned integer array retrieved from the *resname* resource when the list is applied to an HLU object using the **NhlFGetValues** function.

rarray (output, Fortran only)
: The multi-dimensioned real array retrieved from the *resname* resource when the list is applied to an HLU object using the **NhlFGetValues** function.

darray (output, Fortran only)
: The multi-dimensioned double precision array retrieved from the *resname* resource when the list is applied to an HLU object using the **NhlFGetValues** function.

ierr (output, Fortran only)
: Error code.

Types

```
Type name:               NhlTErrorTypes
Definition:
typedef enum _NhlErrType{
        NhlFATAL        = -4,    /* "FATAL"       */
        NhlWARNING      = -3,    /* "WARNING"     */
        NhlINFO         = -2,    /* "INFO"        */
        NhlNOERROR      = -1     /* "NOERROR"     */
} NhlErrorTypes;
```

Description

These functions can only be used on a **NhlGETRL** *ResList*. They are used to retrieve *name/data_addr* pairs from the *ResList* that will be passed to the NhlGetValues function. These functions need to be used to retrieve an Array resource from an HLU object. The programmer is responsible for freeing the memory returned to the *data_addr*, *type_element*, and *len_dimensions* parameters after the call to **NhlGetValues**.

Return values

The C functions return a value of type NhlErrorTypes, and the Fortran subroutines return the error value in *ierr*. If they are unable to retrieve the *name/value* pair to the list, they will return **NhlFATAL**; otherwise they will return **NhlNOERROR**.

See also

- NhlGetValues function

Copyright

Copyright 1987-1999 University Corporation for Atmospheric Research
The use of this Software is governed by a License Agreement.

NCAR Graphics is a registered trademark of the University Corporation for Atmospheric Research.

NhlRLGet

NhlRLGet, NhlRLGetInteger, NhlRLGetFloat, NhlRLGetDouble, NhlRLGetString

The Fortran names of these functions are **NhlFRLGetInteger**, **NhlFRLGetFloat**, **NhlFRLGetDouble**, **NhlFRLGetString**. There is no Fortran binding for **NhlRLGet**.

Used to add the given *name* and *val_addr* to the *ResList*.

C synopsis

```
#include <ncarg/hlu/hlu.h>

NhlErrorTypes NhlRLGet(
                int             list_id,
                NhlString       resname,
                NhlString       type,
                <AnyType>       *val_addr
                )

NhlErrorTypes NhlRLGetInteger(
                int             list_id,
```

```
                NhlString       resname,
                int             *val_addr
                )

NhlErrorTypes NhlRLGetFloat(
                int             list_id,
                NhlString       resname,
                float           *val_addr
                )

NhlErrorTypes NhlRLGetDouble(
                int             list_id,
                NhlString       resname,
                double          *val_addr
                )

NhlErrorTypes NhlRLGetString(
                int             list_id,
                NhlString       resname,
                NhlString       *val_addr
                )
```

Fortran synopsis

```
      subroutine NhlFRLGetInteger(list_id, resname, ival, ierr)
      integer list_id, ival, ierr
      character*(*) resname

      subroutine NhlFRLGetFloat(list_id, rename, rval, ierr)
      integer list_id, ierr
      character*(*) resname
      real rval

      subroutine NhlFRLGetDouble(list_id, rename, dval, ierr)
      integer list_id, ierr
      character*(*) resname
      double precision dval

      subroutine NhlFRLGetString(list_id, resname, sval, ierr)
      integer list_id, ierr
      character*(*) resname, sval
```

Arguments

list_id (input)
: Id of the *ResList* to retrieve the *resname/val_addr* pair from.

resname (input)
: Specifies the resource name to retrieve the value of.

type (input, C only)
: Specifies the type *val_addr* points to.

val_addr (output, C only)

Specifies the address to place the value retrieved from the *resname* resource when the list is applied to an HLU object using the **NhlGetValues** function.

ival (output, Fortran only)
The integer value retrieved from the *resname* resource when the list is applied to an HLU object using the **NhlFGetValues** function.

rval (output, Fortran only)
The real value retrieved from the *resname* resource when the list is applied to an HLU object using the **NhlFGetValues** function.

dval (output, Fortran only)
The double precision value retrieved from the *resname* resource when the list is applied to an HLU object using the **NhlFGetValues** function.

sval (output, Fortran only)
The string retrieved from the *resname* resource when the list is applied to an HLU object using the **NhlFGetValues** function.

ierr (output, Fortran only)
Error code.

Types

```
Type name:               NhlTErrorTypes
Definition:
typedef enum _NhlErrType{
        NhlFATAL         = -4,    /* "FATAL"       */
        NhlWARNING       = -3,    /* "WARNING"     */
        NhlINFO          = -2,    /* "INFO"        */
        NhlNOERROR       = -1     /* "NOERROR"     */
} NhlErrorTypes;
```

Description

These functions can only be used on a **NhlGETRL** *ResList*. They are used to retrieve *name/val_addr* pairs from the *ResList* that will be passed to the NhlGetValues function.

Return values

The C functions return a value of type NhlErrorTypes, and the Fortran subroutines return the error value in *ierr*. If they are unable to retrive the *name/val_addr* pair from the list, they will return **NhlFATAL**, otherwise they will return **NhlNOERROR**.

See also

- NhlGetValues function

Copyright

Copyright 1987-1999 University Corporation for Atmospheric Research
The use of this Software is governed by a License Agreement.

NCAR Graphics is a registered trademark of the University Corporation for Atmospheric Research.

NhlRLGetArray

NhlRLGetArray, NhlRLGetIntegerArray, NhlRLGetFloatArray, NhlRLGetDoubleArray, NhlRLGetStringArray

The Fortran names of these functions are **NhlFRLGetIntegerArray**, **NhlFRLGetFloatArray**, **NhlFRLGetDoubleArray**, **NhlFRLGetStringArray**. There is no Fortran binding for **NhlRLGetArray**.

Used to add the given *name* and *val_addr* to the *ResList*.

C synopsis

```
#include <ncarg/hlu/hlu.h>

NhlErrorTypes NhlRLGetArray(
                int             list_id,
                NhlString       resname,
                NhlPointer      *data_addr,
                NhlString       *type_element,
                unsigned int    *size_element,
                int             *num_elements
                )

NhlErrorTypes NhlRLGetIntegerArray(
                int             list_id,
                NhlString       resname,
                int             **data_addr,
                int             *num_elements
                )

NhlErrorTypes NhlRLGetFloatArray(
                int             list_id,
                NhlString       resname,
                float           **data_addr,
                int             *num_elements
```

```
                )

NhlErrorTypes NhlRLGetDoubleArray(
                int             list_id,
                NhlString       resname,
                double          **data_addr,
                int             *num_elements
                )

NhlErrorTypes NhlRLGetStringArray(
                int             list_id,
                NhlString       resname,
                NhlString       **data_addr,
                int             *num_elements
                )
```

Fortran synopsis

```
      subroutine NhlFRLGetIntegerArray(list_id, resname, iarray,
     +                                 num_elements, ierr)
      integer list_id, iarray(num_elements), num_elements, ierr
      character*(*) resname

      subroutine NhlFRLGetFloatArray(list_id, resname, rarray,
     +                               num_elements, ierr)
      integer list_id, num_elements, ierr
      real rarray(num_elements)
      character*(*) resname

      subroutine NhlFRLGetDoubleArray(list_id, resname, darray,
     +                                num_elements, ierr)
      integer list_id, num_elements, ierr
      double precision darray(num_elements)
      character*(*) resname

      subroutine NhlFRLGetStringArray(list_id, resname, sarray,
     +                                num_elements, ierr)
      integer list_id, num_elements, ierr
      character*(*) resname, sarray(num_elements)
```

Arguments

list_id (input)
: Id of the *ResList* to retrieve the *name/val_addr* pair from.

resname (input)
: Specifies the resource name to retrieve the value of.

data_addr (output, C only)
: Specifies the address of the pointer to the array retrieved from the *resname* resource when the ResList is applied to an HLU object using the **NhlGetValues** function. The programmers are responsible for freeing this array when they finish with it.

type_element (output, C only)

Returns the name of the type of each element in the *data* array. This parameter must be a valid pointer to an NhlString. The NhlString will be assigned during the call to **NhlGetValues**, and the programmer is responsible for freeing the NhlString.

size_element (output, C only)
: Returns the size, in bytes, of each element in the *data* array. This parameter must be a valid pointer to an unsigned integer.

num_elements (input)
: Returns the number of elements in the *data* array. For a C program, this parameter must be a valid pointer to an integer. For a Fortran program, this parameter must contain the number of elements in the *data* array as it is declared in the program.

iarray (output, Fortran only)
: The integer array retrieved from the *resname* resource when the list is applied to an HLU object using the **NhlFGetValues** function.

rarray (output, Fortran only)
: The real array retrieved from the *resname* resource when the list is applied to an HLU object using the **NhlFGetValues** function.

darray (output, Fortran only)
: The double precision array retrieved from the *resname* resource when the list is applied to an HLU object using the **NhlFGetValues** function.

sarray (output, Fortran only)
: The string array retrieved from the *resname* resource when the list is applied to an HLU object using the **NhlFGetValues** function.

ierr (output, Fortran only)
: Error code.

Types

```
Type name:                NhlTErrorTypes
Definition:
typedef enum _NhlErrType{
        NhlFATAL        = -4,    /* "FATAL"       */
        NhlWARNING      = -3,    /* "WARNING"     */
        NhlINFO         = -2,    /* "INFO"        */
        NhlNOERROR      = -1     /* "NOERROR"     */
} NhlErrorTypes;
```

Description

These functions can only be used on a **NhlGETRL** *ResList*. They are used to retrieve *name/data_addr* pairs from the *ResList* that will be passed to the NhlGetValues function. These functions need to be used to retrieve an array resource from an HLU object. The programmer is responsible for freeing the memory returned to the *data_addr* and *type_element* parameters after the call to **NhlGetValues**.

Return values

The C functions return a value of type NhlErrorTypes, and the Fortran subroutines return the error value in *ierr*. If they are unable to retrieve the *name/value* pair to the list, they will return **NhlFATAL**; otherwise they will return **NhlNOERROR**.

See also

- NhlGetValues function

Copyright

Copyright 1987-1999 University Corporation for Atmospheric Research
The use of this Software is governed by a License Agreement.

NCAR Graphics is a registered trademark of the University Corporation for Atmospheric Research.

NhlRLCreate

NhlRLCreate, NhlRLDestroy, NhlRLClear, NhlRLUnSet, NhlRLIsSet

The Fortran names of these functions are **NhlFRLCreate**, **NhlFRLDestroy**, **NhlFRLClear**, **NhlFRLUnSet**, and **NhlFRLIsSet**. *ResList* support functions.

C synopsis

```
#include <ncarg/hlu/hlu.h>

int NhlRLCreate(
                NhlRLType       list_type
                )

void NhlRLDestroy(
                int             list_id
                )

void NhlRLClear(
                int             list_id
                )
```

```
void NhlRLUnSet(
                int             list_id,
                NhlString       resname
                )

NhlBoolean NhlRLIsSet(
                int             list_id,
                NhlString       resname
                )
```

Fortran synopsis

```
subroutine NhlFRLCreate(list_id, list_type)
integer list_id
character*(*) list_type

subroutine nhlfrldestroy(list_id)
integer list_id

subroutine nhlfrlclear(list_id)
integer list_id

subroutine nhlfrlunset(list_id, resname)
integer list_id
character*(*) resname

subroutine nhlfrlisset(list_id, resname, ival)
integer list_id, ival
character*(*) resname
```

Arguments

list_type (input)
: Specifies the type of ResList to create. Use NhlSETRL (or 'setrl' in Fortran) for a list being used to set values in an object (**NhlCreate** and **NhlSetValues**). Use NhlGETRL (or 'getrl' in Fortran) for a list being used to retrieve values from an object. (**NhlGetValues**)

list_id (input/output)
: Integer identifier for the *ResList*. It is returned from the **NhlRLCreate** function.

resname (input)
: Resource name. In the **NhlRLIsSet** function, this is used to determine if the given resource name has been set in the *ResList*. In the **NhlRLUnSet** function, it is used to remove the given resource name from the *ResList*.

ival (input, Fortran only)
: Flag indicating whether or not a resource specified by *resname* is in the *ResList*.

Types

```
Type name:              <none>
typedef enum NhlRLType_{
        NhlSETRL,       /* "setrl" */
        NhlGETRL        /* "getrl" */
} NhlRlType;
```

Description

The **NhlRLCreate** function is used to create a ResList. A ResList is a dynamic list that stores *name/value* pairs that are assigned to it. The **NhlRLCreate** function returns a *list_id* that is used as an argument to the Create, SetValues, and GetValues *ResList* functions, as well as the other *ResList* support functions.

The **NhlRLDestroy** function is used to *free* an existing *ResList*. The *list_id* number may be reused the next time a **NhlRLCreate** is called.

The **NhlRLClear** function is used to clear all the resource names and values out of the *ResList*. It basically empties out the list.

The **NhlRLUnSet** function is used to remove the resource name specified by the *resname* argument from the *ResList*.

The **NhlRLIsSet** function is used to find out if the resource name specified by the *resname* argument is currently in the *ResList*.

Return values

The **NhlRLCreate** C function returns a positive integer that is used to identify the *ResList*, and the **NhlFRLCreate** Fortran subroutine returns this value in *list_id*. If it returns a non-positive number, then it was unable to create a *ResList*.

The **NhlRLIsSet** function returns True if *resname* is currently contained in the *ResList* and False if it is not; the **NhlFRLIsSet** Fortran subroutine returns this value in *ival*.

See also

- Create function
- SetValues function
- GetValues function

Copyright

Copyright 1987-1999 University Corporation for Atmospheric Research
The use of this Software is governed by a License Agreement.

NCAR Graphics is a registered trademark of the University Corporation for Atmospheric Research.

NhlRLSet

NhlRLSet, NhlRLSetInteger, NhlRLSetFloat, NhlRLSetDouble, NhlRLSetString

The Fortran names of these functions are **NhlFRLSetInteger**, **NhlFRLSetFloat**, **NhlFRLSetDouble**, **NhlFRLSetString**. There is no Fortran binding for **NhlRLSet**.

Used to add the given *name* and *value* to the *ResList*.

C synopsis

```
#include <ncarg/hlu/hlu.h>

NhlErrorTypes NhlRLSet(
                int             list_id,
                NhlString       resname,
                NhlString       type,
                <AnyType>       value
                )

NhlErrorTypes NhlRLSetInteger(
                int             list_id,
                NhlString       resname,
                int             value
                )

NhlErrorTypes NhlRLSetFloat(
                int             list_id,
                NhlString       resname,
                float           value
                )

NhlErrorTypes NhlRLSetDouble(
                int             list_id,
                NhlString       resname,
                double          value
                )

NhlErrorTypes NhlRLSetString(
```

```
int             list_id,
NhlString       resname,
NhlString       value
)
```

Fortran synopsis

```
subroutine NhlFRLSetInteger(list_id, resname, ival, ierr)
integer list_id, ival, ierr
character*(*) resname

subroutine NhlFRLSetFloat(list_id, resname, rval, ierr)
integer list_id, ierr
real rval
character*(*) resname

subroutine NhlFRLSetDouble(list_id, resname, rval, ierr)
integer list_id, ierr
double precision dval
character*(*) resname

subroutine NhlFRLSetString(list_id, resname, sval, ierr)
integer list_id, ierr
character*(*) resname, sval
```

Arguments

list_id (input)
: Id of the *ResList* to add the *name/value* pair to.

resname (input)
: Specifies the resource name to apply the given *value* to when the list is applied to an HLU object.

type (input, C only)
: Specifies the type for *value*.

value (input, C only)
: Specifies the value that will be applied to the corresponding *resname* when the list is applied to an HLU object.

ival (input, Fortran only)
: Specifies the integer value that will be applied to the corresponding *resname* resource when the list is applied to an HLU object.

rval (input, Fortran only)
: Specifies the real value that will be applied to the corresponding *resname* resource when the list is applied to an HLU object.

dval (input, Fortran only)
: Specifies the double precision value that will be applied to the corresponding *resname* resource when the list is applied to an HLU object.

sval (input, Fortran only)
: Specifies the string that will be applied to the corresponding *resname* resource when the list is applied to an HLU object.

ierr (output, Fortran only)
Error code.

Types

```
Type name:               NhlTErrorTypes
Definition:
typedef enum _NhlErrType{
        NhlFATAL         = -4,    /* "FATAL"       */
        NhlWARNING       = -3,    /* "WARNING"     */
        NhlINFO          = -2,    /* "INFO"        */
        NhlNOERROR       = -1     /* "NOERROR"     */
} NhlErrorTypes;
```

Description

These functions can only be used on a **NhlSETRL** *ResList*. They are used to add *name/value* pairs to the *ResList* that will be passed to the NhlCreate or NhlSetValues functions.

Return values

The C functions return a value of type NhlErrorTypes, and the Fortran subroutines return the error value in *ierr*. If they are unable to add the *name/val_addr* pair to the list, they will return **NhlFATAL**; otherwise they will return **NhlNOERROR**.

See also

- NhlCreate function
- NhlSetValues function

Copyright

Copyright 1987-1999 University Corporation for Atmospheric Research
The use of this Software is governed by a License Agreement.

NCAR Graphics is a registered trademark of the University Corporation for Atmospheric Research.

NhlRLSetArray

NhlRLSetArray, NhlRLSetIntegerArray, NhlRLSetFloatArray, NhlRLSetDoubleArray, NhlRLSetStringArray

The Fortran names of these functions are **NhlFRLSetIntegerArray**, **NhlFRLSetFloatArray**, **NhlFRLSetDoubleArray**, **NhlFRLSetStringArray**. There is no Fortran binding for **NhlRLSetArray**.

Used to add the given *name* and *value* array to the *ResList*.

C synopsis

```
#include <ncarg/hlu/hlu.h>

NhlErrorTypes NhlRLSetArray(
                int             list_id,
                NhlString       resname,
                NhlPointer      data,
                NhlString       type_element,
                unsigned int    size_element,
                int             num_elements
                )

NhlErrorTypes NhlRLSetIntegerArray(
                int             list_id,
                NhlString       resname,
                int             *data,
                int             num_elements
                )

NhlErrorTypes NhlRLSetFloatArray(
                int             list_id,
                NhlString       resname,
                float           *data,
                int             num_elements
                )

NhlErrorTypes NhlRLSetDoubleArray(
                int             list_id,
                NhlString       resname,
                double          *data,
                int             num_elements
                )

NhlErrorTypes NhlRLSetStringArray(
                int             list_id,
                NhlString       resname,
                NhlString       *data,
                int             num_elements
                )
```

Fortran synopsis

```
 subroutine NhlFRLSetIntegerArray(list_id, resname, iarray,
+                                  num_elements, ierr)
 integer list_id, iarray(num_elements), num_elements, ierr
 character*(*) resname

 subroutine NhlFRLSetFloatArray(list_id, resname, rarray,
+                                num_elements, ierr)
 integer list_id, num_elements, ierr
 real rarray(num_elements)
 character*(*) resname

 subroutine NhlFRLSetDoubleArray(list_id, resname, darray,
+                                num_elements, ierr)
 integer list_id, num_elements, ierr
 double precision darray(num_elements)
 character*(*) resname

 subroutine NhlFRLSetStringArray(list_id, resname, sarray,
+                                 num_elements, ierr)
 integer list_id, num_elements, ierr
 character*(*) resname, sarray(num_elements)
```

Arguments

list_id (input)
: Id of the *ResList* to add the *name/value* pair to.

resname (input)
: Specifies the resource name to apply the given *data* to when the list is applied to an HLU object.

data (input, C only)
: Specifies the array value that will be applied to the corresponding *resname* when the list is applied to an HLU object.

type_elements (input, C only)
: Specifies the type of each element of the *data* array.

size_elements (input, C only)
: Specifies the size in bytes of each element of the *data* array.

num_elements (input)
: Specifies the number of elements in the *data* array.

iarray (input, Fortran only)
: Specifies the integer array that will be applied to the corresponding *resname* resource when the list is applied to an HLU object.

rarray (input, Fortran only)
: Specifies the real array that will be applied to the corresponding *resname* resource when the list is applied to an HLU object.

darray (input, Fortran only)
: Specifies the double precision array that will be applied to the corresponding *resname* resource when the list is applied to an HLU object.

sarray (input, Fortran only)

Specifies the string array that will be applied to the corresponding *resname* resource when the list is applied to an HLU object.

ierr (output, Fortran only)

Error code.

Types

```
Type name:                NhlTErrorTypes
Definition:
typedef enum _NhlErrType{
        NhlFATAL        = -4,    /* "FATAL"      */
        NhlWARNING      = -3,    /* "WARNING"    */
        NhlINFO         = -2,    /* "INFO"       */
        NhlNOERROR      = -1     /* "NOERROR"    */
} NhlErrorTypes;
```

Description

These functions can only be used on a **NhlSETRL** *ResList*. They are used to add *name/value* pairs to the *ResList* that will be passed to the NhlCreate or NhlSetValues functions. These functions need to be used when the *value* part of the *name/value* pair is an Array.

Return values

The C functions return a value of type NhlErrorTypes, and the Fortran subroutines return the error value in *ierr*. If they are unable to add the *name/val_addr* pair to the list, they will return **NhlFATAL**; otherwise they will return **NhlNOERROR**.

See also

- NhlCreate function
- NhlSetValues function

Copyright

Copyright 1987-1999 University Corporation for Atmospheric Research
The use of this Software is governed by a License Agreement.

NCAR Graphics is a registered trademark of the University Corporation for Atmospheric Research.

NhlRLSet

NhlRLSet, NhlRLSetInteger, NhlRLSetFloat, NhlRLSetDouble, NhlRLSetString

The Fortran names of these functions are **NhlFRLSetInteger**, **NhlFRLSetFloat**, **NhlFRLSetDouble**, **NhlFRLSetString**. There is no Fortran binding for **NhlRLSet**.

Used to add the given *name* and *value* to the *ResList*.

C synopsis

```
#include <ncarg/hlu/hlu.h>

NhlErrorTypes NhlRLSet(
                int             list_id,
                NhlString       resname,
                NhlString       type,
                <AnyType>       value
                )

NhlErrorTypes NhlRLSetInteger(
                int             list_id,
                NhlString       resname,
                int             value
                )

NhlErrorTypes NhlRLSetFloat(
                int             list_id,
                NhlString       resname,
                float           value
                )

NhlErrorTypes NhlRLSetDouble(
                int             list_id,
                NhlString       resname,
                double          value
                )

NhlErrorTypes NhlRLSetString(
                int             list_id,
                NhlString       resname,
                NhlString       value
                )
```

Fortran synopsis

```
subroutine NhlFRLSetInteger(list_id, resname, ival, ierr)
integer list_id, ival, ierr
character*(*) resname

subroutine NhlFRLSetFloat(list_id, resname, rval, ierr)
integer list_id, ierr
real rval
character*(*) resname

subroutine NhlFRLSetDouble(list_id, resname, rval, ierr)
integer list_id, ierr
double precision dval
character*(*) resname

subroutine NhlFRLSetString(list_id, resname, sval, ierr)
integer list_id, ierr
character*(*) resname, sval
```

Arguments

list_id (input)
Id of the *ResList* to add the *name/value* pair to.
resname (input)
Specifies the resource name to apply the given *value* to when the list is applied to an HLU object.
type (input, C only)
Specifies the type for *value*.
value (input, C only)
Specifies the value that will be applied to the corresponding *resname* when the list is applied to an HLU object.
ival (input, Fortran only)
Specifies the integer value that will be applied to the corresponding *resname* resource when the list is applied to an HLU object.
rval (input, Fortran only)
Specifies the real value that will be applied to the corresponding *resname* resource when the list is applied to an HLU object.
dval (input, Fortran only)
Specifies the double precision value that will be applied to the corresponding *resname* resource when the list is applied to an HLU object.
sval (input, Fortran only)
Specifies the string that will be applied to the corresponding *resname* resource when the list is applied to an HLU object.
ierr (output, Fortran only)
Error code.

Types

`Type name:` **`NhlTErrorTypes`**

```
Definition:
typedef enum _NhlErrType{
        NhlFATAL        = -4,    /* "FATAL"      */
        NhlWARNING      = -3,    /* "WARNING"    */
        NhlINFO         = -2,    /* "INFO"       */
        NhlNOERROR      = -1     /* "NOERROR"    */
} NhlErrorTypes;
```

Description

These functions can only be used on a **NhlSETRL** *ResList*. They are used to add *name/value* pairs to the *ResList* that will be passed to the NhlCreate or NhlSetValues functions.

Return values

The C functions return a value of type NhlErrorTypes, and the Fortran subroutines return the error value in *ierr*. If they are unable to add the *name/val_addr* pair to the list, they will return **NhlFATAL**; otherwise they will return **NhlNOERROR**.

See also

- NhlCreate function
- NhlSetValues function

Copyright

Copyright 1987-1999 University Corporation for Atmospheric Research
The use of this Software is governed by a License Agreement.

NCAR Graphics is a registered trademark of the University Corporation for Atmospheric Research.

NhlRLSetArray

NhlRLSetArray, NhlRLSetIntegerArray, NhlRLSetFloatArray, NhlRLSetDoubleArray, NhlRLSetStringArray

The Fortran names of these functions are **NhlFRLSetIntegerArray**, **NhlFRLSetFloatArray**, **NhlFRLSetDoubleArray**, **NhlFRLSetStringArray**. There is no Fortran binding for **NhlRLSetArray**.

Used to add the given *name* and *value* array to the *ResList*.

C synopsis

```
#include <ncarg/hlu/hlu.h>

NhlErrorTypes NhlRLSetArray(
                int             list_id,
                NhlString       resname,
                NhlPointer      data,
                NhlString       type_element,
                unsigned int    size_element,
                int             num_elements
                )

NhlErrorTypes NhlRLSetIntegerArray(
                int             list_id,
                NhlString       resname,
                int             *data,
                int             num_elements
                )

NhlErrorTypes NhlRLSetFloatArray(
                int             list_id,
                NhlString       resname,
                float           *data,
                int             num_elements
                )

NhlErrorTypes NhlRLSetDoubleArray(
                int             list_id,
                NhlString       resname,
                double          *data,
                int             num_elements
                )

NhlErrorTypes NhlRLSetStringArray(
                int             list_id,
                NhlString       resname,
                NhlString       *data,
                int             num_elements
                )
```

Fortran synopsis

```
      subroutine NhlFRLSetIntegerArray(list_id, resname, iarray,
     +                                  num_elements, ierr)
      integer list_id, iarray(num_elements), num_elements, ierr
      character*(*) resname

      subroutine NhlFRLSetFloatArray(list_id, resname, rarray,
     +                                num_elements, ierr)
      integer list_id, num_elements, ierr
```

```
      real rarray(num_elements)
      character*(*) resname

      subroutine NhlFRLSetDoubleArray(list_id, resname, darray,
     +                               num_elements, ierr)
      integer list_id, num_elements, ierr
      double precision darray(num_elements)
      character*(*) resname

      subroutine NhlFRLSetStringArray(list_id, resname, sarray,
     +                                num_elements, ierr)
      integer list_id, num_elements, ierr
      character*(*) resname, sarray(num_elements)
```

Arguments

list_id (input)
: Id of the *ResList* to add the *name/value* pair to.

resname (input)
: Specifies the resource name to apply the given *data* to when the list is applied to an HLU object.

data (input, C only)
: Specifies the array value that will be applied to the corresponding *resname* when the list is applied to an HLU object.

type_elements (input, C only)
: Specifies the type of each element of the *data* array.

size_elements (input, C only)
: Specifies the size in bytes of each element of the *data* array.

num_elements (input)
: Specifies the number of elements in the *data* array.

iarray (input, Fortran only)
: Specifies the integer array that will be applied to the corresponding *resname* resource when the list is applied to an HLU object.

rarray (input, Fortran only)
: Specifies the real array that will be applied to the corresponding *resname* resource when the list is applied to an HLU object.

darray (input, Fortran only)
: Specifies the double precision array that will be applied to the corresponding *resname* resource when the list is applied to an HLU object.

sarray (input, Fortran only)
: Specifies the string array that will be applied to the corresponding *resname* resource when the list is applied to an HLU object.

ierr (output, Fortran only)
: Error code.

Types

```
Type name:              NhlTErrorTypes
Definition:
typedef enum _NhlErrType{
        NhlFATAL        = -4,    /* "FATAL"       */
        NhlWARNING      = -3,    /* "WARNING"     */
        NhlINFO         = -2,    /* "INFO"        */
        NhlNOERROR      = -1     /* "NOERROR"     */
} NhlErrorTypes;
```

Description

These functions can only be used on a **NhlSETRL** *ResList*. They are used to add *name/value* pairs to the *ResList* that will be passed to the NhlCreate or NhlSetValues functions. These functions need to be used when the *value* part of the *name/value* pair is an Array.

Return values

The C functions return a value of type NhlErrorTypes, and the Fortran subroutines return the error value in *ierr*. If they are unable to add the *name/val_addr* pair to the list, they will return **NhlFATAL**; otherwise they will return **NhlNOERROR**.

See also

- NhlCreate function
- NhlSetValues function

Copyright

Copyright 1987-1999 University Corporation for Atmospheric Research
The use of this Software is governed by a License Agreement.

NCAR Graphics is a registered trademark of the University Corporation for Atmospheric Research.

NhlRLSet

NhlRLSet, NhlRLSetInteger, NhlRLSetFloat, NhlRLSetDouble, NhlRLSetString

The Fortran names of these functions are **NhlFRLSetInteger**, **NhlFRLSetFloat**, **NhlFRLSetDouble**,

NhlFRLSetString. There is no Fortran binding for **NhlRLSet**.

Used to add the given *name* and *value* to the *ResList*.

C synopsis

```
#include <ncarg/hlu/hlu.h>

NhlErrorTypes NhlRLSet(
                int              list_id,
                NhlString        resname,
                NhlString        type,
                <AnyType>        value
                )

NhlErrorTypes NhlRLSetInteger(
                int              list_id,
                NhlString        resname,
                int              value
                )

NhlErrorTypes NhlRLSetFloat(
                int              list_id,
                NhlString        resname,
                float            value
                )

NhlErrorTypes NhlRLSetDouble(
                int              list_id,
                NhlString        resname,
                double           value
                )

NhlErrorTypes NhlRLSetString(
                int              list_id,
                NhlString        resname,
                NhlString        value
                )
```

Fortran synopsis

```
      subroutine NhlFRLSetInteger(list_id, resname, ival, ierr)
      integer list_id, ival, ierr
      character*(*) resname

      subroutine NhlFRLSetFloat(list_id, resname, rval, ierr)
      integer list_id, ierr
      real rval
      character*(*) resname

      subroutine NhlFRLSetDouble(list_id, resname, rval, ierr)
      integer list_id, ierr
      double precision dval
```

```
character*(*) resname

subroutine NhlFRLSetString(list_id, resname, sval, ierr)
integer list_id, ierr
character*(*) resname, sval
```

Arguments

list_id (input)
: Id of the *ResList* to add the *name/value* pair to.

resname (input)
: Specifies the resource name to apply the given *value* to when the list is applied to an HLU object.

type (input, C only)
: Specifies the type for *value*.

value (input, C only)
: Specifies the value that will be applied to the corresponding *resname* when the list is applied to an HLU object.

ival (input, Fortran only)
: Specifies the integer value that will be applied to the corresponding *resname* resource when the list is applied to an HLU object.

rval (input, Fortran only)
: Specifies the real value that will be applied to the corresponding *resname* resource when the list is applied to an HLU object.

dval (input, Fortran only)
: Specifies the double precision value that will be applied to the corresponding *resname* resource when the list is applied to an HLU object.

sval (input, Fortran only)
: Specifies the string that will be applied to the corresponding *resname* resource when the list is applied to an HLU object.

ierr (output, Fortran only)
: Error code.

Types

```
Type name:                NhlTErrorTypes
Definition:
typedef enum _NhlErrType{
        NhlFATAL          = -4,    /* "FATAL"       */
        NhlWARNING        = -3,    /* "WARNING"     */
        NhlINFO           = -2,    /* "INFO"        */
        NhlNOERROR        = -1     /* "NOERROR"     */
} NhlErrorTypes;
```

Description

These functions can only be used on a **NhlSETRL** *ResList*. They are used to add *name/value* pairs to the *ResList* that will be passed to the NhlCreate or NhlSetValues functions.

Return values

The C functions return a value of type NhlErrorTypes, and the Fortran subroutines return the error value in *ierr*. If they are unable to add the *name/val_addr* pair to the list, they will return **NhlFATAL**; otherwise they will return **NhlNOERROR**.

See also

- NhlCreate function
- NhlSetValues function

Copyright

Copyright 1987-1999 University Corporation for Atmospheric Research
The use of this Software is governed by a License Agreement.

NCAR Graphics is a registered trademark of the University Corporation for Atmospheric Research.

NhlRLSetArray

NhlRLSetArray, NhlRLSetIntegerArray, NhlRLSetFloatArray, NhlRLSetDoubleArray, NhlRLSetStringArray

The Fortran names of these functions are **NhlFRLSetIntegerArray**, **NhlFRLSetFloatArray**, **NhlFRLSetDoubleArray**, **NhlFRLSetStringArray**. There is no Fortran binding for **NhlRLSetArray**.

Used to add the given *name* and *value* array to the *ResList*.

C synopsis

```
#include <ncarg/hlu/hlu.h>

NhlErrorTypes NhlRLSetArray(
```

```
                int               list_id,
                NhlString         resname,
                NhlPointer        data,
                NhlString         type_element,
                unsigned int      size_element,
                int               num_elements
                )

NhlErrorTypes NhlRLSetIntegerArray(
                int               list_id,
                NhlString         resname,
                int               *data,
                int               num_elements
                )

NhlErrorTypes NhlRLSetFloatArray(
                int               list_id,
                NhlString         resname,
                float             *data,
                int               num_elements
                )

NhlErrorTypes NhlRLSetDoubleArray(
                int               list_id,
                NhlString         resname,
                double            *data,
                int               num_elements
                )

NhlErrorTypes NhlRLSetStringArray(
                int               list_id,
                NhlString         resname,
                NhlString         *data,
                int               num_elements
                )
```

Fortran synopsis

```
      subroutine NhlFRLSetIntegerArray(list_id, resname, iarray,
     +                                 num_elements, ierr)
      integer list_id, iarray(num_elements), num_elements, ierr
      character*(*) resname

      subroutine NhlFRLSetFloatArray(list_id, resname, rarray,
     +                               num_elements, ierr)
      integer list_id, num_elements, ierr
      real rarray(num_elements)
      character*(*) resname

      subroutine NhlFRLSetDoubleArray(list_id, resname, darray,
     +                                num_elements, ierr)
      integer list_id, num_elements, ierr
      double precision darray(num_elements)
      character*(*) resname

      subroutine NhlFRLSetStringArray(list_id, resname, sarray,
     +                                num_elements, ierr)
```

```
      integer list_id, num_elements, ierr
      character*(*) resname, sarray(num_elements)
```

Arguments

list_id (input)
Id of the *ResList* to add the *name/value* pair to.

resname (input)
Specifies the resource name to apply the given *data* to when the list is applied to an HLU object.

data (input, C only)
Specifies the array value that will be applied to the corresponding *resname* when the list is applied to an HLU object.

type_elements (input, C only)
Specifies the type of each element of the *data* array.

size_elements (input, C only)
Specifies the size in bytes of each element of the *data* array.

num_elements (input)
Specifies the number of elements in the *data* array.

iarray (input, Fortran only)
Specifies the integer array that will be applied to the corresponding *resname* resource when the list is applied to an HLU object.

rarray (input, Fortran only)
Specifies the real array that will be applied to the corresponding *resname* resource when the list is applied to an HLU object.

darray (input, Fortran only)
Specifies the double precision array that will be applied to the corresponding *resname* resource when the list is applied to an HLU object.

sarray (input, Fortran only)
Specifies the string array that will be applied to the corresponding *resname* resource when the list is applied to an HLU object.

ierr (output, Fortran only)
Error code.

Types

```
Type name:                NhlTErrorTypes
Definition:
typedef enum _NhlErrType{
        NhlFATAL          = -4,    /* "FATAL"       */
        NhlWARNING        = -3,    /* "WARNING"     */
        NhlINFO           = -2,    /* "INFO"        */
        NhlNOERROR        = -1     /* "NOERROR"     */
} NhlErrorTypes;
```

Description

These functions can only be used on a **NhlSETRL** *ResList*. They are used to add *name/value* pairs to the *ResList* that will be passed to the NhlCreate or NhlSetValues functions. These functions need to be used when the *value* part of the *name/value* pair is an Array.

Return values

The C functions return a value of type NhlErrorTypes, and the Fortran subroutines return the error value in *ierr*. If they are unable to add the *name/val_addr* pair to the list, they will return **NhlFATAL**; otherwise they will return **NhlNOERROR**.

See also

- NhlCreate function
- NhlSetValues function

Copyright

Copyright 1987-1999 University Corporation for Atmospheric Research
The use of this Software is governed by a License Agreement.

NCAR Graphics is a registered trademark of the University Corporation for Atmospheric Research.

NhlRLSet

NhlRLSet, NhlRLSetInteger, NhlRLSetFloat, NhlRLSetDouble, NhlRLSetString

The Fortran names of these functions are **NhlFRLSetInteger**, **NhlFRLSetFloat**, **NhlFRLSetDouble**, **NhlFRLSetString**. There is no Fortran binding for **NhlRLSet**.

Used to add the given *name* and *value* to the *ResList*.

C synopsis

```
#include <ncarg/hlu/hlu.h>

NhlErrorTypes NhlRLSet(
                int             list_id,
                NhlString       resname,
                NhlString       type,
                <AnyType>       value
                )

NhlErrorTypes NhlRLSetInteger(
                int             list_id,
                NhlString       resname,
                int             value
                )

NhlErrorTypes NhlRLSetFloat(
                int             list_id,
                NhlString       resname,
                float           value
                )

NhlErrorTypes NhlRLSetDouble(
                int             list_id,
                NhlString       resname,
                double          value
                )

NhlErrorTypes NhlRLSetString(
                int             list_id,
                NhlString       resname,
                NhlString       value
                )
```

Fortran synopsis

```
      subroutine NhlFRLSetInteger(list_id, resname, ival, ierr)
      integer list_id, ival, ierr
      character*(*) resname

      subroutine NhlFRLSetFloat(list_id, resname, rval, ierr)
      integer list_id, ierr
      real rval
      character*(*) resname

      subroutine NhlFRLSetDouble(list_id, resname, rval, ierr)
      integer list_id, ierr
      double precision dval
      character*(*) resname

      subroutine NhlFRLSetString(list_id, resname, sval, ierr)
      integer list_id, ierr
      character*(*) resname, sval
```

Arguments

list_id (input)
Id of the *ResList* to add the *name/value* pair to.

resname (input)
Specifies the resource name to apply the given *value* to when the list is applied to an HLU object.

type (input, C only)
Specifies the type for *value*.

value (input, C only)
Specifies the value that will be applied to the corresponding *resname* when the list is applied to an HLU object.

ival (input, Fortran only)
Specifies the integer value that will be applied to the corresponding *resname* resource when the list is applied to an HLU object.

rval (input, Fortran only)
Specifies the real value that will be applied to the corresponding *resname* resource when the list is applied to an HLU object.

dval (input, Fortran only)
Specifies the double precision value that will be applied to the corresponding *resname* resource when the list is applied to an HLU object.

sval (input, Fortran only)
Specifies the string that will be applied to the corresponding *resname* resource when the list is applied to an HLU object.

ierr (output, Fortran only)
Error code.

Types

```
Type name:                NhlTErrorTypes
Definition:
typedef enum _NhlErrType{
        NhlFATAL          = -4,    /* "FATAL"       */
        NhlWARNING        = -3,    /* "WARNING"     */
        NhlINFO           = -2,    /* "INFO"        */
        NhlNOERROR        = -1     /* "NOERROR"     */
} NhlErrorTypes;
```

Description

These functions can only be used on a **NhlSETRL** *ResList*. They are used to add *name/value* pairs to the *ResList* that will be passed to the NhlCreate or NhlSetValues functions.

Return values

The C functions return a value of type NhlErrorTypes, and the Fortran subroutines return the error value in *ierr*. If they are unable to add the *name/val_addr* pair to the list, they will return **NhlFATAL**; otherwise they will return **NhlNOERROR**.

See also

- NhlCreate function
- NhlSetValues function

Copyright

Copyright 1987-1999 University Corporation for Atmospheric Research
The use of this Software is governed by a License Agreement.

NCAR Graphics is a registered trademark of the University Corporation for Atmospheric Research.

NhlRLSetArray

NhlRLSetArray, NhlRLSetIntegerArray, NhlRLSetFloatArray, NhlRLSetDoubleArray, NhlRLSetStringArray

The Fortran names of these functions are **NhlFRLSetIntegerArray**, **NhlFRLSetFloatArray**, **NhlFRLSetDoubleArray**, **NhlFRLSetStringArray**. There is no Fortran binding for **NhlRLSetArray**.

Used to add the given *name* and *value* array to the *ResList*.

C synopsis

```
#include <ncarg/hlu/hlu.h>

NhlErrorTypes NhlRLSetArray(
                int             list_id,
                NhlString       resname,
                NhlPointer      data,
                NhlString       type_element,
                unsigned int    size_element,
                int             num_elements
                )

NhlErrorTypes NhlRLSetIntegerArray(
                int             list_id,
                NhlString       resname,
```

```
                int              *data,
                int              num_elements
                )

NhlErrorTypes NhlRLSetFloatArray(
                int              list_id,
                NhlString        resname,
                float            *data,
                int              num_elements
                )

NhlErrorTypes NhlRLSetDoubleArray(
                int              list_id,
                NhlString        resname,
                double           *data,
                int              num_elements
                )

NhlErrorTypes NhlRLSetStringArray(
                int              list_id,
                NhlString        resname,
                NhlString        *data,
                int              num_elements
                )
```

Fortran synopsis

```
      subroutine NhlFRLSetIntegerArray(list_id, resname, iarray,
     +                                 num_elements, ierr)
      integer list_id, iarray(num_elements), num_elements, ierr
      character*(*) resname

      subroutine NhlFRLSetFloatArray(list_id, resname, rarray,
     +                               num_elements, ierr)
      integer list_id, num_elements, ierr
      real rarray(num_elements)
      character*(*) resname

      subroutine NhlFRLSetDoubleArray(list_id, resname, darray,
     +                               num_elements, ierr)
      integer list_id, num_elements, ierr
      double precision darray(num_elements)
      character*(*) resname

      subroutine NhlFRLSetStringArray(list_id, resname, sarray,
     +                                num_elements, ierr)
      integer list_id, num_elements, ierr
      character*(*) resname, sarray(num_elements)
```

Arguments

list_id (input)
: Id of the *ResList* to add the *name/value* pair to.

resname (input)
: Specifies the resource name to apply the given *data* to when the list is applied to an HLU object.

data (input, C only)
: Specifies the array value that will be applied to the corresponding *resname* when the list is applied to an HLU object.

type_elements (input, C only)
: Specifies the type of each element of the *data* array.

size_elements (input, C only)
: Specifies the size in bytes of each element of the *data* array.

num_elements (input)
: Specifies the number of elements in the *data* array.

iarray (input, Fortran only)
: Specifies the integer array that will be applied to the corresponding *resname* resource when the list is applied to an HLU object.

rarray (input, Fortran only)
: Specifies the real array that will be applied to the corresponding *resname* resource when the list is applied to an HLU object.

darray (input, Fortran only)
: Specifies the double precision array that will be applied to the corresponding *resname* resource when the list is applied to an HLU object.

sarray (input, Fortran only)
: Specifies the string array that will be applied to the corresponding *resname* resource when the list is applied to an HLU object.

ierr (output, Fortran only)
: Error code.

Types

```
Type name:                NhlTErrorTypes
Definition:
typedef enum _NhlErrType{
        NhlFATAL        = -4,    /* "FATAL"      */
        NhlWARNING      = -3,    /* "WARNING"    */
        NhlINFO         = -2,    /* "INFO"       */
        NhlNOERROR      = -1     /* "NOERROR"    */
} NhlErrorTypes;
```

Description

These functions can only be used on a **NhlSETRL** *ResList*. They are used to add *name/value* pairs to the *ResList* that will be passed to the NhlCreate or NhlSetValues functions. These functions need to be used when the *value* part of the *name/value* pair is an Array.

Return values

The C functions return a value of type NhlErrorTypes, and the Fortran subroutines return the error value in *ierr*. If they are unable to add the *name/val_addr* pair to the list, they will return **NhlFATAL**; otherwise they will return **NhlNOERROR**.

See also

- NhlCreate function
- NhlSetValues function

Copyright

Copyright 1987-1999 University Corporation for Atmospheric Research
The use of this Software is governed by a License Agreement.

NCAR Graphics is a registered trademark of the University Corporation for Atmospheric Research.

NhlRLSetMDArray

NhlRLSetMDArray, NhlRLSetMDIntegerArray, NhlRLSetMDFloatArray, NhlRLSetMDDoubleArray

The Fortran names of these functions are **NhlFRLSetMDIntegerArray**, **NhlFRLSetMDFloatArray** **NhlFRLSetMDDoubleArray**. There is no Fortran binding for **NhlRLSetMDArray**.

Used to add the given *name* and *value* array to the *ResList*.

C synopsis

```
#include <ncarg/hlu/hlu.h>

NhlErrorTypes NhlRLSetMDArray(
                int             list_id,
                NhlString       resname,
                NhlPointer      data,
                NhlString       type_element,
                unsigned int    size_element,
                int             num_dimensions,
                int             *len_dimensions
```

```
                )

NhlErrorTypes NhlRLSetMDIntegerArray(
                int             list_id,
                NhlString       resname,
                int             *data,
                int             num_dimensions,
                int             *len_dimensions
                )

NhlErrorTypes NhlRLSetMDFloatArray(
                int             list_id,
                NhlString       resname,
                float           *data,
                int             num_dimensions,
                int             *len_dimensions
                )

NhlErrorTypes NhlRLSetMDDoubleArray(
                int             list_id,
                NhlString       resname,
                double          *data,
                int             num_dimensions,
                int             *len_dimensions
                )
```

Fortran synopsis

```
      subroutine NhlFRLSetMDIntegerArray(list_id, resname, iarray,
     +                   num_dimensions, len_dimensions, ierr)
      integer list_id, iarray(*), num_dimensions, len_dimensions, ierr
      character*(*) resname

      subroutine NhlFRLSetMDFloatArray(list_id, resname, rarray,
     +                   num_dimensions, len_dimensions, ierr)
      integer list_id, num_dimensions, len_dimensions, ierr
      real rarray(*)
      character*(*) resname

      subroutine NhlFRLSetMDDoubleArray(list_id, resname, darray,
     +                   num_dimensions, len_dimensions, ierr)
      integer list_id, num_dimensions, len_dimensions, ierr
      double precision darray(*)
      character*(*) resname
```

Arguments

list_id (input)
: Id of the *ResList* to add the *name/value* pair to.

resname (input)
: Specifies the resource name to apply the given *data* to when the list is applied to an HLU object.

data (input, C only)

Specifies the array value that will be applied to the corresponding *resname* when the list is applied to an HLU object.

type_elements (input, C only)
: Specifies the type of each element of the *data* array.

size_elements (input, C only)
: Specifies the size in bytes of each element of the *data* array.

num_dimensions (input)
: Specifies the number of dimensions in the *data* array.

len_dimensions (input)
: An integer array of length *num_dimensions* that specifies the length of each dimensions in the *data* array.

iarray (input, Fortran only)
: Specifies the multi-dimensioned integer array that will be applied to the corresponding *resname* resource when the list is applied to an HLU object.

rarray (input, Fortran only)
: Specifies the multi-dimensioned real array that will be applied to the corresponding *resname* resource when the list is applied to an HLU object.

darray (input, Fortran only)
: Specifies the multi-dimensioned double precision array that will be applied to the corresponding *resname* resource when the list is applied to an HLU object.

ierr (output, Fortran only)
: Error code.

Types

```
Type name:                NhlTErrorTypes
Definition:
typedef enum _NhlErrType{
        NhlFATAL          = -4,    /* "FATAL"       */
        NhlWARNING        = -3,    /* "WARNING"     */
        NhlINFO           = -2,    /* "INFO"        */
        NhlNOERROR        = -1     /* "NOERROR"     */
} NhlErrorTypes;
```

Description

These functions can only be used on a **NhlSETRL** *ResList*. They are used to add *name/value* pairs to the *ResList* that will be passed to the NhlCreate or NhlSetValues functions. These functions need to be used when the *value* part of the *name/value* pair is an Array.

Return values

The C functions return a value of type NhlErrorTypes, and the Fortran subroutines return the error value in *ierr*. If they are unable to add the *name/value* pair to the list, they will return **NhlFATAL**; otherwise they will return **NhlNOERROR**.

See also

- NhlCreate function
- NhlSetValues function

Copyright

Copyright 1987-1999 University Corporation for Atmospheric Research
The use of this Software is governed by a License Agreement.

NCAR Graphics is a registered trademark of the University Corporation for Atmospheric Research.

NhlRLSetMDArray

NhlRLSetMDArray, NhlRLSetMDIntegerArray, NhlRLSetMDFloatArray, NhlRLSetMDDoubleArray

The Fortran names of these functions are **NhlFRLSetMDIntegerArray**, **NhlFRLSetMDFloatArray** **NhlFRLSetMDDoubleArray**. There is no Fortran binding for **NhlRLSetMDArray**.

Used to add the given *name* and *value* array to the *ResList*.

C synopsis

```
#include <ncarg/hlu/hlu.h>

NhlErrorTypes NhlRLSetMDArray(
                int              list_id,
                NhlString        resname,
                NhlPointer       data,
                NhlString        type_element,
                unsigned int     size_element,
                int              num_dimensions,
                int              *len_dimensions
                )

NhlErrorTypes NhlRLSetMDIntegerArray(
                int              list_id,
```

```
                NhlString       resname,
                int             *data,
                int             num_dimensions,
                int             *len_dimensions
                )

NhlErrorTypes NhlRLSetMDFloatArray(
                int             list_id,
                NhlString       resname,
                float           *data,
                int             num_dimensions,
                int             *len_dimensions
                )

NhlErrorTypes NhlRLSetMDDoubleArray(
                int             list_id,
                NhlString       resname,
                double          *data,
                int             num_dimensions,
                int             *len_dimensions
                )
```

Fortran synopsis

```
      subroutine NhlFRLSetMDIntegerArray(list_id, resname, iarray,
     +                   num_dimensions, len_dimensions, ierr)
      integer list_id, iarray(*), num_dimensions, len_dimensions, ierr
      character*(*) resname

      subroutine NhlFRLSetMDFloatArray(list_id, resname, rarray,
     +                   num_dimensions, len_dimensions, ierr)
      integer list_id, num_dimensions, len_dimensions, ierr
      real rarray(*)
      character*(*) resname

      subroutine NhlFRLSetMDDoubleArray(list_id, resname, darray,
     +                   num_dimensions, len_dimensions, ierr)
      integer list_id, num_dimensions, len_dimensions, ierr
      double precision darray(*)
      character*(*) resname
```

Arguments

list_id (input)
: Id of the *ResList* to add the *name/value* pair to.

resname (input)
: Specifies the resource name to apply the given *data* to when the list is applied to an HLU object.

data (input, C only)
: Specifies the array value that will be applied to the corresponding *resname* when the list is applied to an HLU object.

type_elements (input, C only)

Specifies the type of each element of the *data* array.

size_elements (input, C only)
: Specifies the size in bytes of each element of the *data* array.

num_dimensions (input)
: Specifies the number of dimensions in the *data* array.

len_dimensions (input)
: An integer array of length *num_dimensions* that specifies the length of each dimensions in the *data* array.

iarray (input, Fortran only)
: Specifies the multi-dimensioned integer array that will be applied to the corresponding *resname* resource when the list is applied to an HLU object.

rarray (input, Fortran only)
: Specifies the multi-dimensioned real array that will be applied to the corresponding *resname* resource when the list is applied to an HLU object.

darray (input, Fortran only)
: Specifies the multi-dimensioned double precision array that will be applied to the corresponding *resname* resource when the list is applied to an HLU object.

ierr (output, Fortran only)
: Error code.

Types

```
Type name:              NhlTErrorTypes
Definition:
typedef enum _NhlErrType{
        NhlFATAL        = -4,   /* "FATAL"      */
        NhlWARNING      = -3,   /* "WARNING"    */
        NhlINFO         = -2,   /* "INFO"       */
        NhlNOERROR      = -1    /* "NOERROR"    */
} NhlErrorTypes;
```

Description

These functions can only be used on a **NhlSETRL** *ResList*. They are used to add *name/value* pairs to the *ResList* that will be passed to the NhlCreate or NhlSetValues functions. These functions need to be used when the *value* part of the *name/value* pair is an Array.

Return values

The C functions return a value of type NhlErrorTypes, and the Fortran subroutines return the error value in *ierr*. If they are unable to add the *name/value* pair to the list, they will return **NhlFATAL**; otherwise they will return **NhlNOERROR**.

See also

- NhlCreate function
- NhlSetValues function

Copyright

Copyright 1987-1999 University Corporation for Atmospheric Research
The use of this Software is governed by a License Agreement.

NCAR Graphics is a registered trademark of the University Corporation for Atmospheric Research.

NhlRLSetMDArray

NhlRLSetMDArray, NhlRLSetMDIntegerArray, NhlRLSetMDFloatArray, NhlRLSetMDDoubleArray

The Fortran names of these functions are **NhlFRLSetMDIntegerArray**, **NhlFRLSetMDFloatArray** **NhlFRLSetMDDoubleArray**. There is no Fortran binding for **NhlRLSetMDArray**.

Used to add the given *name* and *value* array to the *ResList*.

C synopsis

```
#include <ncarg/hlu/hlu.h>

NhlErrorTypes NhlRLSetMDArray(
                int              list_id,
                NhlString        resname,
                NhlPointer       data,
                NhlString        type_element,
                unsigned int     size_element,
                int              num_dimensions,
                int              *len_dimensions
                )

NhlErrorTypes NhlRLSetMDIntegerArray(
                int              list_id,
                NhlString        resname,
                int              *data,
                int              num_dimensions,
                int              *len_dimensions
                )
```

```
NhlErrorTypes NhlRLSetMDFloatArray(
                int             list_id,
                NhlString       resname,
                float           *data,
                int             num_dimensions,
                int             *len_dimensions
                )

NhlErrorTypes NhlRLSetMDDoubleArray(
                int             list_id,
                NhlString       resname,
                double          *data,
                int             num_dimensions,
                int             *len_dimensions
                )
```

Fortran synopsis

```
      subroutine NhlFRLSetMDIntegerArray(list_id, resname, iarray,
     +                   num_dimensions, len_dimensions, ierr)
      integer list_id, iarray(*), num_dimensions, len_dimensions, ierr
      character*(*) resname

      subroutine NhlFRLSetMDFloatArray(list_id, resname, rarray,
     +                   num_dimensions, len_dimensions, ierr)
      integer list_id, num_dimensions, len_dimensions, ierr
      real rarray(*)
      character*(*) resname

      subroutine NhlFRLSetMDDoubleArray(list_id, resname, darray,
     +                   num_dimensions, len_dimensions, ierr)
      integer list_id, num_dimensions, len_dimensions, ierr
      double precision darray(*)
      character*(*) resname
```

Arguments

list_id (input)
: Id of the *ResList* to add the *name/value* pair to.

resname (input)
: Specifies the resource name to apply the given *data* to when the list is applied to an HLU object.

data (input, C only)
: Specifies the array value that will be applied to the corresponding *resname* when the list is applied to an HLU object.

type_elements (input, C only)
: Specifies the type of each element of the *data* array.

size_elements (input, C only)
: Specifies the size in bytes of each element of the *data* array.

num_dimensions (input)
: Specifies the number of dimensions in the *data* array.

len_dimensions (input)
An integer array of length *num_dimensions* that specifies the length of each dimensions in the *data* array.

iarray (input, Fortran only)
Specifies the multi-dimensioned integer array that will be applied to the corresponding *resname* resource when the list is applied to an HLU object.

rarray (input, Fortran only)
Specifies the multi-dimensioned real array that will be applied to the corresponding *resname* resource when the list is applied to an HLU object.

darray (input, Fortran only)
Specifies the multi-dimensioned double precision array that will be applied to the corresponding *resname* resource when the list is applied to an HLU object.

ierr (output, Fortran only)
Error code.

Types

```
Type name:                **NhlTErrorTypes**
Definition:
typedef enum _NhlErrType{
        NhlFATAL        = -4,    /* "FATAL"      */
        NhlWARNING      = -3,    /* "WARNING"    */
        NhlINFO         = -2,    /* "INFO"       */
        NhlNOERROR      = -1     /* "NOERROR"    */
} NhlErrorTypes;
```

Description

These functions can only be used on a **NhlSETRL** *ResList*. They are used to add *name/value* pairs to the *ResList* that will be passed to the NhlCreate or NhlSetValues functions. These functions need to be used when the *value* part of the *name/value* pair is an Array.

Return values

The C functions return a value of type NhlErrorTypes, and the Fortran subroutines return the error value in *ierr*. If they are unable to add the *name/value* pair to the list, they will return **NhlFATAL**; otherwise they will return **NhlNOERROR**.

See also

- NhlCreate function
- NhlSetValues function

Copyright

Copyright 1987-1999 University Corporation for Atmospheric Research
The use of this Software is governed by a License Agreement.

NCAR Graphics is a registered trademark of the University Corporation for Atmospheric Research.

NhlRLSetMDArray

NhlRLSetMDArray, NhlRLSetMDIntegerArray, NhlRLSetMDFloatArray, NhlRLSetMDDoubleArray

The Fortran names of these functions are **NhlFRLSetMDIntegerArray**, **NhlFRLSetMDFloatArray** **NhlFRLSetMDDoubleArray**. There is no Fortran binding for **NhlRLSetMDArray**.

Used to add the given *name* and *value* array to the *ResList*.

C synopsis

```
#include <ncarg/hlu/hlu.h>

NhlErrorTypes NhlRLSetMDArray(
                int             list_id,
                NhlString       resname,
                NhlPointer      data,
                NhlString       type_element,
                unsigned int    size_element,
                int             num_dimensions,
                int             *len_dimensions
                )

NhlErrorTypes NhlRLSetMDIntegerArray(
                int             list_id,
                NhlString       resname,
                int             *data,
                int             num_dimensions,
                int             *len_dimensions
                )

NhlErrorTypes NhlRLSetMDFloatArray(
                int             list_id,
                NhlString       resname,
                float           *data,
                int             num_dimensions,
```

```
                int             *len_dimensions
                )

NhlErrorTypes NhlRLSetMDDoubleArray(
                int             list_id,
                NhlString       resname,
                double          *data,
                int             num_dimensions,
                int             *len_dimensions
                )
```

Fortran synopsis

```
      subroutine NhlFRLSetMDIntegerArray(list_id, resname, iarray,
     +                   num_dimensions, len_dimensions, ierr)
      integer list_id, iarray(*), num_dimensions, len_dimensions, ierr
      character*(*) resname

      subroutine NhlFRLSetMDFloatArray(list_id, resname, rarray,
     +                   num_dimensions, len_dimensions, ierr)
      integer list_id, num_dimensions, len_dimensions, ierr
      real rarray(*)
      character*(*) resname

      subroutine NhlFRLSetMDDoubleArray(list_id, resname, darray,
     +                   num_dimensions, len_dimensions, ierr)
      integer list_id, num_dimensions, len_dimensions, ierr
      double precision darray(*)
      character*(*) resname
```

Arguments

list_id (input)
: Id of the *ResList* to add the *name/value* pair to.

resname (input)
: Specifies the resource name to apply the given *data* to when the list is applied to an HLU object.

data (input, C only)
: Specifies the array value that will be applied to the corresponding *resname* when the list is applied to an HLU object.

type_elements (input, C only)
: Specifies the type of each element of the *data* array.

size_elements (input, C only)
: Specifies the size in bytes of each element of the *data* array.

num_dimensions (input)
: Specifies the number of dimensions in the *data* array.

len_dimensions (input)
: An integer array of length *num_dimensions* that specifies the length of each dimensions in the *data* array.

iarray (input, Fortran only)

Specifies the multi-dimensioned integer array that will be applied to the corresponding *resname* resource when the list is applied to an HLU object.

rarray (input, Fortran only)

Specifies the multi-dimensioned real array that will be applied to the corresponding *resname* resource when the list is applied to an HLU object.

darray (input, Fortran only)

Specifies the multi-dimensioned double precision array that will be applied to the corresponding *resname* resource when the list is applied to an HLU object.

ierr (output, Fortran only)

Error code.

Types

```
Type name:                NhlTErrorTypes
Definition:
typedef enum _NhlErrType{
        NhlFATAL        = -4,    /* "FATAL"       */
        NhlWARNING      = -3,    /* "WARNING"     */
        NhlINFO         = -2,    /* "INFO"        */
        NhlNOERROR      = -1     /* "NOERROR"     */
} NhlErrorTypes;
```

Description

These functions can only be used on a **NhlSETRL** *ResList*. They are used to add *name/value* pairs to the *ResList* that will be passed to the NhlCreate or NhlSetValues functions. These functions need to be used when the *value* part of the *name/value* pair is an Array.

Return values

The C functions return a value of type NhlErrorTypes, and the Fortran subroutines return the error value in *ierr*. If they are unable to add the *name/value* pair to the list, they will return **NhlFATAL**; otherwise they will return **NhlNOERROR**.

See also

- NhlCreate function
- NhlSetValues function

Copyright

Copyright 1987-1999 University Corporation for Atmospheric Research
The use of this Software is governed by a License Agreement.

NCAR Graphics is a registered trademark of the University Corporation for Atmospheric Research.

NhlRLSet

NhlRLSet, NhlRLSetInteger, NhlRLSetFloat, NhlRLSetDouble, NhlRLSetString

The Fortran names of these functions are **NhlFRLSetInteger**, **NhlFRLSetFloat**, **NhlFRLSetDouble**, **NhlFRLSetString**. There is no Fortran binding for **NhlRLSet**.

Used to add the given *name* and *value* to the *ResList*.

C synopsis

```
#include <ncarg/hlu/hlu.h>

NhlErrorTypes NhlRLSet(
                int             list_id,
                NhlString       resname,
                NhlString       type,
                <AnyType>       value
                )

NhlErrorTypes NhlRLSetInteger(
                int             list_id,
                NhlString       resname,
                int             value
                )

NhlErrorTypes NhlRLSetFloat(
                int             list_id,
                NhlString       resname,
                float           value
                )

NhlErrorTypes NhlRLSetDouble(
                int             list_id,
                NhlString       resname,
                double          value
                )

NhlErrorTypes NhlRLSetString(
                int             list_id,
```

```
            NhlString       resname,
            NhlString       value
            )
```

Fortran synopsis

```
subroutine NhlFRLSetInteger(list_id, resname, ival, ierr)
integer list_id, ival, ierr
character*(*) resname

subroutine NhlFRLSetFloat(list_id, resname, rval, ierr)
integer list_id, ierr
real rval
character*(*) resname

subroutine NhlFRLSetDouble(list_id, resname, rval, ierr)
integer list_id, ierr
double precision dval
character*(*) resname

subroutine NhlFRLSetString(list_id, resname, sval, ierr)
integer list_id, ierr
character*(*) resname, sval
```

Arguments

list_id (input)
: Id of the *ResList* to add the *name/value* pair to.

resname (input)
: Specifies the resource name to apply the given *value* to when the list is applied to an HLU object.

type (input, C only)
: Specifies the type for *value*.

value (input, C only)
: Specifies the value that will be applied to the corresponding *resname* when the list is applied to an HLU object.

ival (input, Fortran only)
: Specifies the integer value that will be applied to the corresponding *resname* resource when the list is applied to an HLU object.

rval (input, Fortran only)
: Specifies the real value that will be applied to the corresponding *resname* resource when the list is applied to an HLU object.

dval (input, Fortran only)
: Specifies the double precision value that will be applied to the corresponding *resname* resource when the list is applied to an HLU object.

sval (input, Fortran only)
: Specifies the string that will be applied to the corresponding *resname* resource when the list is applied to an HLU object.

ierr (output, Fortran only)

Error code.

Types

```
Type name:               NhlTErrorTypes
Definition:
typedef enum _NhlErrType{
        NhlFATAL        = -4,    /* "FATAL"       */
        NhlWARNING      = -3,    /* "WARNING"     */
        NhlINFO         = -2,    /* "INFO"        */
        NhlNOERROR      = -1     /* "NOERROR"     */
} NhlErrorTypes;
```

Description

These functions can only be used on a **NhlSETRL** *ResList*. They are used to add *name/value* pairs to the *ResList* that will be passed to the NhlCreate or NhlSetValues functions.

Return values

The C functions return a value of type NhlErrorTypes, and the Fortran subroutines return the error value in *ierr*. If they are unable to add the *name/val_addr* pair to the list, they will return **NhlFATAL**; otherwise they will return **NhlNOERROR**.

See also

- NhlCreate function
- NhlSetValues function

Copyright

Copyright 1987-1999 University Corporation for Atmospheric Research
The use of this Software is governed by a License Agreement.

NCAR Graphics is a registered trademark of the University Corporation for Atmospheric Research.

NhlRLSetArray

NhlRLSetArray, NhlRLSetIntegerArray, NhlRLSetFloatArray, NhlRLSetDoubleArray, NhlRLSetStringArray

The Fortran names of these functions are **NhlFRLSetIntegerArray**, **NhlFRLSetFloatArray**, **NhlFRLSetDoubleArray**, **NhlFRLSetStringArray**. There is no Fortran binding for **NhlRLSetArray**.

Used to add the given *name* and *value* array to the *ResList*.

C synopsis

```
#include <ncarg/hlu/hlu.h>

NhlErrorTypes NhlRLSetArray(
                int             list_id,
                NhlString       resname,
                NhlPointer      data,
                NhlString       type_element,
                unsigned int    size_element,
                int             num_elements
                )

NhlErrorTypes NhlRLSetIntegerArray(
                int             list_id,
                NhlString       resname,
                int             *data,
                int             num_elements
                )

NhlErrorTypes NhlRLSetFloatArray(
                int             list_id,
                NhlString       resname,
                float           *data,
                int             num_elements
                )

NhlErrorTypes NhlRLSetDoubleArray(
                int             list_id,
                NhlString       resname,
                double          *data,
                int             num_elements
                )

NhlErrorTypes NhlRLSetStringArray(
                int             list_id,
                NhlString       resname,
                NhlString       *data,
                int             num_elements
                )
```

Fortran synopsis

```
      subroutine NhlFRLSetIntegerArray(list_id, resname, iarray,
     +                                 num_elements, ierr)
      integer list_id, iarray(num_elements), num_elements, ierr
      character*(*) resname

      subroutine NhlFRLSetFloatArray(list_id, resname, rarray,
     +                               num_elements, ierr)
      integer list_id, num_elements, ierr
      real rarray(num_elements)
      character*(*) resname

      subroutine NhlFRLSetDoubleArray(list_id, resname, darray,
     +                               num_elements, ierr)
      integer list_id, num_elements, ierr
      double precision darray(num_elements)
      character*(*) resname

      subroutine NhlFRLSetStringArray(list_id, resname, sarray,
     +                                num_elements, ierr)
      integer list_id, num_elements, ierr
      character*(*) resname, sarray(num_elements)
```

Arguments

list_id (input)
: Id of the *ResList* to add the *name/value* pair to.

resname (input)
: Specifies the resource name to apply the given *data* to when the list is applied to an HLU object.

data (input, C only)
: Specifies the array value that will be applied to the corresponding *resname* when the list is applied to an HLU object.

type_elements (input, C only)
: Specifies the type of each element of the *data* array.

size_elements (input, C only)
: Specifies the size in bytes of each element of the *data* array.

num_elements (input)
: Specifies the number of elements in the *data* array.

iarray (input, Fortran only)
: Specifies the integer array that will be applied to the corresponding *resname* resource when the list is applied to an HLU object.

rarray (input, Fortran only)
: Specifies the real array that will be applied to the corresponding *resname* resource when the list is applied to an HLU object.

darray (input, Fortran only)
: Specifies the double precision array that will be applied to the corresponding *resname* resource when the list is applied to an HLU object.

sarray (input, Fortran only)
: Specifies the string array that will be applied to the corresponding *resname* resource when the list is applied to an HLU object.

ierr (output, Fortran only)
Error code.

Types

```
Type name:              NhlTErrorTypes
Definition:
typedef enum _NhlErrType{
        NhlFATAL        = -4,   /* "FATAL"      */
        NhlWARNING      = -3,   /* "WARNING"    */
        NhlINFO         = -2,   /* "INFO"       */
        NhlNOERROR      = -1    /* "NOERROR"    */
} NhlErrorTypes;
```

Description

These functions can only be used on a **NhlSETRL** *ResList*. They are used to add *name/value* pairs to the *ResList* that will be passed to the NhlCreate or NhlSetValues functions. These functions need to be used when the *value* part of the *name/value* pair is an Array.

Return values

The C functions return a value of type NhlErrorTypes, and the Fortran subroutines return the error value in *ierr*. If they are unable to add the *name/val_addr* pair to the list, they will return **NhlFATAL**; otherwise they will return **NhlNOERROR**.

See also

- NhlCreate function
- NhlSetValues function

Copyright

Copyright 1987-1999 University Corporation for Atmospheric Research
The use of this Software is governed by a License Agreement.

NCAR Graphics is a registered trademark of the University Corporation for Atmospheric Research.

NhlRLCreate

NhlRLCreate, NhlRLDestroy, NhlRLClear, NhlRLUnSet, NhlRLIsSet

The Fortran names of these functions are **NhlFRLCreate**, **NhlFRLDestroy**, **NhlFRLClear**, **NhlFRLUnSet**, and **NhlFRLIsSet**. *ResList* support functions.

C synopsis

```
#include <ncarg/hlu/hlu.h>

int NhlRLCreate(
                NhlRLType       list_type
                )

void NhlRLDestroy(
                int             list_id
                )

void NhlRLClear(
                int             list_id
                )

void NhlRLUnSet(
                int             list_id,
                NhlString       resname
                )

NhlBoolean NhlRLIsSet(
                int             list_id,
                NhlString       resname
                )
```

Fortran synopsis

```
      subroutine NhlFRLCreate(list_id, list_type)
      integer list_id
      character*(*) list_type

      subroutine nhlfrldestroy(list_id)
      integer list_id

      subroutine nhlfrlclear(list_id)
      integer list_id

      subroutine nhlfrlunset(list_id, resname)
      integer list_id
```

```
character*(*) resname

subroutine nhlfrlisset(list_id, resname, ival)
integer list_id, ival
character*(*) resname
```

Arguments

list_type (input)
: Specifies the type of ResList to create. Use NhlSETRL (or 'setrl' in Fortran) for a list being used to set values in an object (**NhlCreate** and **NhlSetValues**). Use NhlGETRL (or 'getrl' in Fortran) for a list being used to retrieve values from an object. (**NhlGetValues**)

list_id (input/output)
: Integer identifier for the *ResList*. It is returned from the **NhlRLCreate** function.

resname (input)
: Resource name. In the **NhlRLIsSet** function, this is used to determine if the given resource name has been set in the *ResList*. In the **NhlRLUnSet** function, it is used to remove the given resource name from the *ResList*.

ival (input, Fortran only)
: Flag indicating whether or not a resource specified by *resname* is in the *ResList*.

Types

```
Type name:              <none>
typedef enum NhlRLType_{
        NhlSETRL,       /* "setrl" */
        NhlGETRL        /* "getrl" */
} NhlRlType;
```

Description

The **NhlRLCreate** function is used to create a ResList. A ResList is a dynamic list that stores *name/value* pairs that are assigned to it. The **NhlRLCreate** function returns a *list_id* that is used as an argument to the Create, SetValues, and GetValues *ResList* functions, as well as the other *ResList* support functions.

The **NhlRLDestroy** function is used to *free* an existing *ResList*. The *list_id* number may be reused the next time a **NhlRLCreate** is called.

The **NhlRLClear** function is used to clear all the resource names and values out of the *ResList*. It basically empties out the list.

The **NhlRLUnSet** function is used to remove the resource name specified by the *resname* argument from the *ResList*.

The **NhlRLIsSet** function is used to find out if the resource name specified by the *resname* argument is currently in the *ResList*.

Return values

The **NhlRLCreate** C function returns a positive integer that is used to identify the *ResList*, and the **NhlFRLCreate** Fortran subroutine returns this value in *list_id*. If it returns a non-positive number, then it was unable to create a *ResList*.

The **NhlRLIsSet** function returns True if *resname* is currently contained in the *ResList* and False if it is not; the **NhlFRLIsSet** Fortran subroutine returns this value in *ival*.

See also

- Create function
- SetValues function
- GetValues function

Copyright

Copyright 1987-1999 University Corporation for Atmospheric Research
The use of this Software is governed by a License Agreement.

NCAR Graphics is a registered trademark of the University Corporation for Atmospheric Research.

NhlMalloc

NhlMalloc, NhlRealloc, NhlFree

No corresponding Fortran routines.

Memory allocation functions.

Synopsis

```
#include <ncarg/hlu/hlu.h>

void *NhlMalloc(
                unsigned int    size
                )

void *NhlRealloc(
                void            *data,
                unsigned int    size
                )

NhlErrorTypes NhlFree(
                void            *data
                )
```

Arguments

size (input)
Specifies the size of the memory to allocate.
data (input)
Specifies a pointer to memory previously returned from NhlMalloc or NhlRealloc.

Types

```
Type name:              NhlTErrorTypes
Definition:
typedef enum _NhlErrType{
        NhlFATAL        = -4,   /* "FATAL"      */
        NhlWARNING      = -3,   /* "WARNING"    */
        NhlINFO         = -2,   /* "INFO"       */
        NhlNOERROR      = -1    /* "NOERROR"    */
} NhlErrorTypes;
```

Description

These functions behave like the stdlib functions **malloc**, **realloc** and **free**.

NhlMalloc returns a pointer to a block of memory of at least *size* bytes. The argument to **NhlFree** is a pointer to a block previously allocated with **NhlMalloc**. If **NhlMalloc** is called with a *size* of 0, it returns **NULL**.

NhlFree frees the memory pointed at by *data* so the memory can be re-used. **NhlFree** must be called on all memory allocated by the HLU library on behalf of the user. This includes all arrays retrieved from objects using **NhlGetValues**. If **NhlFree** is called with a **NULL** pointer, it doesn't work.

NhlRealloc changes the size of the block of memory pointed at by *data*. It may extend the current block or allocate another block, copy the contents pointed at by *data* to the new location, and then free the old block. It returns a pointer to the new block. If *data* is **NULL**, **NhlRealloc** behaves like **NhlMalloc**.

Return values

NhlMalloc
: returns a pointer to a block of memory, or **NULL** if a *size* 0 is specified or if it can't allocate the memory.

NhlRealloc
: returns a pointer to a block of memory, or **NULL** if a *size* 0 is specified or if it can't allocate the memory.

NhlFree
: returns a value of NhlErrorTypes:
NhlNOERROR if there is no error, or NhlWARNING if there is a detectable error.

Copyright

Copyright 1987-1999 University Corporation for Atmospheric Research
The use of this Software is governed by a License Agreement.

NCAR Graphics is a registered trademark of the University Corporation for Atmospheric Research.

NhlRemoveAnnotation

The Fortran name of this function is **NhlFRemoveAnnotation**.

This function removes an arbitrary **View** object annotation from a Plot Object, destroying its **AnnoManager** object.

C Synopsis

```
#include <ncarg/hlu/hlu.h>
#include <hfile for Plot Object>

NhlErrorTypes NhlRemoveAnnotation(
                int     plot_id,
                int     anno_manager_id
                )
```

Fortran Synopsis

```
subroutine NhlFRemoveAnnotation(plot_id, anno_manager_id, ierr)
integer plot_id, anno_manager_id, ierr
```

Arguments

plot_id (input)
: Identifier of the plot object from which the **View** object annotation is to be removed.

anno_manager_id (input)
: Identifier of the **AnnoManager** for the **View** object annotation to be removed.

ierr (output, Fortran only)
: Error code.

Types

```
Type name:                **NhlTErrorTypes**
Definition:
typedef enum _NhlErrType{
        NhlFATAL        = -4,    /* "FATAL"      */
        NhlWARNING      = -3,    /* "WARNING"    */
        NhlINFO         = -2,    /* "INFO"       */
        NhlNOERROR      = -1     /* "NOERROR"    */
} NhlErrorTypes;
```

Description

Use this function to remove a user-created external annotation from a plot object. Along with the plot object's id, you must supply the identifier of the controlling **AnnoManager** object. If you have only the view id, you can always retrieve the **AnnoManager's** id by getting the value of the **View** resource *vpAnnoManagerId*. Assuming the object ids are valid, the plot object destroys the **AnnoManager** object, and removes the ids of the **View** object and the **AnnoManager** from the array resources *pmAnnoViews* and *pmAnnoManagers* This removes any further association between the **View** object and the plot object. Note that you are responsible for destroying the **View** object when you are finished with it.

Return Values

The **NhlRemoveAnnotation** C function returns a value of type **NhlErrorTypes**, and the **NhlFRemoveAnnotation** Fortran subroutine returns the error value in *ierr*. The following table indicates what the various return Error Values mean:

```
Value   |                        Meaning
--------+------------------------------------------------
NOERROR |       view succesfully removed from the plot object
--------+------------------------------------------------
INFO    |       minor recoverable error
--------+------------------------------------------------
WARNING |       recoverable error
--------+------------------------------------------------
FATAL   |       The function was unsuccessful
```

See Also

- **AnnoManager** object
- **View** object
- **PlotManager** object
- **NhlAddAnnotation** function

Copyright

Copyright 1987-1999 University Corporation for Atmospheric Research
The use of this Software is governed by a License Agreement.

NCAR Graphics is a registered trademark of the University Corporation for Atmospheric Research.

NhlRemoveData

The Fortran name of this function is **NhlFRemoveData**.

This function is used to dissociate a **DataItem** object from a data resource in a **DataComm** object.

C synopsis

```
#include <ncarg/hlu/DataComm.h>

NhlErrorTypes NhlRemoveData(
                int             dcommid,
                NhlString       res_name,
                int             ditemid
                )
```

Fortran synopsis

```
subroutine NhlFRemoveData(dcommid, res_name, ditemid, ierr)
integer dcommid, ditemid, ierr
character*(*) res_name
```

Arguments

dcommid (input)
Identifies the **DataComm** object with the data resource.
res_name (input)
Identifies the data resource to remove the *ditemid* from in the **DataComm** object.
ditemid (input)
Identifies the id of the **DataItem** or **DataSpec** object to dissociate from the *res_name* resource in the *dcommid* **DataComm** object.
ierr (output, Fortran only)
Error code.

Types

```
Type name:              NhlTErrorTypes
Definition:
typedef enum _NhlErrType{
        NhlFATAL        = -4,   /* "FATAL"      */
        NhlWARNING      = -3,   /* "WARNING"    */
        NhlINFO         = -2,   /* "INFO"       */
        NhlNOERROR      = -1    /* "NOERROR"    */
} NhlErrorTypes;
```

Description

This function is used to dissociate the given *ditemid* **DataItem** object from the given *res_name* data resource in the given *dcommid* **DataComm** object. The *ditemid* must have already been added to the *res_name* resource of the *dcommid* object with an **AddData** function first.

To allow a higher degree of configurability, it is also possible that a **DataSpec** object had been added to the *res_name* resource. If that is the case, then the **DataSpec** id must be used to dissociate the **DataItem** as well.

Return values

The **NhlRemoveData** C function returns a value of type **NhlErrorTypes**, and the **NhlFRemoveData** Fortran subroutine returns the error in *ierr*. The following table indicates what the return error values mean:

Value	Meaning
NOERROR	The **DataItem** was removed from the *res_name* data resource without any problems.
INFO	Recoverable error.
WARNING	Recoverable error.
FATAL	Unable to remove the given *dcommid* from the object. Usually indicates that the user provided an invalid *dcommid* or *res_name*. If there is problem internally, then it is possible that the *dcommid* object was destroyed to maintain the integrity of the HLU library.

See also

- NhlAddData
- DataComm object
- DataItem object

Copyright

Copyright 1987-1999 University Corporation for Atmospheric Research
The use of this Software is governed by a License Agreement.

NCAR Graphics is a registered trademark of the University Corporation for Atmospheric Research.

NhlRemoveOverlay

The Fortran name of this function is **NhlFRemoveOverlay**.

This function removes an overlay plot member from a base plot.

C Synopsis

```
#include <ncarg/hlu/hlu.h>
#include <hfile for Base Plot Object>

NhlErrorTypes NhlRemoveOverlay(
                int             base_id,
                int             overlay_id,
                NhlBoolean      restore
                )
```

Fortran Synopsis

```
      subroutine NhlFRemoveOverlay(base_id, overlay_id, restore, ierr)
      integer base_id, overlay_id, restore, ierr
```

Arguments

base_id (input)
Identifies the base plot from which the overlay is to be removed.

overlay_id (input)
Identifies the overlay plot member to be removed. If *plot_id* is equal to *base_id* all overlays are restored to their previous status as **Transform** objects.

restore (input)
If this boolean value is True, and the overlay was originally an plot object with overlays of its own, the overlay is restored to its original condition along with all its overlay member plots. Otherwise, its overlays remain behind with the base plot. Annotations belonging to the plot object are always restored.

ierr (output, Fortran only)
Error code.

Types

```
Type name:              NhlTErrorTypes
Definition:
typedef enum _NhlErrType{
        NhlFATAL        = -4,   /* "FATAL"      */
        NhlWARNING      = -3,   /* "WARNING"    */
        NhlINFO         = -2,   /* "INFO"       */
        NhlNOERROR      = -1    /* "NOERROR"    */
} NhlErrorTypes;
```

Description

Use this function to remove an overlay from a base plot. If the overlay was originally itself a base plot, you can choose to restore any overlays that originally belonged to it, or leave them behind with the base plot identified by the *base_id* parameter. Note that annotations belonging to the plot object are always returned to it.

If *base_id* and *plot_id* are the same, the effect of this function is to remove all the overlays from the base plot. In this case, the *restore* parameter has no effect. All plot objects are restored to their original state before being added to the base plot.

Return Values

The **NhlRemoveOverlay** function returns a value of type **NhlErrorTypes**, and the **NhlFRemoveOverlay** Fortran subroutine returns the error value in *ierr*. The following table indicates what the various return Error Values mean:

```
Value   |                       Meaning
--------+-------------------------------------------------
NOERROR |       overlay successfully removed from the
        |       base plot.
--------+-------------------------------------------------
INFO    |       minor recoverable error.
--------+-------------------------------------------------
WARNING |       recoverable error - the base_id may not
        |       have referred to a base plot object, or
        |       the plot_id may not currently be an overlay.
--------+-------------------------------------------------
FATAL   |       the base_id or plot_id may not be valid object
        !       ids, or a system-level error has occurred.
```

See Also

- **PlotManager** object
- **NhlAddOverlay** function

Copyright

Copyright 1987-1999 University Corporation for Atmospheric Research
The use of this Software is governed by a License Agreement.

NCAR Graphics is a registered trademark of the University Corporation for Atmospheric Research.

NhlSetColor

The Fortran name of this function is **NhlFSetColor**.

This function sets the RGB color value for a specified HLU color index.

C Synopsis

```
#include <ncarg/hlu/hlu.h>
#include <hfile for Workstation class object>

NhlErrorTypes NhlSetColor(
                int     workid,
                int     ci,
                float   red,
                float   green,
                float   blue
                )
```

Fortran Synopsis

```
      subroutine NhlFSetColor(workid, ci, red, green, blue, ierr)
      integer workid, ci, ierr
      real red, green blue
```

Arguments

workid (input)
: Integer identifier of an instance of a **Workstation** class object

ci (input)
: Integer color index in the range 1 through 255 that is to be allocated.

red (input)
: Value of the red component of the desired color in the range 0.0 to 1.0 inclusive.

green (input)
: Value of the green component of the desired color in the range 0.0 to 1.0 inclusive.

blue (input)
: Value of the blue component of the desired color in the range 0.0 to 1.0 inclusive.

ierr (output, Fortran only)
: Error code.

Types

```
Type name:               NhlTErrorTypes
Definition:
typedef enum _NhlErrType{
        NhlFATAL         = -4,    /* "FATAL"       */
        NhlWARNING       = -3,    /* "WARNING"     */
        NhlINFO          = -2,    /* "INFO"        */
        NhlNOERROR       = -1     /* "NOERROR"     */
} NhlErrorTypes;
```

Description

Use this function to allocate a specific HLU color index or redefine the color of a previously allocated index. The index must be in the range 1 through 255. Note that the HLU library does not require that allocated color indexes be contiguous. Thus, for example, using **NhlSetColor** you could allocated indexes 10, 100, and 200 without allocating any of the indexes in between. If you set the color of some feature in a plot object using an unallocated color index, the **Workstation** will using the color assigned to the foreground color (HLU color index 1, NhlFOREGROUND).

Return Values

The **NhlSetColor** C function returns a value of type **NhlErrorTypes**, and the **NhlFSetColor** Fortran subroutine returns the error in *ierr*. The following table indicates what the various return Error Values mean:

Value	Meaning
NOERROR	function successful
INFO	minor recoverable error
WARNING	recoverable error
FATAL	function failed; continuing the program may result in core dump

---

See Also

- **NhlNewColor**
- **NhlFreeColor**
- **NhlIsAllocatedColor**
- **Workstation object**
- **XWorkstation object**

- **NcgmWorkstation object**

Copyright

Copyright 1987-1999 University Corporation for Atmospheric Research
The use of this Software is governed by a License Agreement.

NCAR Graphics is a registered trademark of the University Corporation for Atmospheric Research.

NhlSetValues

The Fortran name of this function is **NhlFSetValues**.

This function is used to change the resource values of an instance of any HLU object. It is provided as an access function to objects that are Base class.

C synopsis

```
#include <ncarg/hlu/hlu.h>

NhlErrorTypes NhlSetValues(
                int             obj_id,
                int             rlist_id
                )
```

Fortran synopsis

```
      subroutine NhlFSetValues(obj_id, rlist_id, ierr)
      integer obj_id, rlist_id, ierr
```

Arguments

obj_id (input)
: Id that identifies the object. Returned from the NhlCreate function.

rlist_id (input)
: Id of a ResList that specifies what resources to modify and what values to modify them to.

ierr (output, Fortran only)
: Error code.

Types

```
Type name:              NhlTErrorTypes
Definition:
typedef enum _NhlErrType{
        NhlFATAL        = -4,   /* "FATAL"      */
        NhlWARNING      = -3,   /* "WARNING"    */
        NhlINFO         = -2,   /* "INFO"       */
        NhlNOERROR      = -1    /* "NOERROR"    */
} NhlErrorTypes;
```

Description

This function is used to change the resource values of an object after it has been created. The **NhlSetValues** function uses the ResList specified by the *rlist_id* parameter to determine what resources the programmer wants to modify, and the values to modify them to. The *rlist_id* is allocated using the NhlRLCreate function.

The NhlGetValues function can be used to retrieve the current resource values of an object.

Return values

The **NhlSetValues** C function returns a value of type **NhlErrorTypes**, and the **NhlFSetValues** Fortran subroutine returns the error in *ierr*. The following table indicates what the return error values mean:

Value	Meaning
NOERROR	Object was modified with no problems.
INFO	Error was recoverable - usually indicates that the programmer passed an invalid resource name or value.
WARNING	Error was recoverable - usually indicates that there may be some parts of the object that are not configured properly, and that a FATAL error may occur at a later time.
FATAL	An error occurred such that the object being modified had to be destroyed to maintain the integrity of the HLU library.

See also

- NhlCreate
- NhlGetValues

Copyright

Copyright 1987-1999 University Corporation for Atmospheric Research
The use of this Software is governed by a License Agreement.

NCAR Graphics is a registered trademark of the University Corporation for Atmospheric Research.

NhlUpdateData

The Fortran name of this function is **NhlFUpdateData**.

This function is used to force a **DataComm** object to update its internal state to any changes that have occurred in associated **DataItems**.

C synopsis

```
#include <ncarg/hlu/DataComm.h>

NhlErrorTypes NhlUpdateData(
                int             dcommid
                )
```

Fortran synopsis

```
      subroutine NhlFUpdateData(dcommid, ierr)
      integer dcommid, ierr
```

Arguments

dcommid (input)
  Identifies the **DataComm** object to update.
ierr (output, Fortran only)
  Error code.

Types

```
Type name:               NhlTErrorTypes
Definition:
typedef enum _NhlErrType{
        NhlFATAL        = -4,    /* "FATAL"       */
        NhlWARNING      = -3,    /* "WARNING"     */
        NhlINFO         = -2,    /* "INFO"        */
        NhlNOERROR      = -1     /* "NOERROR"     */
} NhlErrorTypes;
```

Description

This function is used to force the given **DataComm** object to update its internal state. This function is only useful if the dcDelayCompute resource of the **DataComm** object is set to true. Otherwise the **DataComm** object will already be up to date. It is not really necessary to call this function, since the **DataComm** object automatically updates itself when the NhlDraw function is called.

The **DataComm** does not get out-of-date when the application programmer makes **AddData** or **RemoveData** calls to it. It is only if the application programmer makes a **SetValues** call to an associated **DataItem** object that the **DataComm** object has the possibility of being out-of-date. For example, if the application programmer changed the **Minimum** value in an associated **DataItem** object, it is possible the data extents of the plot (**DataComm** object) may be affected. It is important that the **DataComm** object respond to these changes in the data.

Return values

The **NhlUpdateData** C function returns a value of type **NhlErrorTypes**, and the **NhlFUpdateData** Fortran subroutine returns the error in *ierr*. The following table indicates what the return error values mean:

Value	Meaning
NOERROR	The **DataComm** object was updated without any problem. (It may have already been up-to-date.)
INFO	Recoverable error.
WARNING	Recoverable error, but there may be more problems.
FATAL	Unable to update the **DataComm** object.

See also

- DataComm object
- NhlAddData
- NhlRemoveData

Copyright

Copyright 1987-1999 University Corporation for Atmospheric Research
The use of this Software is governed by a License Agreement.

NCAR Graphics is a registered trademark of the University Corporation for Atmospheric Research.

NhlUpdateWorkstation

The Fortran name of this function is **NhlFUpdateWorkstation**.

This function updates the drawing area of an instance of a **Workstation** object.

C Synopsis

```
#include <ncarg/hlu/hlu.h>
#include <hfile for Workstation class object>

NhlErrorTypes NhlUpdateWorkstation(
                int      workid
                )
```

Fortran Synopsis

```
      subroutine NhlFUpdateWorkstation(workid, ierr)
      integer workid, ierr
```

Arguments

workid (input)
    Integer identifier of an instance of a **Workstation** class object

ierr (output, Fortran only)
Error code.

Types

```
Type name:              NhlTErrorTypes
Definition:
typedef enum _NhlErrType{
        NhlFATAL        = -4,   /* "FATAL"      */
        NhlWARNING      = -3,   /* "WARNING"    */
        NhlINFO         = -2,   /* "INFO"       */
        NhlNOERROR      = -1    /* "NOERROR"    */
} NhlErrorTypes;
```

Description

After executing **NhlDraw** for one or more plot objects, use this function to update the drawing area of the **Workstation**. For the **XWorkstation** object this function makes sure the X events queue is cleared which in turn causes all pending draw commands to be performed. For the **NcgmWorkstation** object this function causes all pending draw commands to be written to the metafile.

You can replace the usual call sequence of **NhlUpdateWorkstation** followed by **NhlClearWorkstation** with a single call to **NhlFrame**.

Return Values

The **NhlUpdateWorkstation** C function returns a value of type **NhlErrorTypes**, and the **NhlFUpdateWorkstation** Fortran subroutine returns the error in *ierr*. The following table indicates what the various return Error Values mean:

Value	Meaning
NOERROR	function successful
INFO	minor recoverable error
WARNING	recoverable error
FATAL	function failed; continuing the program may result in core dump

---

See Also

- **NhlDraw**
- **NhlClearWorkstation**
- **NhlFrame**
- **Workstation object**
- **XWorkstation object**
- **NcgmWorkstation object**

Copyright

Copyright 1987-1999 University Corporation for Atmospheric Research
The use of this Software is governed by a License Agreement.

NCAR Graphics is a registered trademark of the University Corporation for Atmospheric Research.

HLU API reference page format

This module describes the layout of the reference page for the HLU APIs.

API name

This section lists the Fortran function name(s) and provides a brief overview of the API.

C Synopsis

This section lists the include files that are needed in order to use the API. It also lists the API syntax for a C program.

Fortran Synopsis

This section lists the include files that are needed in order to use the API. It also lists the API syntax for a Fortran program.

Arguments

This section lists the arguments used in the API and gives a brief description of their meaning.

Types

This section lists the relevant types that are used by the API.

Description

This section gives a detailed overview of how to use the API, and it explains the functionality that is available.

Return Values

This section lists return values from the function.

See also

This section lists related topics.

Copyright

This is the copyright notice.

Resources - alphabetically

There are hundreds of resources in the full HLU resource table. Therefore, if you click on a specific class name in the following list, you will be transferred to the appropriate starting point in the table for resources related to that class.

- AnnoManager - resources with prefix am
- App - resources with prefix app
- ContourPlot - resources with prefix cn
- CoordArrTable - resources with prefix ct
- CoordArrays - resources with prefix ca
- DataComm - resources with prefix dc
- Error - resources with prefix err
- GraphicStyle - resources with prefix gs
- IrregularTransformation - resources with prefix tr
- LabelBar - resources with prefix lb
- Legend - resources with prefix lg
- LogLinTransformation - resources with prefix tr
- MapPlot - resources with prefix mp
- MapTransformation - resources with prefix mp
- NcgmWorkstation - resources with prefix wk
- PSWorkstation - resources with prefix wk
- PlotManager - resources with prefix pm
- ScalarField - resources with prefix sf
- StreamlinePlot - resources with prefix st
- TextItem - resources with prefix tx
- TickMark - resources with prefix tm
- Title - resources with prefix ti
- Transform - resources with prefix tf

- Transformation - resources with prefix tr
- VectorField - resources with prefix vf
- VectorPlot - resources with prefix vc
- View - resources with prefix vp
- Workspace - resources with prefix ws
- Workstation - resources with prefix wk
- XWorkstation - resources with prefix wk
- XyPlot - resources with prefix xy

AnnoManager resources with prefix am

amDataXF
    NhlTFloat
amDataYF
    NhlTFloat
amJust
    NhlTJustification
amOn
    NhlTBoolean
amOrthogonalPosF
    NhlTFloat
amParallelPosF
    NhlTFloat
amResizeNotify
    NhlTBoolean
amSide
    NhlTPosition
amTrackData
    NhlTBoolean
amViewId
    NhlTObjId
amZone
    NhlTInteger

App resources with prefix app

appDefaultParent
    NhlTBoolean
appFileSuffix
    NhlTString
appResources
    NhlTStringGenArray
appSysDir
    NhlTString
appUsrDir
    NhlTString

CoordArrays resources with prefix ca

caCopyArrays
    NhlTBoolean
caXArray
    NhlTGenArray
caXCast
    NhlTcaCastMode
caXMaxV
    NhlTVariable
caXMinV
    NhlTVariable
caXMissingV
    NhlTVariable
caYArray
    NhlTGenArray
caYCast
    NhlTcaCastMode
caYMaxV
    NhlTVariable
caYMinV
    NhlTVariable
caYMissingV
    NhlTVariable

ContourPlot resources with prefix cn

cnConstFLabelAngleF
    NhlTFloat
cnConstFLabelBackgroundColor
    NhlTColorIndex
cnConstFLabelConstantSpacingF
    NhlTFloat
cnConstFLabelFont
    NhlTFont
cnConstFLabelFontAspectF
    NhlTFloat
cnConstFLabelFontColor
    NhlTColorIndex
cnConstFLabelFontHeightF
    NhlTFloat
cnConstFLabelFontQuality
    NhlTFontQuality
cnConstFLabelFontThicknessF
    NhlTFloat
cnConstFLabelFormat
    NhlTString
cnConstFLabelFuncCode
    NhlTCharacter
cnConstFLabelJust

NhlTJustification
cnConstFLabelOn
NhlTBoolean
cnConstFLabelOrthogonalPosF
NhlTFloat
cnConstFLabelParallelPosF
NhlTFloat
cnConstFLabelPerimColor
NhlTColorIndex
cnConstFLabelPerimOn
NhlTInteger
cnConstFLabelPerimSpaceF
NhlTFloat
cnConstFLabelPerimThicknessF
NhlTFloat
cnConstFLabelSide
NhlTPosition
cnConstFLabelString
NhlTString
cnConstFLabelTextDirection
NhlTTextDirection
cnConstFLabelZone
NhlTInteger
cnConstFUseInfoLabelRes
NhlTBoolean
cnExplicitLabelBarLabelsOn
NhlTBoolean
cnExplicitLegendLabelsOn
NhlTBoolean
cnExplicitLineLabelsOn
NhlTBoolean
cnFillBackgroundColor
NhlTColorIndex
cnFillColor
NhlTColorIndex
cnFillColors
NhlTColorIndexGenArray
cnFillDrawOrder
NhlTDrawOrder
cnFillOn
NhlTBoolean
cnFillPattern
NhlTFillIndex
cnFillPatterns
NhlTFillIndexGenArray
cnFillScaleF
NhlTFloat
cnFillScales

NhlTFloatGenArray
cnGridBoundPerimColor
NhlTColorIndex
cnGridBoundPerimDashPattern
NhlTDashIndex
cnGridBoundPerimOn
NhlTInteger
cnGridBoundPerimThicknessF
NhlTFloat
cnHighLabelAngleF
NhlTFloat
cnHighLabelBackgroundColor
NhlTColorIndex
cnHighLabelConstantSpacingF
NhlTFloat
cnHighLabelFont
NhlTFont
cnHighLabelFontAspectF
NhlTFloat
cnHighLabelFontColor
NhlTColorIndex
cnHighLabelFontHeightF
NhlTFloat
cnHighLabelFontQuality
NhlTFontQuality
cnHighLabelFontThicknessF
NhlTFloat
cnHighLabelFormat
NhlTString
cnHighLabelFuncCode
NhlTCharacter
cnHighLabelPerimColor
NhlTColorIndex
cnHighLabelPerimOn
NhlTInteger
cnHighLabelPerimSpaceF
NhlTFloat
cnHighLabelPerimThicknessF
NhlTFloat
cnHighLabelString
NhlTString
cnHighLabelsOn
NhlTBoolean
cnHighLowLabelOverlapMode
NhlTcnHighLowLabelOverlapMode
cnHighUseLineLabelRes
NhlTBoolean
cnInfoLabelAngleF

NhlTFloat
cnInfoLabelBackgroundColor
NhlTColorIndex
cnInfoLabelConstantSpacingF
NhlTFloat
cnInfoLabelFont
NhlTFont
cnInfoLabelFontAspectF
NhlTFloat
cnInfoLabelFontColor
NhlTColorIndex
cnInfoLabelFontHeightF
NhlTFloat
cnInfoLabelFontQuality
NhlTFontQuality
cnInfoLabelFontThicknessF
NhlTFloat
cnInfoLabelFormat
NhlTString
cnInfoLabelFuncCode
NhlTCharacter
cnInfoLabelJust
NhlTJustification
cnInfoLabelOn
NhlTBoolean
cnInfoLabelOrthogonalPosF
NhlTFloat
cnInfoLabelParallelPosF
NhlTFloat
cnInfoLabelPerimColor
NhlTColorIndex
cnInfoLabelPerimOn
NhlTInteger
cnInfoLabelPerimSpaceF
NhlTFloat
cnInfoLabelPerimThicknessF
NhlTFloat
cnInfoLabelSide
NhlTPosition
cnInfoLabelString
NhlTString
cnInfoLabelTextDirection
NhlTTextDirection
cnInfoLabelZone
NhlTInteger
cnLabelBarEndLabelsOn
NhlTBoolean
cnLabelDrawOrder

NhlTDrawOrder
cnLabelMasking
NhlTBoolean
cnLabelScaleFactorF
NhlTFloat
cnLabelScaleValueF
NhlTFloat
cnLabelScalingMode
NhlTScalingMode
cnLegendLevelFlags
NhlTcnLevelUseModeGenArray
cnLevelCount
NhlTInteger
cnLevelFlag
NhlTcnLevelUseMode
cnLevelFlags
NhlTcnLevelUseModeGenArray
cnLevelSelectionMode
NhlTLevelSelectionMode
cnLevelSpacingF
NhlTFloat
cnLevels
NhlTFloatGenArray
cnLineColor
NhlTColorIndex
cnLineColors
NhlTColorIndexGenArray
cnLineDashPattern
NhlTDashIndex
cnLineDashPatterns
NhlTDashIndexGenArray
cnLineDashSegLenF
NhlTFloat
cnLineDrawOrder
NhlTDrawOrder
cnLineLabelAngleF
NhlTFloat
cnLineLabelBackgroundColor
NhlTColorIndex
cnLineLabelConstantSpacingF
NhlTFloat
cnLineLabelFont
NhlTFont
cnLineLabelFontAspectF
NhlTFloat
cnLineLabelFontColor
NhlTColorIndex
cnLineLabelFontColors

NhlTColorIndexGenArray
cnLineLabelFontHeightF
NhlTFloat
cnLineLabelFontQuality
NhlTFontQuality
cnLineLabelFontThicknessF
NhlTFloat
cnLineLabelFormat
NhlTString
cnLineLabelFuncCode
NhlTCharacter
cnLineLabelInterval
NhlTFloat
cnLineLabelPerimColor
NhlTColorIndex
cnLineLabelPerimOn
NhlTInteger
cnLineLabelPerimSpaceF
NhlTFloat
cnLineLabelPerimThicknessF
NhlTFloat
cnLineLabelPlacementMode
NhlTcnLineLabelPlacementMode
cnLineLabelStrings
NhlTStringGenArray
cnLineLabelsOn
NhlTBoolean
cnLineThicknessF
NhlTFloat
cnLineThicknesses
NhlTFloatGenArray
cnLinesOn
NhlTBoolean
cnLowLabelAngleF
NhlTFloat
cnLowLabelBackgroundColor
NhlTColorIndex
cnLowLabelConstantSpacingF
NhlTFloat
cnLowLabelFont
NhlTFont
cnLowLabelFontAspectF
NhlTFloat
cnLowLabelFontColor
NhlTColorIndex
cnLowLabelFontHeightF
NhlTFloat
cnLowLabelFontQuality

NhlTFontQuality
cnLowLabelFontThicknessF
NhlTFloat
cnLowLabelFormat
NhlTString
cnLowLabelFuncCode
NhlTCharacter
cnLowLabelPerimColor
NhlTColorIndex
cnLowLabelPerimOn
NhlTInteger
cnLowLabelPerimSpaceF
NhlTFloat
cnLowLabelPerimThicknessF
NhlTFloat
cnLowLabelString
NhlTString
cnLowLabelsOn
NhlTBoolean
cnLowUseHighLabelRes
NhlTBoolean
cnMaxDataValueFormat
NhlTString
cnMaxLevelCount
NhlTInteger
cnMaxLevelValF
NhlTFloat
cnMaxPointDistanceF
NhlTFloat
cnMinLevelValF
NhlTFloat
cnMissingValFillColor
NhlTColorIndex
cnMissingValFillPattern
NhlTInteger
cnMissingValFillScaleF
NhlTFloat
cnMissingValPerimColor
NhlTColorIndex
cnMissingValPerimDashPattern
NhlTDashIndex
cnMissingValPerimOn
NhlTInteger
cnMissingValPerimThicknessF
NhlTFloat
cnMonoFillColor
NhlTBoolean
cnMonoFillPattern

NhlTBoolean
cnMonoFillScale
NhlTBoolean
cnMonoLevelFlag
NhlTBoolean
cnMonoLineColor
NhlTBoolean
cnMonoLineDashPattern
NhlTBoolean
cnMonoLineLabelFontColor
NhlTBoolean
cnMonoLineThickness
NhlTBoolean
cnNoDataLabelOn
NhlTBoolean
cnNoDataLabelString
NhlTString
cnOutOfRangePerimColor
NhlTColorIndex
cnOutOfRangePerimDashPattern
NhlTDashIndex
cnOutOfRangePerimOn
NhlTInteger
cnOutOfRangePerimThicknessF
NhlTFloat
cnRasterCellSizeF
NhlTFloat
cnRasterMinCellSizeF
NhlTFloat
cnRasterModeOn
NhlTBoolean
cnRasterSampleFactorF
NhlTFloat
cnRasterSmoothingOn
NhlTBoolean
cnScalarFieldData
NhlTInteger
cnSmoothingDistanceF
NhlTFloat
cnSmoothingOn
NhlTBoolean
cnSmoothingTensionF
NhlTFloat

CoordArrTable resources with prefix ct

ctCopyTables
NhlTBoolean

ctXElementSize
    NhlTInteger
ctXMaxV
    NhlTVariable
ctXMinV
    NhlTVariable
ctXMissingV
    NhlTVariable
ctXTable
    NhlTPointerGenArray
ctXTableLengths
    NhlTIntegerGenArray
ctXTableType
    NhlTString
ctYElementSize
    NhlTInteger
ctYMaxV
    NhlTVariable
ctYMinV
    NhlTVariable
ctYMissingV
    NhlTVariable
ctYTable
    NhlTPointerGenArray
ctYTableLengths
    NhlTIntegerGenArray
ctYTableType
    NhlTString

DataComm resources with prefix dc

dcDelayCompute
    NhlTBoolean

Error resources with prefix err

errBuffer
    NhlTBoolean
errFileName
    NhlTString
errLevel
    NhlTErrorTypes
errPrint
    NhlTBoolean

GraphicStyle resources with prefix gs

gsClipOn

NhlTBoolean
gsEdgeColor
NhlTColorIndex
gsEdgeDashPattern
NhlTDashIndex
gsEdgeDashSegLenF
NhlTFloat
gsEdgeThicknessF
NhlTFloat
gsEdgesOn
NhlTBoolean
gsFillBackgroundColor
NhlTColorIndex
gsFillColor
NhlTColorIndex
gsFillIndex
NhlTFillIndex
gsFillLineThicknessF
NhlTFloat
gsFillScaleF
NhlTFloat
gsFont
NhlTFont
gsFontAspectF
NhlTFloat
gsFontColor
NhlTColorIndex
gsFontHeightF
NhlTFloat
gsFontQuality
NhlTFontQuality
gsFontThicknessF
NhlTFloat
gsLineColor
NhlTColorIndex
gsLineDashPattern
NhlTDashIndex
gsLineDashSegLenF
NhlTFloat
gsLineLabelConstantSpacingF
NhlTFloat
gsLineLabelFont
NhlTFont
gsLineLabelFontAspectF
NhlTFloat
gsLineLabelFontColor
NhlTColorIndex
gsLineLabelFontHeightF

NhlTFloat
gsLineLabelFontQuality
NhlTFontQuality
gsLineLabelFontThicknessF
NhlTFloat
gsLineLabelFuncCode
NhlTCharacter
gsLineLabelString
NhlTString
gsLineThicknessF
NhlTFloat
gsMarkerColor
NhlTColorIndex
gsMarkerIndex
NhlTMarkerIndex
gsMarkerSizeF
NhlTFloat
gsMarkerThicknessF
NhlTFloat
gsTextAngleF
NhlTFloat
gsTextConstantSpacingF
NhlTFloat
gsTextDirection
NhlTTextDirection
gsTextFuncCode
NhlTCharacter
gsTextJustification
NhlTJustification

LabelBar resources with prefix lb

lbAutoManage
NhlTBoolean
lbBottomMarginF
NhlTFloat
lbBoxCount
NhlTInteger
lbBoxFractions
NhlTFloatGenArray
lbBoxLineColor
NhlTInteger
lbBoxLineDashPattern
NhlTInteger
lbBoxLineDashSegLenF
NhlTFloat
lbBoxLineThicknessF
NhlTFloat

lbBoxLinesOn
NhlTBoolean
lbBoxMajorExtentF
NhlTFloat
lbBoxMinorExtentF
NhlTFloat
lbBoxSizing
NhlTlbBoxSizingMode
lbFillBackground
NhlTInteger
lbFillColor
NhlTColorIndex
lbFillColors
NhlTColorIndexGenArray
lbFillLineThicknessF
NhlTFloat
lbFillPattern
NhlTFillIndex
lbFillPatterns
NhlTFillIndexGenArray
lbFillScaleF
NhlTFloat
lbFillScales
NhlTFloatGenArray
lbJustification
NhlTJustification
lbLabelAlignment
NhlTlbLabelAlignmentMode
lbLabelAngleF
NhlTFloat
lbLabelBarOn
NhlTBoolean
lbLabelConstantSpacingF
NhlTFloat
lbLabelDirection
NhlTTextDirection
lbLabelFont
NhlTFont
lbLabelFontAspectF
NhlTFloat
lbLabelFontColor
NhlTInteger
lbLabelFontHeightF
NhlTFloat
lbLabelFontQuality
NhlTFontQuality
lbLabelFontThicknessF
NhlTFloat

lbLabelFuncCode
    NhlTCharacter
lbLabelJust
    NhlTJustification
lbLabelOffsetF
    NhlTFloat
lbLabelPosition
    NhlTPosition
lbLabelStride
    NhlTInteger
lbLabelStrings
    NhlTStringGenArray
lbLabelsOn
    NhlTBoolean
lbLeftMarginF
    NhlTFloat
lbMonoFillColor
    NhlTBoolean
lbMonoFillPattern
    NhlTBoolean
lbMonoFillScale
    NhlTBoolean
lbOrientation
    NhlTOrientation
lbPerimColor
    NhlTInteger
lbPerimDashPattern
    NhlTInteger
lbPerimDashSegLenF
    NhlTFloat
lbPerimFill
    NhlTInteger
lbPerimFillColor
    NhlTInteger
lbPerimOn
    NhlTBoolean
lbPerimThicknessF
    NhlTFloat
lbRightMarginF
    NhlTFloat
lbTitleAngleF
    NhlTFloat
lbTitleConstantSpacingF
    NhlTFloat
lbTitleDirection
    NhlTTextDirection
lbTitleExtentF
    NhlTFloat

lbTitleFont
NhlTFont
lbTitleFontAspectF
NhlTFloat
lbTitleFontColor
NhlTInteger
lbTitleFontHeightF
NhlTFloat
lbTitleFontQuality
NhlTFontQuality
lbTitleFontThicknessF
NhlTFloat
lbTitleFuncCode
NhlTCharacter
lbTitleJust
NhlTJustification
lbTitleOffsetF
NhlTFloat
lbTitleOn
NhlTBoolean
lbTitlePosition
NhlTPosition
lbTitleString
NhlTString
lbTopMarginF
NhlTFloat

Legend resources with prefix lg

lgAutoManage
NhlTBoolean
lgBottomMarginF
NhlTFloat
lgBoxBackground
NhlTColorIndex
lgBoxLineColor
NhlTColorIndex
lgBoxLineDashPattern
NhlTDashIndex
lgBoxLineDashSegLenF
NhlTFloat
lgBoxLineThicknessF
NhlTFloat
lgBoxLinesOn
NhlTBoolean
lgBoxMajorExtentF
NhlTFloat
lgBoxMinorExtentF

NhlTFloat
lgDashIndex
NhlTDashIndex
lgDashIndexes
NhlTDashIndexGenArray
lgItemCount
NhlTInteger
lgItemPlacement
NhlTlgItemPlacementMode
lgItemPositions
NhlTFloatGenArray
lgItemType
NhlTMarkLineMode
lgItemTypes
NhlTMarkLineModeGenArray
lgJustification
NhlTJustification
lgLabelAlignment
NhlTlgLabelAlignmentMode
lgLabelAngleF
NhlTFloat
lgLabelConstantSpacingF
NhlTFloat
lgLabelDirection
NhlTTextDirection
lgLabelFont
NhlTFont
lgLabelFontAspectF
NhlTFloat
lgLabelFontColor
NhlTColorIndex
lgLabelFontHeightF
NhlTFloat
lgLabelFontQuality
NhlTFontQuality
lgLabelFontThicknessF
NhlTFloat
lgLabelFuncCode
NhlTCharacter
lgLabelJust
NhlTJustification
lgLabelOffsetF
NhlTFloat
lgLabelPosition
NhlTPosition
lgLabelStride
NhlTInteger
lgLabelStrings

NhlTStringGenArray
lgLabelsOn
NhlTBoolean
lgLeftMarginF
NhlTFloat
lgLegendOn
NhlTBoolean
lgLineColor
NhlTColorIndex
lgLineColors
NhlTColorIndexGenArray
lgLineDashSegLenF
NhlTFloat
lgLineDashSegLens
NhlTFloatGenArray
lgLineLabelConstantSpacingF
NhlTFloat
lgLineLabelFont
NhlTFont
lgLineLabelFontAspectF
NhlTFloat
lgLineLabelFontColor
NhlTColorIndex
lgLineLabelFontColors
NhlTColorIndexGenArray
lgLineLabelFontHeightF
NhlTFloat
lgLineLabelFontHeights
NhlTFloatGenArray
lgLineLabelFontQuality
NhlTFontQuality
lgLineLabelFontThicknessF
NhlTFloat
lgLineLabelFuncCode
NhlTCharacter
lgLineLabelStrings
NhlTStringGenArray
lgLineLabelsOn
NhlTBoolean
lgLineThicknessF
NhlTFloat
lgLineThicknesses
NhlTFloatGenArray
lgMarkerColor
NhlTColorIndex
lgMarkerColors
NhlTColorIndexGenArray
lgMarkerIndex

NhlTMarkerIndex
lgMarkerIndexes
NhlTMarkerIndexGenArray
lgMarkerSizeF
NhlTFloat
lgMarkerSizes
NhlTFloatGenArray
lgMarkerThicknessF
NhlTFloat
lgMarkerThicknesses
NhlTFloatGenArray
lgMonoDashIndex
NhlTBoolean
lgMonoItemType
NhlTBoolean
lgMonoLineColor
NhlTBoolean
lgMonoLineDashSegLen
NhlTBoolean
lgMonoLineLabelFontColor
NhlTBoolean
lgMonoLineLabelFontHeight
NhlTBoolean
lgMonoLineThickness
NhlTBoolean
lgMonoMarkerColor
NhlTBoolean
lgMonoMarkerIndex
NhlTBoolean
lgMonoMarkerSize
NhlTBoolean
lgMonoMarkerThickness
NhlTBoolean
lgOrientation
NhlTOrientation
lgPerimColor
NhlTColorIndex
lgPerimDashPattern
NhlTDashIndex
lgPerimDashSegLenF
NhlTFloat
lgPerimFill
NhlTFillIndex
lgPerimFillColor
NhlTColorIndex
lgPerimOn
NhlTBoolean
lgPerimThicknessF

NhlTFloat
lgRightMarginF
NhlTFloat
lgTitleAngleF
NhlTFloat
lgTitleConstantSpacingF
NhlTFloat
lgTitleDirection
NhlTTextDirection
lgTitleExtentF
NhlTFloat
lgTitleFont
NhlTFont
lgTitleFontAspectF
NhlTFloat
lgTitleFontColor
NhlTColorIndex
lgTitleFontHeightF
NhlTFloat
lgTitleFontQuality
NhlTFontQuality
lgTitleFontThicknessF
NhlTFloat
lgTitleFuncCode
NhlTCharacter
lgTitleJust
NhlTJustification
lgTitleOffsetF
NhlTFloat
lgTitleOn
NhlTBoolean
lgTitlePosition
NhlTPosition
lgTitleString
NhlTString
lgTopMarginF
NhlTFloat

MapPlot resources with prefix mp

mpAreaGroupCount
NhlTInteger
mpAreaMaskingOn
NhlTBoolean
mpAreaNames
NhlTStringGenArray
mpAreaTypes
NhlTIntegerGenArray

mpDataBaseVersion
    NhlTMapDataBaseVersion
mpDefaultFillColor
    NhlTColorIndex
mpDefaultFillPattern
    NhlTFillIndex
mpDefaultFillScaleF
    NhlTFloat
mpDynamicAreaGroups
    NhlTIntegerGenArray
mpFillAreaSpecifiers
    NhlTStringGenArray
mpFillBoundarySets
    NhlTMapBoundarySets
mpFillColor
    NhlTColorIndex
mpFillColors
    NhlTColorIndexGenArray
mpFillDrawOrder
    NhlTDrawOrder
mpFillOn
    NhlTBoolean
mpFillPattern
    NhlTFillIndex
mpFillPatternBackground
    NhlTColorIndex
mpFillPatterns
    NhlTFillIndexGenArray
mpFillScaleF
    NhlTFloat
mpFillScales
    NhlTFloatGenArray
mpFixedAreaGroups
    NhlTIntegerGenArray
mpGeophysicalLineColor
    NhlTColorIndex
mpGeophysicalLineDashPattern
    NhlTDashIndex
mpGeophysicalLineDashSegLenF
    NhlTFloat
mpGeophysicalLineThicknessF
    NhlTFloat
mpGridAndLimbDrawOrder
    NhlTDrawOrder
mpGridAndLimbOn
    NhlTBoolean
mpGridLatSpacingF
    NhlTFloat

mpGridLineColor
    NhlTColorIndex
mpGridLineDashPattern
    NhlTDashIndex
mpGridLineDashSegLenF
    NhlTFloat
mpGridLineThicknessF
    NhlTFloat
mpGridLonSpacingF
    NhlTFloat
mpGridMaskMode
    NhlTMapGridMaskMode
mpGridMaxLatF
    NhlTFloat
mpGridPolarLonSpacingF
    NhlTFloat
mpGridSpacingF
    NhlTFloat
mpInlandWaterFillColor
    NhlTColorIndex
mpInlandWaterFillPattern
    NhlTFillIndex
mpInlandWaterFillScaleF
    NhlTFloat
mpLabelDrawOrder
    NhlTDrawOrder
mpLabelFontColor
    NhlTColorIndex
mpLabelFontHeightF
    NhlTFloat
mpLabelsOn
    NhlTBoolean
mpLandFillColor
    NhlTColorIndex
mpLandFillPattern
    NhlTFillIndex
mpLandFillScaleF
    NhlTFloat
mpLimbLineColor
    NhlTColorIndex
mpLimbLineDashPattern
    NhlTDashIndex
mpLimbLineDashSegLenF
    NhlTFloat
mpLimbLineThicknessF
    NhlTFloat
mpMaskAreaSpecifiers
    NhlTStringGenArray

mpMonoFillColor
NhlTBoolean
mpMonoFillPattern
NhlTBoolean
mpMonoFillScale
NhlTBoolean
mpNationalLineColor
NhlTColorIndex
mpNationalLineDashPattern
NhlTDashIndex
mpNationalLineDashSegLenF
NhlTFloat
mpNationalLineThicknessF
NhlTFloat
mpOceanFillColor
NhlTColorIndex
mpOceanFillPattern
NhlTFillIndex
mpOceanFillScaleF
NhlTFloat
mpOutlineBoundarySets
NhlTMapBoundarySets
mpOutlineDrawOrder
NhlTDrawOrder
mpOutlineOn
NhlTBoolean
mpOutlineSpecifiers
NhlTStringGenArray
mpPerimDrawOrder
NhlTDrawOrder
mpPerimLineColor
NhlTColorIndex
mpPerimLineDashPattern
NhlTDashIndex
mpPerimLineDashSegLenF
NhlTFloat
mpPerimLineThicknessF
NhlTFloat
mpPerimOn
NhlTBoolean
mpShapeMode
NhlTMapShapeMode
mpSpecifiedFillColors
NhlTColorIndexGenArray
mpSpecifiedFillDirectIndexing
NhlTBoolean
mpSpecifiedFillPatterns
NhlTFillIndexGenArray

mpSpecifiedFillPriority
NhlTSpecifiedFillPriority
mpSpecifiedFillScales
NhlTFloatGenArray
mpUSStateLineColor
NhlTColorIndex
mpUSStateLineDashPattern
NhlTDashIndex
mpUSStateLineDashSegLenF
NhlTFloat
mpUSStateLineThicknessF
NhlTFloat

MapTransformation resources with prefix mp

mpBottomAngleF
NhlTFloat
mpBottomMapPosF
NhlTFloat
mpBottomNDCF
NhlTFloat
mpBottomNPCF
NhlTFloat
mpBottomPointLatF
NhlTFloat
mpBottomPointLonF
NhlTFloat
mpBottomWindowF
NhlTFloat
mpCenterLatF
NhlTFloat
mpCenterLonF
NhlTFloat
mpCenterRotF
NhlTFloat
mpEllipticalBoundary
NhlTBoolean
mpGreatCircleLinesOn
NhlTBoolean
mpLambertMeridianF
NhlTFloat
mpLambertParallel1F
NhlTFloat
mpLambertParallel2F
NhlTFloat
mpLeftAngleF
NhlTFloat
mpLeftCornerLatF

NhlTFloat
mpLeftCornerLonF
NhlTFloat
mpLeftMapPosF
NhlTFloat
mpLeftNDCF
NhlTFloat
mpLeftNPCF
NhlTFloat
mpLeftPointLatF
NhlTFloat
mpLeftPointLonF
NhlTFloat
mpLeftWindowF
NhlTFloat
mpLimitMode
NhlTMapLimitMode
mpMaxLatF
NhlTFloat
mpMaxLonF
NhlTFloat
mpMinLatF
NhlTFloat
mpMinLonF
NhlTFloat
mpProjection
NhlTProjection
mpRelativeCenterLat
NhlTBoolean
mpRelativeCenterLon
NhlTBoolean
mpRightAngleF
NhlTFloat
mpRightCornerLatF
NhlTFloat
mpRightCornerLonF
NhlTFloat
mpRightMapPosF
NhlTFloat
mpRightNDCF
NhlTFloat
mpRightNPCF
NhlTFloat
mpRightPointLatF
NhlTFloat
mpRightPointLonF
NhlTFloat
mpRightWindowF

NhlTFloat
mpSatelliteAngle1F
NhlTFloat
mpSatelliteAngle2F
NhlTFloat
mpSatelliteDistF
NhlTFloat
mpTopAngleF
NhlTFloat
mpTopMapPosF
NhlTFloat
mpTopNDCF
NhlTFloat
mpTopNPCF
NhlTFloat
mpTopPointLatF
NhlTFloat
mpTopPointLonF
NhlTFloat
mpTopWindowF
NhlTFloat

PlotManager resources with prefix pm

pmAnnoManagers
NhlTObjIdGenArray
pmAnnoViews
NhlTObjIdGenArray
pmLabelBarDisplayMode
NhlTAnnotationDisplayMode
pmLabelBarHeightF
NhlTFloat
pmLabelBarOrthogonalPosF
NhlTFloat
pmLabelBarParallelPosF
NhlTFloat
pmLabelBarSide
NhlTJustification
pmLabelBarWidthF
NhlTFloat
pmLabelBarZone
NhlTInteger
pmLegendDisplayMode
NhlTAnnotationDisplayMode
pmLegendHeightF
NhlTFloat
pmLegendOrthogonalPosF
NhlTFloat

pmLegendParallelPosF
    NhlTFloat
pmLegendSide
    NhlTPosition
pmLegendWidthF
    NhlTFloat
pmLegendZone
    NhlTInteger
pmOverlaySequenceIds
    NhlTObjIdGenArray
pmTickMarkDisplayMode
    NhlTAnnotationDisplayMode
pmTickMarkZone
    NhlTInteger
pmTitleDisplayMode
    NhlTAnnotationDisplayMode
pmTitleZone
    NhlTInteger

ScalarField resources with prefix sf

sfCopyData
    NhlTBoolean
sfDataArray
    NhlTGenArray
sfDataMaxV
    NhlTVariable
sfDataMinV
    NhlTVariable
sfExchangeDimensions
    NhlTBoolean
sfMissingValueV
    NhlTVariable
sfXArray
    NhlTGenArray
sfXCActualEndF
    NhlTFloat
sfXCActualStartF
    NhlTFloat
sfXCEndIndex
    NhlTInteger
sfXCEndSubsetV
    NhlTVariable
sfXCEndV
    NhlTVariable
sfXCStartIndex
    NhlTInteger
sfXCStartSubsetV

NhlTVariable
sfXCStartV
NhlTVariable
sfXCStride
NhlTInteger
sfYArray
NhlTGenArray
sfYCActualEndF
NhlTFloat
sfYCActualStartF
NhlTFloat
sfYCEndIndex
NhlTInteger
sfYCEndSubsetV
NhlTVariable
sfYCEndV
NhlTVariable
sfYCStartIndex
NhlTInteger
sfYCStartSubsetV
NhlTVariable
sfYCStartV
NhlTVariable
sfYCStride
NhlTInteger

StreamlinePlot resources with prefix st

stArrowLengthF
NhlTFloat
stArrowStride
NhlTInteger
stCrossoverCheckCount
NhlTInteger
stLengthCheckCount
NhlTInteger
stLineColor
NhlTInteger
stLineStartStride
NhlTInteger
stLineThicknessF
NhlTFloat
stMapDirection
NhlTBoolean
stMinArrowSpacingF
NhlTFloatGenArray
stMinLineSpacingF
NhlTFloatGenArray

stMinStepFactorF
NhlTFloatGenArray
stNoDataLabelOn
NhlTBoolean
stNoDataLabelString
NhlTString
stStepSizeF
NhlTFloat
stStreamlineDrawOrder
NhlTDrawOrder
stVectorFieldData
NhlTInteger
stZeroFLabelAngleF
NhlTFloat
stZeroFLabelBackgroundColor
NhlTColorIndex
stZeroFLabelConstantSpacingF
NhlTFloat
stZeroFLabelFont
NhlTFont
stZeroFLabelFontAspectF
NhlTFloat
stZeroFLabelFontColor
NhlTColorIndex
stZeroFLabelFontHeightF
NhlTFloat
stZeroFLabelFontQuality
NhlTFontQuality
stZeroFLabelFontThicknessF
NhlTFloat
stZeroFLabelFormat
NhlTString
stZeroFLabelFuncCode
NhlTCharacter
stZeroFLabelJust
NhlTJustification
stZeroFLabelOn
NhlTBoolean
stZeroFLabelOrthogonalPosF
NhlTFloat
stZeroFLabelParallelPosF
NhlTFloat
stZeroFLabelPerimColor
NhlTColorIndex
stZeroFLabelPerimOn
NhlTBoolean
stZeroFLabelPerimSpaceF
NhlTFloat

stZeroFLabelPerimThicknessF
    NhlTFloat
stZeroFLabelSide
    NhlTPosition
stZeroFLabelString
    NhlTString
stZeroFLabelTextDirection
    NhlTTextDirection
stZeroFLabelZone
    NhlTInteger

Transform resources with prefix tf

tfDoNDCOverlay
    NhlTBoolean
tfPlotManagerOn
    NhlTBoolean

Title resources with prefix ti

tiDeltaF
    NhlTFloat
tiMainAngleF
    NhlTFloat
tiMainConstantSpacingF
    NhlTFloat
tiMainDirection
    NhlTTextDirection
tiMainFont
    NhlTFont
tiMainFontAspectF
    NhlTFloat
tiMainFontColor
    NhlTColorIndex
tiMainFontHeightF
    NhlTFloat
tiMainFontQuality
    NhlTFontQuality
tiMainFontThicknessF
    NhlTFloat
tiMainFuncCode
    NhlTCharacter
tiMainJust
    NhlTJustification
tiMainOffsetXF
    NhlTFloat
tiMainOffsetYF
    NhlTFloat

tiMainOn
NhlTBoolean
tiMainPosition
NhlTPosition
tiMainSide
NhlTPosition
tiMainString
NhlTString
tiUseMainAttributes
NhlTBoolean
tiXAxisAngleF
NhlTFloat
tiXAxisConstantSpacingF
NhlTFloat
tiXAxisDirection
NhlTTextDirection
tiXAxisFont
NhlTFont
tiXAxisFontAspectF
NhlTFloat
tiXAxisFontColor
NhlTColorIndex
tiXAxisFontHeightF
NhlTFloat
tiXAxisFontQuality
NhlTFontQuality
tiXAxisFontThicknessF
NhlTFloat
tiXAxisFuncCode
NhlTCharacter
tiXAxisJust
NhlTColorIndex
tiXAxisOffsetXF
NhlTFloat
tiXAxisOffsetYF
NhlTFloat
tiXAxisOn
NhlTBoolean
tiXAxisPosition
NhlTPosition
tiXAxisSide
NhlTPosition
tiXAxisString
NhlTString
tiYAxisAngleF
NhlTFloat
tiYAxisConstantSpacingF
NhlTFloat

tiYAxisDirection
NhlTTextDirection
tiYAxisFont
NhlTFont
tiYAxisFontAspectF
NhlTFloat
tiYAxisFontColor
NhlTColorIndex
tiYAxisFontHeightF
NhlTFloat
tiYAxisFontQuality
NhlTFontQuality
tiYAxisFontThicknessF
NhlTFloat
tiYAxisFuncCode
NhlTCharacter
tiYAxisJust
NhlTJustification
tiYAxisOffsetXF
NhlTFloat
tiYAxisOffsetYF
NhlTFloat
tiYAxisOn
NhlTBoolean
tiYAxisPosition
NhlTPosition
tiYAxisSide
NhlTPosition
tiYAxisString
NhlTString

TickMark resources with prefix tm

tmBorderLineColor
NhlTColorIndex
tmBorderThicknessF
NhlTFloat
tmSciNoteCutoff
NhlTInteger
tmXBAutoPrecision
NhlTBoolean
tmXBBorderOn
NhlTBoolean
tmXBDataLeftF
NhlTFloat
tmXBDataRightF
NhlTFloat
tmXBFormat

NhlTString
tmXBIrrTensionF
NhlTFloat
tmXBIrregularPoints
NhlTFloatGenArray
tmXBLabelAngleF
NhlTFloat
tmXBLabelConstantSpacingF
NhlTFloat
tmXBLabelDeltaF
NhlTFloat
tmXBLabelDirection
NhlTTextDirection
tmXBLabelFont
NhlTFont
tmXBLabelFontAspectF
NhlTFloat
tmXBLabelFontColor
NhlTColorIndex
tmXBLabelFontHeightF
NhlTFloat
tmXBLabelFontQuality
NhlTFontQuality
tmXBLabelFontThicknessF
NhlTFloat
tmXBLabelFuncCode
NhlTCharacter
tmXBLabelJust
NhlTJustification
tmXBLabelStride
NhlTInteger
tmXBLabels
NhlTStringGenArray
tmXBLabelsOn
NhlTBoolean
tmXBMajorLengthF
NhlTFloat
tmXBMajorLineColor
NhlTColorIndex
tmXBMajorOutwardLengthF
NhlTFloat
tmXBMajorThicknessF
NhlTFloat
tmXBMaxTicks
NhlTInteger
tmXBMinorLengthF
NhlTFloat
tmXBMinorLineColor

NhlTColorIndex
tmXBMinorOn
NhlTBoolean
tmXBMinorOutwardLengthF
NhlTFloat
tmXBMinorPerMajor
NhlTInteger
tmXBMinorThicknessF
NhlTFloat
tmXBMinorValues
NhlTFloatGenArray
tmXBMode
NhlTTickMarkMode
tmXBOn
NhlTBoolean
tmXBPrecision
NhlTInteger
tmXBStyle
NhlTTickMarkStyle
tmXBTickEndF
NhlTFloat
tmXBTickSpacingF
NhlTFloat
tmXBTickStartF
NhlTFloat
tmXBValues
NhlTFloatGenArray
tmXMajorGrid
NhlTBoolean
tmXMajorGridLineColor
NhlTColorIndex
tmXMajorGridLineDashPattern
NhlTDashIndex
tmXMajorGridThicknessF
NhlTFloat
tmXMinorGrid
NhlTBoolean
tmXMinorGridLineColor
NhlTColorIndex
tmXMinorGridLineDashPattern
NhlTDashIndex
tmXMinorGridThicknessF
NhlTFloat
tmXTAutoPrecision
NhlTBoolean
tmXTBorderOn
NhlTBoolean
tmXTDataLeftF

NhlTFloat
tmXTDataRightF
NhlTFloat
tmXTFormat
NhlTString
tmXTIrrTensionF
NhlTFloat
tmXTIrregularPoints
NhlTFloatGenArray
tmXTLabelAngleF
NhlTFloat
tmXTLabelConstantSpacingF
NhlTFloat
tmXTLabelDeltaF
NhlTFloat
tmXTLabelDirection
NhlTTextDirection
tmXTLabelFont
NhlTFont
tmXTLabelFontAspectF
NhlTFloat
tmXTLabelFontColor
NhlTColorIndex
tmXTLabelFontHeightF
NhlTFloat
tmXTLabelFontQuality
NhlTFontQuality
tmXTLabelFontThicknessF
NhlTFloat
tmXTLabelFuncCode
NhlTCharacter
tmXTLabelJust
NhlTJustification
tmXTLabelStride
NhlTInteger
tmXTLabels
NhlTStringGenArray
tmXTLabelsOn
NhlTBoolean
tmXTMajorLengthF
NhlTFloat
tmXTMajorLineColor
NhlTColorIndex
tmXTMajorOutwardLengthF
NhlTFloat
tmXTMajorThicknessF
NhlTFloat
tmXTMaxTicks

NhlTInteger
tmXTMinorLengthF
NhlTFloat
tmXTMinorLineColor
NhlTColorIndex
tmXTMinorOn
NhlTBoolean
tmXTMinorOutwardLengthF
NhlTFloat
tmXTMinorPerMajor
NhlTInteger
tmXTMinorThicknessF
NhlTFloat
tmXTMinorValues
NhlTFloatGenArray
tmXTMode
NhlTTickMarkMode
tmXTOn
NhlTBoolean
tmXTPrecision
NhlTInteger
tmXTStyle
NhlTTickMarkStyle
tmXTTickEndF
NhlTFloat
tmXTTickSpacingF
NhlTFloat
tmXTTickStartF
NhlTFloat
tmXTValues
NhlTFloatGenArray
tmXUseBottom
NhlTBoolean
tmYLAutoPrecision
NhlTBoolean
tmYLBorderOn
NhlTBoolean
tmYLDataBottomF
NhlTFloat
tmYLDataTopF
NhlTFloat
tmYLFormat
NhlTString
tmYLIrrTensionF
NhlTFloat
tmYLIrregularPoints
NhlTFloatGenArray
tmYLLabelAngleF

NhlTFloat
tmYLLabelConstantSpacingF
NhlTFloat
tmYLLabelDeltaF
NhlTFloat
tmYLLabelDirection
NhlTTextDirection
tmYLLabelFont
NhlTFont
tmYLLabelFontAspectF
NhlTFloat
tmYLLabelFontColor
NhlTColorIndex
tmYLLabelFontHeightF
NhlTFloat
tmYLLabelFontQuality
NhlTFontQuality
tmYLLabelFontThicknessF
NhlTFloat
tmYLLabelFuncCode
NhlTCharacter
tmYLLabelJust
NhlTJustification
tmYLLabelStride
NhlTInteger
tmYLLabels
NhlTStringGenArray
tmYLLabelsOn
NhlTBoolean
tmYLMajorLengthF
NhlTFloat
tmYLMajorLineColor
NhlTColorIndex
tmYLMajorOutwardLengthF
NhlTFloat
tmYLMajorThicknessF
NhlTFloat
tmYLMaxTicks
NhlTInteger
tmYLMinorLengthF
NhlTFloat
tmYLMinorLineColor
NhlTColorIndex
tmYLMinorOn
NhlTBoolean
tmYLMinorOutwardLengthF
NhlTFloat
tmYLMinorPerMajor

NhlTInteger
tmYLMinorThicknessF
NhlTFloat
tmYLMinorValues
NhlTFloatGenArray
tmYLMode
NhlTTickMarkMode
tmYLOn
NhlTBoolean
tmYLPrecision
NhlTInteger
tmYLStyle
NhlTTickMarkStyle
tmYLTickEndF
NhlTFloat
tmYLTickSpacingF
NhlTFloat
tmYLTickStartF
NhlTFloat
tmYLValues
NhlTFloatGenArray
tmYMajorGrid
NhlTBoolean
tmYMajorGridLineColor
NhlTColorIndex
tmYMajorGridLineDashPattern
NhlTDashIndex
tmYMajorGridThicknessF
NhlTFloat
tmYMinorGrid
NhlTBoolean
tmYMinorGridLineColor
NhlTColorIndex
tmYMinorGridLineDashPattern
NhlTDashIndex
tmYMinorGridThicknessF
NhlTFloat
tmYRAutoPrecision
NhlTBoolean
tmYRBorderOn
NhlTBoolean
tmYRDataBottomF
NhlTFloat
tmYRDataTopF
NhlTFloat
tmYRFormat
NhlTString
tmYRIrrTensionF

NhlTFloat
tmYRIrregularPoints
NhlTFloatGenArray
tmYRLabelAngleF
NhlTFloat
tmYRLabelConstantSpacingF
NhlTFloat
tmYRLabelDeltaF
NhlTFloat
tmYRLabelDirection
NhlTTextDirection
tmYRLabelFont
NhlTFont
tmYRLabelFontAspectF
NhlTFloat
tmYRLabelFontColor
NhlTColorIndex
tmYRLabelFontHeightF
NhlTFloat
tmYRLabelFontQuality
NhlTFontQuality
tmYRLabelFontThicknessF
NhlTFloat
tmYRLabelFuncCode
NhlTCharacter
tmYRLabelJust
NhlTJustification
tmYRLabelStride
NhlTInteger
tmYRLabels
NhlTStringGenArray
tmYRLabelsOn
NhlTBoolean
tmYRMajorLengthF
NhlTFloat
tmYRMajorLineColor
NhlTColorIndex
tmYRMajorOutwardLengthF
NhlTFloat
tmYRMajorThicknessF
NhlTFloat
tmYRMaxTicks
NhlTInteger
tmYRMinorLengthF
NhlTFloat
tmYRMinorLineColor
NhlTColorIndex
tmYRMinorOn

NhlTBoolean
tmYRMinorOutwardLengthF
NhlTFloat
tmYRMinorPerMajor
NhlTInteger
tmYRMinorThicknessF
NhlTFloat
tmYRMinorValues
NhlTFloatGenArray
tmYRMode
NhlTTickMarkMode
tmYROn
NhlTBoolean
tmYRPrecision
NhlTInteger
tmYRStyle
NhlTTickMarkStyle
tmYRTickEndF
NhlTFloat
tmYRTickSpacingF
NhlTFloat
tmYRTickStartF
NhlTFloat
tmYRValues
NhlTFloatGenArray
tmYUseLeft
NhlTBoolean

IrregularTransformation resources with prefix tr

trXAxisType
NhlTAxisType
trXCoordPoints
NhlTFloatGenArray
trXInterPoints
NhlTFloatGenArray
trXSamples
NhlTInteger
trXTensionF
NhlTFloat
trYAxisType
NhlTAxisType
trYCoordPoints
NhlTFloatGenArray
trYInterPoints
NhlTFloatGenArray
trYSamples
NhlTInteger

trYTensionF
    NhlTFloat

LogLinTransformation resources with prefix tr

trXLog
    NhlTInteger
trYLog
    NhlTInteger

Transformation resources with prefix tr

trXMaxF
    NhlTFloat
trXMinF
    NhlTFloat
trXReverse
    NhlTInteger
trYMaxF
    NhlTFloat
trYMinF
    NhlTFloat
trYReverse
    NhlTInteger

TextItem resources with prefix tx

txAngleF
    NhlTFloat
txBackgroundFillColor
    NhlTColorIndex
txConstantSpacingF
    NhlTFloat
txDirection
    NhlTTextDirection
txFont
    NhlTFont
txFontAspectF
    NhlTFloat
txFontColor
    NhlTColorIndex
txFontHeightF
    NhlTFloat
txFontQuality
    NhlTFontQuality
txFontThicknessF
    NhlTFloat
txFuncCode

NhlTCharacter
txJust
NhlTJustification
txPerimColor
NhlTColorIndex
txPerimDashLengthF
NhlTFloat
txPerimDashPattern
NhlTDashIndex
txPerimOn
NhlTBoolean
txPerimSpaceF
NhlTFloat
txPerimThicknessF
NhlTFloat
txPosXF
NhlTFloat
txPosYF
NhlTFloat
txString
NhlTString

VectorPlot resources with prefix vc

vcExplicitLabelBarLabelsOn
NhlTBoolean
vcFillArrowEdgeColor
NhlTColorIndex
vcFillArrowEdgeThicknessF
NhlTFloat
vcFillArrowFillColor
NhlTColorIndex
vcFillArrowHeadInteriorXF
NhlTFloat
vcFillArrowHeadMinFracXF
NhlTFloat
vcFillArrowHeadMinFracYF
NhlTFloat
vcFillArrowHeadXF
NhlTFloat
vcFillArrowHeadYF
NhlTFloat
vcFillArrowMinFracWidthF
NhlTFloat
vcFillArrowWidthF
NhlTFloat
vcFillArrowsOn
NhlTBoolean

vcFillOverEdge
NhlTBoolean
vcGlyphStyle
NhlTVectorGlyphStyle
vcLabelBarEndLabelsOn
NhlTBoolean
vcLabelFontColor
NhlTColorIndex
vcLabelFontHeightF
NhlTFloat
vcLabelsOn
NhlTBoolean
vcLabelsUseVectorColor
NhlTBoolean
vcLevelColors
NhlTColorIndexGenArray
vcLevelCount
NhlTInteger
vcLevelSelectionMode
NhlTLevelSelectionMode
vcLevelSpacingF
NhlTFloat
vcLevels
NhlTFloatGenArray
vcLineArrowColor
NhlTColorIndex
vcLineArrowHeadMaxSizeF
NhlTFloat
vcLineArrowHeadMinSizeF
NhlTFloat
vcLineArrowThicknessF
NhlTFloat
vcMagnitudeFormat
NhlTString
vcMagnitudeScaleFactorF
NhlTFloat
vcMagnitudeScaleValueF
NhlTFloat
vcMagnitudeScalingMode
NhlTScalingMode
vcMapDirection
NhlTBoolean
vcMaxLevelCount
NhlTInteger
vcMaxLevelValF
NhlTFloat
vcMaxMagnitudeF
NhlTFloat

vcMinAnnoAngleF
NhlTFloat
vcMinAnnoArrowAngleF
NhlTFloat
vcMinAnnoArrowEdgeColor
NhlTColorIndex
vcMinAnnoArrowFillColor
NhlTColorIndex
vcMinAnnoArrowLineColor
NhlTColorIndex
vcMinAnnoArrowMinOffsetF
NhlTFloat
vcMinAnnoArrowSpaceF
NhlTFloat
vcMinAnnoArrowUseVecColor
NhlTBoolean
vcMinAnnoBackgroundColor
NhlTColorIndex
vcMinAnnoConstantSpacingF
NhlTFloat
vcMinAnnoExplicitMagnitudeF
NhlTFloat
vcMinAnnoFont
NhlTFont
vcMinAnnoFontAspectF
NhlTFloat
vcMinAnnoFontColor
NhlTColorIndex
vcMinAnnoFontHeightF
NhlTFloat
vcMinAnnoFontQuality
NhlTFontQuality
vcMinAnnoFontThicknessF
NhlTFloat
vcMinAnnoFormat
NhlTString
vcMinAnnoFuncCode
NhlTCharacter
vcMinAnnoJust
NhlTJustification
vcMinAnnoOn
NhlTBoolean
vcMinAnnoOrientation
NhlTOrientation
vcMinAnnoOrthogonalPosF
NhlTFloat
vcMinAnnoParallelPosF
NhlTFloat

vcMinAnnoPerimColor
NhlTColorIndex
vcMinAnnoPerimOn
NhlTBoolean
vcMinAnnoPerimSpaceF
NhlTFloat
vcMinAnnoPerimThicknessF
NhlTFloat
vcMinAnnoSide
NhlTPosition
vcMinAnnoString1
NhlTString
vcMinAnnoString1On
NhlTBoolean
vcMinAnnoString2
NhlTString
vcMinAnnoString2On
NhlTBoolean
vcMinAnnoTextDirection
NhlTTextDirection
vcMinAnnoZone
NhlTInteger
vcMinDistanceF
NhlTFloat
vcMinFracLengthF
NhlTFloat
vcMinLevelValF
NhlTFloat
vcMinMagnitudeF
NhlTFloat
vcMonoFillArrowEdgeColor
NhlTBoolean
vcMonoFillArrowFillColor
NhlTBoolean
vcMonoLineArrowColor
NhlTBoolean
vcMonoWindBarbColor
NhlTBoolean
vcNoDataLabelOn
NhlTBoolean
vcNoDataLabelString
NhlTString
vcPositionMode
NhlTVectorPositionMode
vcRefAnnoAngleF
NhlTFloat
vcRefAnnoArrowAngleF
NhlTFloat

vcRefAnnoArrowEdgeColor
NhlTColorIndex
vcRefAnnoArrowFillColor
NhlTColorIndex
vcRefAnnoArrowLineColor
NhlTColorIndex
vcRefAnnoArrowMinOffsetF
NhlTFloat
vcRefAnnoArrowSpaceF
NhlTFloat
vcRefAnnoArrowUseVecColor
NhlTBoolean
vcRefAnnoBackgroundColor
NhlTColorIndex
vcRefAnnoConstantSpacingF
NhlTFloat
vcRefAnnoExplicitMagnitudeF
NhlTFloat
vcRefAnnoFont
NhlTFont
vcRefAnnoFontAspectF
NhlTFloat
vcRefAnnoFontColor
NhlTColorIndex
vcRefAnnoFontHeightF
NhlTFloat
vcRefAnnoFontQuality
NhlTFontQuality
vcRefAnnoFontThicknessF
NhlTFloat
vcRefAnnoFormat
NhlTString
vcRefAnnoFuncCode
NhlTCharacter
vcRefAnnoJust
NhlTJustification
vcRefAnnoOn
NhlTBoolean
vcRefAnnoOrientation
NhlTOrientation
vcRefAnnoOrthogonalPosF
NhlTFloat
vcRefAnnoParallelPosF
NhlTFloat
vcRefAnnoPerimColor
NhlTColorIndex
vcRefAnnoPerimOn
NhlTBoolean

vcRefAnnoPerimSpaceF
NhlTFloat
vcRefAnnoPerimThicknessF
NhlTFloat
vcRefAnnoSide
NhlTPosition
vcRefAnnoString1
NhlTString
vcRefAnnoString1On
NhlTBoolean
vcRefAnnoString2
NhlTString
vcRefAnnoString2On
NhlTBoolean
vcRefAnnoTextDirection
NhlTTextDirection
vcRefAnnoZone
NhlTInteger
vcRefLengthF
NhlTFloat
vcRefMagnitudeF
NhlTFloat
vcScalarFieldData
NhlTInteger
vcScalarMissingValColor
NhlTColorIndex
vcScalarValueFormat
NhlTString
vcScalarValueScaleFactorF
NhlTFloat
vcScalarValueScaleValueF
NhlTFloat
vcScalarValueScalingMode
NhlTScalingMode
vcUseRefAnnoRes
NhlTBoolean
vcUseScalarArray
NhlTBoolean
vcVectorDrawOrder
NhlTDrawOrder
vcVectorFieldData
NhlTInteger
vcWindBarbCalmCircleSizeF
NhlTFloat
vcWindBarbColor
NhlTColorIndex
vcWindBarbLineThicknessF
NhlTFloat

vcWindBarbScaleFactorF
NhlTFloat
vcWindBarbTickAngleF
NhlTFloat
vcWindBarbTickLengthF
NhlTFloat
vcWindBarbTickSpacingF
NhlTFloat
vcZeroFLabelAngleF
NhlTFloat
vcZeroFLabelBackgroundColor
NhlTColorIndex
vcZeroFLabelConstantSpacingF
NhlTFloat
vcZeroFLabelFont
NhlTFont
vcZeroFLabelFontAspectF
NhlTFloat
vcZeroFLabelFontColor
NhlTColorIndex
vcZeroFLabelFontHeightF
NhlTFloat
vcZeroFLabelFontQuality
NhlTFontQuality
vcZeroFLabelFontThicknessF
NhlTFloat
vcZeroFLabelFormat
NhlTString
vcZeroFLabelFuncCode
NhlTCharacter
vcZeroFLabelJust
NhlTJustification
vcZeroFLabelOn
NhlTBoolean
vcZeroFLabelOrthogonalPosF
NhlTFloat
vcZeroFLabelParallelPosF
NhlTFloat
vcZeroFLabelPerimColor
NhlTColorIndex
vcZeroFLabelPerimOn
NhlTBoolean
vcZeroFLabelPerimSpaceF
NhlTFloat
vcZeroFLabelPerimThicknessF
NhlTFloat
vcZeroFLabelSide
NhlTPosition

vcZeroFLabelString
NhlTString
vcZeroFLabelTextDirection
NhlTTextDirection
vcZeroFLabelZone
NhlTInteger

VectorField resources with prefix vf

vfCopyData
NhlTBoolean
vfDataArray
NhlTGenArray
vfExchangeDimensions
NhlTBoolean
vfExchangeUVData
NhlTBoolean
vfMagMaxV
NhlTVariable
vfMagMinV
NhlTVariable
vfMissingVValueV
NhlTVariable
vfPolarData
NhlTBoolean
vfSingleMissingValue
NhlTBoolean
vfUDataArray
NhlTGenArray
vfUMaxV
NhlTVariable
vfUMinV
NhlTVariable
vfVDataArray
NhlTGenArray
vfVMaxV
NhlTVariable
vfVMinV
NhlTVariable
vfXArray
NhlTGenArray
vfXCActualEndF
NhlTFloat
vfXCActualStartF
NhlTFloat
vfXCEndIndex
NhlTInteger
vfXCEndSubsetV

NhlTVariable
vfXCEndV
NhlTVariable
vfXCStartIndex
NhlTInteger
vfXCStartSubsetV
NhlTVariable
vfXCStartV
NhlTVariable
vfXCStride
NhlTInteger
vfYArray
NhlTGenArray
vfYCActualEndF
NhlTFloat
vfYCActualStartF
NhlTFloat
vfYCEndIndex
NhlTInteger
vfYCEndSubsetV
NhlTVariable
vfYCEndV
NhlTVariable
vfYCStartIndex
NhlTInteger
vfYCStartSubsetV
NhlTVariable
vfYCStartV
NhlTVariable
vfYCStride
NhlTInteger

View resources with prefix vp

vpAnnoManagerId
NhlTObjId
vpHeightF
NhlTFloat
vpKeepAspect
NhlTBoolean
vpOn
NhlTBoolean
vpUseSegments
NhlTBoolean
vpWidthF
NhlTFloat
vpXF
NhlTFloat

vpYF
    NhlTFloat

NcgmWorkstation resources with prefix wk

wkMetaName
    NhlTString

PSWorkstation resources with prefix wk

wkDeviceLowerX
    NhlTInteger
wkDeviceLowerY
    NhlTInteger
wkDeviceUpperX
    NhlTInteger
wkDeviceUpperY
    NhlTInteger
wkFullBackground
    NhlTBoolean
wkOrientation
    NhlTWorkOrientation
wkPSFileName
    NhlTString
wkPSFormat
    NhlTPSFormat
wkPSResolution
    NhlTInteger
wkVisualType
    NhlTVisualType

Workstation resources with prefix wk

wkBackgroundColor
    NhlTFloatGenArray
wkColorMap
    NhlTFloatGenArray
wkColorMapLen
    NhlTInteger
wkDashTableLength
    NhlTInteger
wkDefGraphicStyleId
    NhlTInteger
wkFillTableLength
    NhlTInteger
wkForegroundColor
    NhlTFloatGenArray
wkGksWorkId

NhlTInteger
wkMarkerTableLength
NhlTInteger

XWorkstation resources with prefix wk

wkPause
NhlTBoolean
wkWindowId
NhlTInteger
wkXColorMode
NhlTXColorMode

Workspace resources with prefix ws

wsCurrentSize
NhlTLong
wsMaximumSize
NhlTLong
wsThresholdSize
NhlTLong

XyPlot resources with prefix xy

xyComputeXMax
NhlTBoolean
xyComputeXMin
NhlTBoolean
xyComputeYMax
NhlTBoolean
xyComputeYMin
NhlTBoolean
xyCoordData
NhlTObjIdGenArray
xyCoordDataSpec
NhlTObjIdGenArray
xyDashPattern
NhlTDashIndex
xyDashPatterns
NhlTDashIndexGenArray
xyExplicitLabels
NhlTStringGenArray
xyExplicitLegendLabels
NhlTStringGenArray
xyLabelMode
NhlTLineLabelMode
xyLineColor
NhlTColorIndex

xyLineColors
    NhlTColorIndexGenArray
xyLineDashSegLenF
    NhlTFloat
xyLineLabelFontColor
    NhlTColorIndex
xyLineLabelFontColors
    NhlTColorIndexGenArray
xyLineLabelFontHeightF
    NhlTFloat
xyLineThicknessF
    NhlTFloat
xyLineThicknesses
    NhlTFloatGenArray
xyMarkLineMode
    NhlTMarkLineMode
xyMarkLineModes
    NhlTMarkLineModeGenArray
xyMarker
    NhlTMarkerIndex
xyMarkerColor
    NhlTColorIndex
xyMarkerColors
    NhlTColorIndexGenArray
xyMarkerSizeF
    NhlTFloat
xyMarkerSizes
    NhlTFloatGenArray
xyMarkerThicknessF
    NhlTFloat
xyMarkerThicknesses
    NhlTFloatGenArray
xyMarkers
    NhlTMarkerIndexGenArray
xyMonoDashPattern
    NhlTBoolean
xyMonoLineColor
    NhlTBoolean
xyMonoLineLabelFontColor
    NhlTBoolean
xyMonoLineThickness
    NhlTBoolean
xyMonoMarkLineMode
    NhlTBoolean
xyMonoMarker
    NhlTBoolean
xyMonoMarkerColor
    NhlTBoolean

xyMonoMarkerSize
    NhlTBoolean
xyMonoMarkerThickness
    NhlTBoolean
xyXIrrTensionF
    NhlTFloat
xyXIrregularPoints
    NhlTFloatGenArray
xyXStyle
    NhlTTickMarkStyle
xyYIrrTensionF
    NhlTFloat
xyYIrregularPoints
    NhlTFloatGenArray
xyYStyle
    NhlTTickMarkStyle

AnnoManager class resource descriptions

amOn

This boolean resource specifies whether the base plot should draw the annotation **View** object along with the other objects comprising the plot.

Default: True

amViewId

You can use this resource to retrieve the id of the **View** object controlled by the **AnnoManager**. Note that this resource can be set only when the **AnnoManager** object is created.

Default: <dynamic>

amResizeNotify

If this boolean resource is True, the **PlotManager** sets the height and width of the annotation item whenever the size of the base plot's viewport changes. Height is adjusted based on the ratio of the new viewport height to the old height. Width is adjusted based on the ratio of the new viewport width to the old width.

Default: False

amTrackData

When *amTrackData* is set False, the **PlotManager** locates the annotation **View** object in NDC space using the PlotManager Location Control Model resources *amZone*, *amSide*, *amParallelPosF*, *amOthogonalPosF* and *amJust*. If *amTrackData* is True, the annotation is positioned relative to the data coordinate space of the base plot. In this case, the **PlotManager** determines the base position of the annotation item from the values of the resources *amDataXF* and *amDataYF*. The resources *amZone*, *amSide*, *amParallelPosF*, and *amOthogonalPosF* are ignored. However, the resource *amJust* still determines the justification point of the annotation with respect to its base location.

Default: False

amZone

If *amTrackData* is False, *amZone* specifies the **PlotManager** zone assigned to the annotation. *amZone* is one of the resources you must set in order to control the location of the annotation in a manner consistent with the rules of the PlotManager Location Control Model. If *amZone* is set to 0, the annotation is located relative to the center of the **PlotManager** viewport; otherwise, it is located relative to one side of the zone's interior boundary, which is the bounding box of the previous zone.

Default: 0

amSide

If *amTrackData* is False, this resource of type NhlTPosition determines which of the zone's interior sides to use as a base for establishing the coordinate system within which the annotation is placed. The PlotManager Location Control Model requires this resource to allow control of the annotation in a manner consistent with other annotations. Its value determines the origin of the coordinate system and the direction of the positional resources, *amParallelPosF* and *amOrthogonalPosF*. It also constrains the value of *amJust* appropriately. There are four settings, which behave as follows, unless *amZone* is set to one of the special zones (0 or 1):

NhlTOP

The **PlotManager** locates the annotation relative to a line paralleling the top viewport boundary. *amOrthogonalPosF* increases in the direction of increasing NDC Y-Axis values and is constrained to positive values. *amParallelPosF* increases in the direction of increasing NDC X-Axis values. *amJust* is constrained to NhlBOTTOMRIGHT, NhlBOTTOMCENTER, or NhlBOTTOMLEFT.

NhlBOTTOM

The **PlotManager** locates the annotation relative to a line paralleling the bottom viewport boundary. *amOrthogonalPosF* increases in the direction of decreasing NDC Y-Axis values and is constrained to positive values. *amParallelPosF* increases in the direction of increasing NDC X-Axis values. *amJust* is constrained to NhlTOPRIGHT, NhlTOPCENTER, or NhlTOPLEFT.

NhlRIGHT

The **PlotManager** locates the annotation relative to a line paralleling the right viewport boundary. *amOrthogonalPosF* increases in the direction of increasing NDC X-Axis values and is constrained to positive values. *amParallelPosF* increases in the direction of increasing NDC Y-Axis values. *amJust* is constrained to NhlTOPLEFT, NhlCENTERLEFT, or NhlBOTTOMLEFT.

NhlLEFT

The **PlotManager** locates the annotation item relative to a line paralleling the left viewport boundary. *amOrthogonalPosF* increases in the direction of decreasing NDC X-Axis values and is constrained to positive values. *amParallelPosF* increases in the direction of increasing NDC Y-Axis values. *amJust* is constrained to NhlTOPRIGHT, NhlCENTERRIGHT, or NhlBOTTOMRIGHT.

If **amZone** is set to 0, the **PlotManager** locates the annotation item relative to the viewport center. If **amZone** is 1, the annotation is located relative to the viewport boundary itself, and the direction of the *amOrthogonalPosF* is opposite to the specification given above. Also if the **amZone** is either 0 or 1, *amJust* is not constrained, and *amOrthogonalPosF* may take on negative values.

Default: `Bottom`

amParallelPosF

If *amTrackData* is False, *amParallelPosF* specifies the coordinate of the base location of the annotation parallel to the current *amSide* and directed toward increasing NDC values. The PlotManager Location Control Model requires this resource to allow control of the annotation in a manner consistent with other annotations.

Default: 0.0

amOrthogonalPosF

If *amTrackData* is False and *amZone* is not equal to one, *amOrthogonalPosF* sets the coordinate of the base location of the annotation orthogonal to the current *amSide* and directed away from the center of the **PlotManager** viewport. If *amZone is* equal to one, the direction is from the viewport side specified by *amSide toward* the viewport center. The PlotManager Location Control Model requires this resource to allow control of the annotation in a manner consistent with other annotations.

Default: 0.0

amJust

This resource of type NhlTJustification, after constraint to an appropriate value based on *amSide*, sets the justification point of the annotation with respect to its base location. The PlotManager Location Control Model requires this resource to allow control of the annotation in a manner consistent with other annotations. Note that *amJust* still has an effect when *amTrackData* is True. Normally, *amJust* is constrained based on the value of *amSide*. However, if *amZone* is 0 or 1, or *amTrackData* is True, the **PlotManager** does not constrain *amJust*.

Note in addition that the position of the justification point is determined from the viewport of the annotation **View** object only. That is to say, while it is possible that **View** object's bounding box might be larger than its viewport, any extent outside the viewport is not considered when determining the justification point. This means that that there is the possibility of overlap if a **View** object has elements extending outside its own viewport along the edge closest to the **PlotManager** Plot viewport.

Default: `CenterCenter`

amDataXF

When *amTrackData* is True, *amDataXF* sets the X coordinate of the annotation item's base location.

Default: 0.0

amDataYF

When *amTrackData* is True, *amDataYF* sets the Y coordinate of the annotation item's base location.

Default: 0.0

App class resource descriptions

All of the following resources are only used if the Application programmer directly creates an **App** object. If they simply use the **NhlOpen** function, then these resources are not necessary.

One thing that is a little tricky about these resources is that they can be specified in the $(NCARG_USRRESFILE) and the $(NCARG_SYSRESFILE), but they will not affect anything if they are placed in the application specific resource file. This is simply because these resources are used to find those files, so they can't be specified in them.

appUsrDir

This resource is used to indicate what directory the **App** object should look in for the User's application specific resource file.

Default: *./ (Current Working Directory)*

appSysDir

This resource is used to indicate what directory the **App** object should look in for the System application specific resource file.

Default: <dynamic>
This resource defaults to the value of the $(NCARG_SYSAPPRES) environment variable.

appFileSuffix

This resource is used to specify a suffix for the Application specific resource files. It is useful to set this resource in the $(NCARG_USRRESFILE) and $(NCARG_SYSRESFILE), if an application programmer wants to provide more than one Application Specific Resource file. For example, if some of the end users do not have color monitors, then there could be two resource files. One *appname*.bw.res, and another *appname*.color.res. Then the *appFileSuffix* resource can be specified to indicate which Application specific resource file to use.

Default: .res

appDefaultParent

This resource is only *True* for one **App** object at a time. If it is *True* in a given **App** object, then that **App** object is the one that will be used as the parent of any objects created with the *parentid* argument of **NhlCreate** set to `NhlDEFAULT_APP` or `0`.

Default: <dynamic>
This resource defaults to *True* for the First App object created, otherwise it is *False*.

appResources

This resource specifies an array of names. Each one of these names dynamically becomes a valid resource for the **App** object. This is a very useful feature for developers of multi-user applications because configuration parameters for the application can become resources to the **App** object. Then, users of the application can configure the application using the same resource files that they use to configure the HLU portion of the application.

Default: NULL

CoordArrays class resource descriptions

caCopyArrays

This resource is used to determine if the **CoordArrays** object should make its own copy of the *caXArray* and *caYArray* data. If this resource is False, it is important for the programmer to keep the data around without modifying it. However; it is more efficient--if more dangerous--that way.

Default: True

caXArray

This resource specifies the **X** values of the X/Y coordinate data. It is an array of **X** values. If it is a one-dimensional array, then it specifies a single vector of **X** values. If it is a two-dimensional array, then the *caXCast* resource is used to determine which dimension indicates the number of vectors, and which dimension indicates the number of elements within each vector. If the *caXArray* resource is not set, then all of the **Y** values set with the *caYArray* resource will be paired with their integer array index. For example, assuming *caYCast* is `MultipleVectors` and **C** syntax:

```
(1, yarray[0][0])
(2, yarray[0][1])
(N, yarray[0][N-1])
```

If the *caXArray* resource is not set, then the *caYArray* resource must be set.

Default: NULL

caXCast

This resource is used to tell the **CoordArrays** object how to interpret the *caXArray* resource. The three valid values are:

`SingleVector`

The value `SingleVector` indicates that the *caXArray* provides the data for a single vector. This vector will be reused to match every vector specified by the *caYArray*. If the *caXArray* resource is set with a one-dimensional array, then the entire array will be used for the vector. If the *caXArray* resource is set with a two-dimensional array, then the vector will be made up from the values in the fastest-changing dimension. For example, in pseudo **C** array notation:

```
float   xarray[0][0-(N-1)];
```

In pseudo **Fortran** array notation:

```
REAL    XARR(1-N,1)
```

Where `N` is the length of the given dimension.

`MultipleVectors`

The value `MultipleVectors` indicates that if the *caXArray* resource is set with a one-dimensional array, then the entire array is used to specify a single vector that is not reused. If the *caXArray* resource is set with a two-dimensional array, then the *caXArray* has the array ordered such that the fastest-changing dimension contains the elements of each vector, and the other dimension contains the vectors. For example, in pseudo **C** array notation:

```
float    xarray[NUM_VECTORS][NUM_ELEMENTS];
```

In pseudo **Fortran** array notation:

```
REAL     XARR(NUM_ELEMENTS,NUM_VECTORS)
```

`SplitVectors`

The value `SplitVectors` indicates that the *caXArray* has the array ordered such that the fastest-changing dimension contains the vectors, and the other dimension contains the elements. For example, in pseudo **C** array notation:

```
float    xarray[NUM_ELEMENTS][NUM_VECTORS];
```

In pseudo **Fortran** array notation:

```
REAL     XARR(NUM_VECTORS,NUM_ELEMENTS)
```

Default: <dynamic>
If the *caXArray* resource is specified with a one-dimensional array or not at all, then the default value is `SingleVector`. Otherwise, the default value is `MultipleVectors`.
NOTE: If the *caXCast* resource is set to `SplitVectors`, then the *caXArray* resource will need to be reordered and copied internally so it is not as efficient.

caXMissingV

This resource indicates a **missing** value for the elements in the *caXArray* resource. When the HLU library is parsing the data in the *caXArray* elements, it will treat any element with this value as **missing** data. This resource has a *dynamic* type so that elements of any type can be set to it.

Default: None

caXMaxV

This resource is used to tell the **CoordArrays** object the maximum value contained in the *caXArray* resource. If it is not specified, the **CoordArrays** object will compute it. This resource has a *dynamic* type so that elements of any type can be set to it.

Default: <*dynamic*>
The value will be computed if it is not set.

caXMinV

This resource is used to tell the **CoordArrays** object the minimum value contained in the *caXArray* resource. If it is not specified, the **CoordArrays** object will compute it. This resource has a *dynamic* type so that elements of any type can be set to it.

Default: <*dynamic*>
The value will be computed if it is not set.

caYArray

This resource specifies the **Y** values of the X/Y coordinate data. It is an array of **Y** values. If it is a one-dimensional array, then it specifies a single vector of **Y** values. If it is a two-dimensional array, then the *caYCast* resource is used to determine which dimension indicates the number of vectors, and which dimension indicates the number of elements within each vector. If the *caYArray* resource is not set, then all of the **X** values set with the *caXArray* resource will be paired with their integer array index. For example, assuming *caYCast* is `MultipleVectors` and **C** array syntax:

```
(xarray[0][0],1)
(xarray[0][1],2)
(xarray[0][N-1],N)
```

If the *caYArray* resource is not set, then the *caXArray* resource must be set.

Default: NULL

caYCast

This resource is used to tell the **CoordArrays** object how to interpret the *caYArray* resource. The three valid values are:

`SingleVector`

The value `SingleVector` indicates that the *caYArray* provides the data for a single vector. This vector will be reused to match every vector specified by the *caXArray*. If the *caYArray* resource is set with a one dimensional array, then the entire array will be used for the vector. If the *caYArray* resource is set with a two dimensional array, then the vector will be made up from the values in the fastest changing dimension. For example, in pseudo **C** array notation:

```
float   yarray[0][0-(N-1)];
```

In pseudo **Fortran** array notation:

```
REAL    YARR(1-N,1)
```

Where `N` is the length of the given dimension.

`MultipleVectors`

The value `MultipleVectors` indicates that if the *caYArray* resource is set with a one-dimensional array, then the entire array is used to specify a single vector that is not reused. If the *caYArray* resource is set with a two-dimensional array, then the *caYArray* has the array ordered such that the fastest-changing dimension contains the elements of each vector, and the other dimension contains the vectors. For example, in pseudo **C** array notation:

```
float   yarray[NUM_VECTORS][NUM_ELEMENTS];
```

In pseudo **Fortran** array notation:

```
REAL    YARR(NUM_ELEMENTS,NUM_VECTORS)
```

`SplitVectors`

The value `SplitVectors` indicates that the *caYArray* has the array ordered such that the fastest-changing dimension contains the vectors, and the other dimension contains the elements. For example, in pseudo **C** array notation:

```
float   yarray[NUM_ELEMENTS][NUM_VECTORS];
```

In pseudo **Fortran** array notation:

```
REAL    YARR(NUM_VECTORS,NUM_ELEMENTS)
```

Default: <dynamic>
If the *caYArray* resource is specified with a one-dimensional array or not at all, then the default value is `SingleVector`. Otherwise, the default value is `MultipleVectors`.
Note: If the *caYCast* resource is set to `SplitVectors`, then the *caYArray* resource will need to be reordered and copied internally so it is not as efficient.

caYMissingV

This resource indicates a **missing** value for elements in the *caYArray* resource. When the HLU library is parsing the data in the *caYArray* elements, it will treat any element with this value as **missing** data. This resource has a *dynamic* type so that elements of any type can be set to it.

Default: None

caYMaxV

This resource is used to tell the **CoordArrays** object the maximum value contained in the *caYArray*resource. If it is not specified, the **CoordArrays** object will compute it. This resource has a *dynamic* type so that elements of any type can be set to it.

Default: <*dynamic*>
The value will be computed if it is not set.

caYMinV

This resource is used to tell the **CoordArrays** object the minimum value contained in the *caYArray* resource. If it is not specified, the **CoordArrays** object will compute it. This resource has a *dynamic* type so that elements of any type can be set to it.

Default: <*dynamic*>
The value will be computed if it is not set.

ContourPlot class resource descriptions

cnScalarFieldData

Specifies the id of a **ScalarField** data object. There is no default for this resource; it is the only resource that must be set for the **ContourPlot** object to draw a plot. You *may* create a **ContourPlot** object without setting the *cnScalarFieldData*, and auxiliary annotations such as tick marks and titles may appear as the result of a draw, but the **ContourPlot** itself will not show up. The **ScalarField** object can provide either regularly spaced or irregular rectangular gridded data to the **ContourPlot** object, and it provides a number of resources for controlling the ingestion of the raw data.

Default: <None>

cnLevelSelectionMode

This enumerated resource of type NhlTLevelSelectionMode provides four methods for selecting the contour intervals displayed in a plot:

`AutomaticLevels`

Ordinarily this mode determines contour levels by picking a spacing value from a set of relatively "round" numbers scaled by powers of 10 to the range of the data. This set of numbers is as follows: 1.0, 2.0, 2.5, 4.0, 5.0. The number of levels chosen will be as close as possible to the value of *cnMaxLevelCount* without exceeding it. Once the spacing is chosen, the minimum contour level is set to the value of the least multiple of the spacing greater than the minimum data value. Likewise the maximum contour level becomes the greatest multiple of the spacing less than the maximum data value. Based on these values, **ContourPlot** sets the resources *cnLevelSpacingF*, *cnMinLevelValF*, and *cnMaxLevelValF* appropriately.

On the other hand, if you explicitly set the resource *cnLevelSpacingF* to a valid value greater than 0.0 and less than the range of the data, it will be used as the interval spacing. The minimum and maximum levels are calculated as before. If as a consequence, *cnMaxLevelCount* is less than the number of levels so specified, it will be set to the number of levels actually needed. However, if the choice of spacing causes the absolute maximum number of levels, currently 255, to be exceeded, **ContourPlot** will issue a warning message and recalculate the spacing as previously described.

In any case, **ContourPlot** sets the elements of the array resource *cnLevels* to the values of the contour levels chosen and the read-only resource *cnLevelCount* to the number of levels.

`ManualLevels`

`ManualLevels` mode bases the choice of contour levels on the values of the resources *cnLevelSpacingF*, *cnMinLevelValF*, and *cnMaxLevelValF*. Starting at *cnMinLevelValF*, contour levels are created at intervals spaced by the value of *cnLevelSpacingF* until *cnMaxLevelValF* is reached. The final contour level will always be *cnMaxLevelValF*. **ContourPlot** sets elements of the array resource *cnLevels* to the values of each contour level chosen and the read-only resource *cnLevelCount* to the number of levels. If the current value of *cnMaxLevelCount* is less than *cnLevelCount*, it is reset to the value of *cnLevelCount*. However, if the level count would exceed the absolute maximum number of levels, currently 255, **ContourPlot** issues a warning and chooses a new value of *cnLevelSpacingF* based on the value of *cnMaxLevelCount*.

If you choose `ManualLevels` selection mode when the **ContourPlot** object is created, and if you do not set *cnMinLevelValF*, **ContourPlot** will choose levels as if you had set `AutomaticLevels` mode. If you set *cnMinLevelValF* only, a default spacing is used, and the value of *cnMaxLevelValF* is determined as it would be for `AutomaticLevels` mode.

`ExplicitLevels`

This mode allows you to specify the value of each contour level by explicitly setting the contents of the *cnLevels* array. If you choose `ExplicitLevels` selection mode when creating a **ContourPlot** object, but do not specify the contents of the *cnLevels* array, **ContourPlot** will choose levels as if you had specified `AutomaticLevels` mode. Thereafter, when you set `ExplicitLevels` mode, **ContourPlot** uses the current contents of *cnLevels*, whether or not you set it explicitly.

If the number of elements in *cnLevels* exceeds the absolute maximum number of levels (currently 255), **ContourPlot** issues a warning and the mode defaults to `AutomaticLevels`. Note that **ContourPlot** always sorts the elements of *cnLevels* into a monotonically increasing sequence. After sorting, *cnMinLevelValF* is set equal to the value of first element of *cnLevels*, and *cnMaxLevelValF* is set to the value of the last element. *cnLevelSpacingF* is set to the average value of the spacing separating each level.

`EqualSpacedLevels`

For this mode, **ContourPlot** divides the range spanning the dataset minimum and maximum values into *cnMaxLevelCount*+1 equally spaced intervals. *cnLevelSpacingF* is set to the value of this interval. *cnMinLevelValF* is set to the value of the dataset minimum plus the value of *cnLevelSpacingF*. *cnMaxLevelSpacingF* is set to the value of the dataset maximum minus the value of *cnLevelSpacingF*. This results in *cnMaxLevelCount* levels; therefore **ContourPlot** sets the read-only resource *cnLevelCount* equal to *cnMaxLevelCount*.

Default: `AutomaticLevels`

cnLevelCount

A read-only resource set to the actual number of levels chosen during the level selection process.

Default: <dynamic>

cnMaxLevelCount

When the *cnLevelSelectionMode* is `AutomaticLevels` and *cnLevelSpacingF* is not explicitly set, **ContourPlot** picks a number of levels less than or equal to the current value of *cnMaxLevelCount*. If the *cnLevelSelectionMode* is `EqualSpacedLevels`, **ContourPlot** picks exactly *cnMaxLevelCount* levels. If *cnMaxLevelCount* exceeds the absolute maximum level count allowed by **ContourPlot** (currently 255), a warning is issued and the value is reset to this maximum. If *cnLevelSelectionMode* is `ManualLevels` or `ExplicitLevels` or `AutomaticLevels` with *cnLevelSpacingF* explicitly set, **ContourPlot** sets *cnMaxLevelCount* to the number of levels actually used if this number is greater than the current value of *cnMaxLevelCount*.

Default: 16

cnLevelSpacingF

When the *cnLevelSelectionMode* is `ManualLevels` or when the *cnLevelSelectionMode* is `AutomaticLevels`, and *cnLevelSpacingF* is explicitly set, *cnLevelSpacingF* determines the spacing between contour intervals. Otherwise, the **ContourPlot** object sets the value of *cnLevelSpacingF* based on the contour levels actually chosen. When the *cnLevelSelectionMode* is `ExplicitLevels`, *cnLevelSpacingF* will be set to the arithmetic average of the spacing between levels.

Default: 5.0

cnMinLevelValF

When the *cnLevelSelectionMode* is `ManualLevels`, the value of *cnMinLevelValF*, if set, determines the lowest contour level. Otherwise, **ContourPlot** sets the *cnMinLevelValF* to a value equal to the lowest contour level picked.

Default: <dynamic>

cnMaxLevelValF

When the *cnLevelSelectionMode* is `ManualLevels`, the value of *cnMaxLevelValF*, if set, determines the highest contour level. Otherwise, **ContourPlot** sets the *cnMaxLevelValF* to a value equal to the highest contour level picked.

Default: <dynamic>

cnLevels

An array of floats containing the contour levels used to render the contour plot. If the *cnLevelSelectionMode* is `ExplicitLevels`, you may set these values yourself. Otherwise, the **ContourPlot** object sets the elements of this array.

Default: <dynamic>

cnMonoLevelFlag

When *cnMonoLevelFlag* is set True, the scalar resource *cnLevelFlag* sets the level flag controlling the appearance of the line and/or the line label to a single value. In this case, the *cnLineLabelInterval* resource has no effect. Otherwise, when *cnMonoLevelFlag* is set False, you can set the level flag for each line individually using the resource *cnLevelFlags*, or set line labels to appear at uniform contour intervals using the resource *cnLineLabelInterval*.

Default: False

cnLevelFlag

When *cnMonoLevelFlag* is set True, this resource of type NhlTcnLevelUseMode sets the rendering of all contour levels to one of four possible values:

`NoLine`
: No lines or labels appear.

`LineOnly`
: Lines but no labels appear.

`LabelOnly`
: Labels but no lines appear.

`LineAndLabel`
: Both lines and labels appear.

Note that the resources *cnLinesOn* and *cnLineLabelsOn* must both be set True to enable all possible settings of *cnLevelFlag*.

Default: `LineOnly`

cnLevelFlags

When *cnMonoLevelFlag* is False, *cnLevelFlags* is an array of type

NhlTcnLevelUseModeGenArray that allows you individual control over whether a line and/or a line label is to appear at each contour level. It has no effect on contour fill. There are four choices:

`NoLine`
: No lines or labels appear at the contour level.

`LineOnly`
: Lines but no labels appear at the contour level.

`LabelOnly`
: Labels but no lines appear at the contour level.

`LineAndLabel`
: Both lines and labels appear at the contour level.

Note that *cnLinesOn* and *cnLineLabelsOn* exert underlying control over the appearance of contour lines and line labels. If *cnLinesOn* is set False, no lines will appear regardless of the values of any *cnLevelFlags* elements; likewise, if *cnLineLabels* is False, no line labels will appear.

If *cnMonoLevelFlag* is False and you do not explicitly set *cnLevelFlags*, the **ContourPlot** object will set its elements for you based on the value of *cnLineLabelInterval*. Elements will be set to `LineOnly` except that if *cnLineLabelInterval* is greater than 0, elements separated by the interval of *cnLineLabelInterval* will be set to `LineAndLabel`. This effect will occur both at initialization and any time you explicitly set *cnLineLabelInterval*.

Default: <dynamic>

cnLineLabelInterval

When *cnMonoLevelFlag* is False and you do not explicitly set the array *cnLevelFlags*, a positive value of *cnLineLabelInterval* sets the number of levels from a labeled level to the next labeled level. The interval is generated in both directions from the contour level requiring the least number of significant digits to express (with the value 0.0 considered to require zero significant digits). If *cnLineLabelInterval* is less than or equal to zero, then no contour levels are labeled.

Default: 2

cnLinesOn

If this boolean resource is set False, no contour lines will appear in the contour plot, regardless of the values contained in the *cnLevelFlags* array resource. It has no effect on line labels.

Default: True

cnLineDrawOrder

This resource of type NhlTDrawOrder determines when the contour lines are drawn relative to other elements of the plot. There are three choices:

`PreDraw`
: Draw the contour lines before the standard draw; the lines will be overlaid by any subsequently drawn elements.

`Draw`
: Draw the lines during the standard draw; the lines will overlay any elements drawn during the predraw phase, but will underlie elements drawn during the postdraw phase.

`PostDraw`
: Draw the lines after the standard draw; the lines will overlay any elements drawn during the predraw and draw phases.

Default: `Draw`

cnMonoLineColor

When set True, all contour lines are set to a single color, as specified by the value of the scalar resource *cnLineColor*. Otherwise, the elements of the array resource *cnLineColors* control the color of each line individually.

Default: True

cnLineColor

When *cnMonoLineColor* is set True, this resource of type NhlTColorIndex sets a uniform color for all contour lines.

Default: `Foreground` (1)

cnLineColors

When *cnMonoLineColor* is False, each element of this array resource of type NhlTColorIndexGenArray individually sets the color of the contour line drawn at the corresponding contour level. If the array is not set explicitly, it will dynamically default to a set of colors spread numerically over the range of defined HLU color indexes. The colors used will therefore depend on the number of contour levels. If the array currently contains fewer elements than *cnLevelCount*, more elements will be added to the array and given values according to the default color assignment scheme; existing elements with valid color index values will not be modified.

Default: <dynamic>

cnMonoLineDashPattern

When set True, all contour lines are rendered using the same dash pattern, as specified by the value of the scalar resource *cnLineDashPattern*. Otherwise, the elements of the array resource *cnLineDashPatterns* control the dash pattern of each line individually.

Default: True

cnLineDashPattern

When *cnMonoLineDashPattern* is set True, this resource of type NhlTDashIndex sets a uniform dash pattern for all contour lines.

Default: `SolidLine` (0)

cnLineDashPatterns

When *cnMonoLineDashPattern* is False, this array resource of type NhlTDashIndexGenArray individually sets the dash pattern of a contour line drawn at the corresponding contour level. If the array is not set explicitly, the **ContourPlot** object will assign sequential dash pattern indexes to each array element--starting with dash pattern index 1, not `"SolidLine"` (0)--up to the maximum number of existing dash patterns. Thereafter, it will repeat the sequence using modular arithmetic. If the array currently contains fewer elements than *cnLevelCount*, more elements will be added to the array and given values according to the same assignment scheme; existing elements with valid dash pattern index values will not be modified.

Default: <dynamic>

cnMonoLineThickness

When set True, all contour lines are rendered using the same line thickness, as specified by the value of the scalar resource *cnLineThicknessF*. Otherwise, the elements of the array resource *cnLineThicknesses* control the thickness of each line individually.

Default: True

cnLineThicknessF

When *cnMonoLineThickness* is True, this resource sets a uniform line thickness for all contour lines.

Default: 1.0

cnLineThicknesses

If *cnMonoLineThickness* is False, each element of this array resource specifies the line thickness of a contour line drawn at the corresponding contour level. If the array is not set explicitly, all elements default to the value 1.0. If the array currently contains fewer elements than *cnLevelCount*, more elements will be added to the array and set to the default value, 1.0; existing elements with valid line thicknesses (greater than 0.0) will not be modified.

Default: 1.0 for all elements

cnLineDashSegLenF

This resource indicates the length of each segment of the dash patterns used to draw contour lines. It is the length in NDC units before the dash pattern repeats itself. If *cnLineLabelPlacementMode* is set to `Constant` the line labels are rendered as part of the dash pattern, and therefore *cnLineDashSegLenF* also controls the spacing between the line labels.

cnLineDashSegLenF automatically scales with changes in the size of the viewport of the **ContourPlot** object. **ContourPlot** sets its default value dynamically based on the ratio of the actual plot viewport width to the reference viewport width.

Default: 0.15 (for a viewport width of 0.6)

cnFillOn

This boolean resource controls whether the areas between contour levels are filled using a fill pattern (possibly solid) and a fill color.

Default: False

cnRasterModeOn

If this boolean resource is set True, **ContourPlot** generates a representation of the data by individually assigning colors to the elements of a two-dimensional array of rectangular cells superimposed on an area bounding the data grid. This contrasts with the area fill method, in which a plot is generated by coloring the irregular areas formed by the boundaries of the lines defining adjacent contour levels. When raster mode is in effect, only solid fill is possible, and therefore the following resources have no effect:

- *cnFillBackgroundColor*
- *cnMonoFillColor*
- *cnFillColor*
- *cnMonoFillPattern*
- *cnFillPattern*
- *cnFillPatterns*
- *cnMonoFillScale*
- *cnFillScaleF*
- *cnFillScales*
- *cnMissingValFillPattern*

- *cnMissingValFillScaleF*

The color chosen for each cell is based on the color indexes contained in the *cnFillColors* array resource unless the cell center is in a missing-value area or outside the grid. If the cell center is in a missing-value area, it is assigned the color index given by the value of the resource *cnMissingValFillColor*. If it is outside the grid, it is set to the value `Background` (0). Also note that since only solid fill is possible, the fill color `Transparent` (-1) is not supported; if an element of *cnFillColors* or *cnMissingValFillColor* is set to `Transparent`, **ContourPlot** substitutes the value `Background` in the cell array.

In general, within the limits of the **Workstation** output device resolution, the quality of a raster plot improves as the density of the cell array is increased. However, processing time and memory requirements also increase.

Default: False

cnRasterSmoothingOn

If *cnRasterSmoothingOn* is set True, the level (and hence the color) assigned to each cell is determined by interpolating the values of neighboring points in the data grid. If *cnRasterSmoothingOn* is False, **ContourPlot** creates a discrete raster plot: any raster cell whose center lies within the rectangular area bounded by lines halfway between each grid point (in data space) is given the color assigned to the level representing the grid point datum.

The effect of this resource on the appearance of the plot depends on the density of the data relative to the number of cells in the raster array as limited by the resolution of the **Workstation** output device. If the data grid is more densely spaced than the cell array, two plots generated with opposite settings of this resource will appear qualitatively similar, although there may be differences of detail. However, if the data grid is sparse relative to the cell array density, turning smoothing off results in a plot that appears blocky and discontinuous compared to a plot drawn with smoothing turned on.

Default: False

cnRasterMinCellSizeF

This resource supplies a lower limit for the size of a raster cell whenever **ContourPlot** determines the cell array density dynamically. It applies in all situations except when you set the cell size explicitly using *cnRasterCellSizeF* or you set *cnRasterSampleFactorF* to 0.0. It prevents the cell array from becoming unreasonably large when drawing a raster plot on a high-resolution **workstation** such as the **PSWorkstation**. Set it to a smaller value if you actually require plots at higher resolution. However, note that there is an inverse quadratic relationship between the value of this resource and the size of the cell array, with consequent effects on storage requirements and processing time. You can use a larger value if you want quick plots where detail is not so important.

Default: 0.001

cnRasterSampleFactorF

If *cnRasterCellSizeF* is not explicitly set, **ContourPlot** adjusts the raster cell size dynamically in order to create a plot with as much resolution as is necessary and/or practical, while simultaneously trying to keep the cell array as small as possible. The default minimum cell size is the **workstation** device pixel size or the value of *cnRasterMinCellSizeF*, whichever is bigger. **ContourPlot** uses this minimum size as the default dynamic size whenever *cnRasterSmoothingOn* is True.

If *cnRasterSmoothingOn* is False, **ContourPlot** uses a larger default size if there is a linear mapping from data to NDC space and if the data grid has a lower resolution than a grid of minimum-sized cells would have. Linearity is considered separately along each axis: if the plot has one log axis and one linear axis, for example, rectangular cells may be created with the dimension along the log axis set to the minimum size while the linear dimension has a much larger size. If the plot is overlaid on a map projection, cells are set to the minimum size except for the special case of a non-rotated cylindrical equidistant projection with the center latitude set to 0.0.

Based on the dynamically determined default cell size and the NDC width and height of the data extent, **ContourPlot** calculates the number of cells needed along each dimension of the cell array. The *cnRasterSampleFactorF* resource acts as a multiplier of the number of raster cells along each dimension. Setting it to 2.0, for example, would double the number of cells along each axis, quadrupuling the overall size of the cell array. However, if this operation would result in the cells falling below the minimum size specified by *cnRasterMinCellSizeF*, their number is limited as required along the affected dimension. You can set *cnRasterSampleFactorF* to a value less than 1.0 in order improve plotting speed; the tradeoff will be diminished plot quality.

As a special case, if *cnRasterSampleFactorF* is set to 0.0, **ContourPlot** creates the cell array the same size as the data array, resulting in a one-to-one correspondence between data grid points and cells in the raster array. This is regardless of the **Workstation** device resolution or the setting of *cnRasterMinCellSizeF*.

Default: 1.0

cnRasterCellSizeF

If this resource is explicitly set to a value greater than 0.0, it specifies the height and width of each cell of the raster array in NDC units. The number of cells in the raster array is determined by how many cells of the specified size it takes to cover a rectangular area bounding the data grid. In this case, the number of elements in the data grid, the resolution of the **Workstation** output device, and the setting of *cnRasterSampleFactorF* have no effect on the size of the raster cell array.

No limit is placed on how small you may set this resource; it is not affected by the value of *cnRasterMinCellSizeF*. However, note that halving its size quadruples the overall number of cells in raster array, with consequent effects on processing time and memory usage. Once set, the value of this resource scales proportionally to changes in the viewport width, ensuring the same relative quality of output and processing time for any plot size.

Giving *cnRasterCellSizeF* a value less than or equal to 0.0 has the effect of 'unsetting' the resource, restoring the default method of sizing the raster cell array based on the device resolution, the size of the data array, and the values of the resources *cnRasterMinCellSizeF* and *cnRasterSampleFactorF*.

Even when unset, you may retrieve the value of this resource after a draw operation to find the minimum dimension of the cells in NDC units that was actually used to draw the plot.

Default: <dynamic>

cnFillDrawOrder

This resource of type NhlTDrawOrder determines when areas of the contour plot are filled relative to the drawing of other elements of the plot. There are three choices:

`PreDraw`
: Fill contour areas before the standard draw phase; fill areas will be overlaid by any subsequently drawn elements.

`Draw`
: Fill contour areas during the standard draw phase; the fill will overlay any elements drawn during the predraw phase, but will underlie elements drawn during the postdraw phase.

`PostDraw`
: Fill contour areas after the standard draw; the fill will overlay any elements drawn during the predraw and draw phases.

You must manipulate this resource in order to mask contours within the context of an Overlay of multiple plot objects.

Default: `Draw`

cnFillBackgroundColor

This resource of type NhlTColorIndex sets the background color used for **ContourPlot** fill patterns. It only has an effect for fill patterns that are neither `SolidFill` nor `HollowFill`. By default, this resource is set to `Transparent`, resulting in the underlying background appearing in the spaces between the elements of the fill pattern. If set to any other valid color index, the specified color will solidly fill all spaces between the fill pattern elements.

Default: `Transparent`

cnMonoFillColor

When set True, all contour fill areas are set to a single color, as specified by the value of the scalar resource *cnFillColor*. When False, the elements of the array resource *cnFillColors* control the color of each fill area individually.

Default: False

cnFillColor

When *cnMonoFillColor* is set True, this resource of type NhlTColorIndex sets a uniform fill color for all contour fill areas.

Default: `Foreground`

cnFillColors

If *cnMonoFillColor* is False, each element of this array of type NhlTColorIndexGenArray specifies the color of a contour fill area. Note that there is always one more fill area than there are contour levels. The first element of *cnFillColors* specifies the fill pattern for any region containing a data value less than the value of *cnMinLevelValF*, while the highest currently used element of *cnFillColors* specifies the fill pattern for regions containing data values greater than the value of *cnMaxLevelValF*. Otherwise *cnFillColors* works in a fashion similar to *cnLineColors*. If the array is not set explicitly, it will default to a set of color indexes sequentially incremented beginning with color index 1 (NhlFOREGROUND). If the array currently contains fewer elements than *cnLevelCount*+1, more elements will be added to the array and given values according to the default color assignment scheme; existing elements with valid color index values will not be modified.

Default: <dynamic>

cnMonoFillPattern

When set True, all contour fill areas use a single fill pattern, as specified by the value of the scalar resource *cnFillPattern*. When False, the elements of the array resource *cnFillPatterns* control the pattern used for each fill area individually.

Default: True

cnFillPattern

When *cnMonoFillPattern* is set True, this resource of type NhlTFillIndex sets a uniform pattern for all contour fill areas.

Default: `SolidFill`

cnFillPatterns

If *cnMonoFillPattern* is False, each element of this array of type NhlTFillIndexGenArray specifies the fill pattern used for a contour fill area. Note that there is always one more fill area than there are contour levels. The first element of *cnFillPatterns* specifies the fill pattern for any region containing a data value less than the value of *cnMinLevelValF*, while the highest currently used element of *cnFillPatterns* specifies the fill pattern for regions containing data values greater than the value of *cnMaxLevelValF*. If the array is not set explicitly, the **ContourPlot** object will assign sequential fill pattern indexes (starting with fill index 1--not "SolidFill", which is fill index 0) to each array element up to the maximum number of existing fill patterns. Thereafter it will repeat the sequence using modular arithmetic. If the array currently contains fewer elements than *cnLevelCount*+1, more elements will be added to the array and given values according to the same assignment scheme; existing elements with valid fill pattern index values will not be modified.

Default: <dynamic>

cnMonoFillScale

When set True, all contour fill patterns are scaled by a single factor, as specified by the value of the scalar resource *cnFillScale*. When False, the elements of the array resource *cnFillScales* control the scaling applied to the fill pattern assigned to each fill area individually.

Default: True

cnFillScaleF

When *cnMonoFillScale* is True, this resource sets a uniform scaling value for all contour fill patterns.

Default: 1.0

cnFillScales

When *cnMonoFillScale* is False, the *cnFillScales* array resource allows individual control of the scaling of the pattern used for each contour fill area. It has no effect for solid fill (fill pattern `"SolidFill"`, index 0). Values greater than 1.0 make the pattern spacing bigger than the default spacing, resulting in a fill that appears less dense. Values less than 1.0 have the opposite effect. Values less than or equal to 0.0 are invalid, generate a warning message, and are reset to the default value, 1.0.

Note that there is always one more fill area than there are contour levels. If the array is not set explicitly, the **ContourPlot** object will assign the value 1.0 to all elements of the array. If the array currently contains fewer elements than *cnLevelCount*+1, more elements will be added to the array and given the same default value of 1.0; existing elements with valid fill scale index values will not be modified.

Default: 1.0 for all elements

cnLabelDrawOrder

This resource of type NhlTDrawOrder determines when contour plot labels are drawn relative to the drawing of other elements of the plot. There are three choices:

`PreDraw`
: Draw contour labels before the standard draw phase; the labels will be overlaid by any subsequently drawn elements.

`Draw`
: Draw contour labels during the standard draw phase; the labels will overlay any elements drawn during the predraw phase as well as fill and contour lines drawn during the standard draw phase, but will underlie elements drawn during the postdraw phase.

`PostDraw`
: Draw contour labels after the standard draw; the labels will overlay any elements drawn during the predraw and draw phases as well as lines and fill drawn during the postdraw phase.

Default: `Draw`

cnLabelMasking

This resource controls whether **ContourPlot** lines and fill are masked in the areas where **ContourPlot** labels are to appear. When working with raster type devices, it is seldom necessary to mask for labels, since they can be made to solidly overlay the areas where they appear. Also there is a noticeable performance penalty when labels are masked, so the use of this option is discouraged unless it is necessary.

Default: False

cnLowUseHighLabelRes

If the boolean resource *cnLowUseHighLabelRes* is set True, all resources applying to low labels, except for *cnLowLabelString*, take their value from the corresponding high label resources.

Default: False

cnHighUseLineLabelRes

If the boolean resource *cnHighUseLineLabelRes* is set True, all resources applying to high labels, except for *cnHighLabelString* and *cnHighLabelAngleF*, take their value from the corresponding line label resources.

Default: False

cnConstFUseInfoLabelRes

If the boolean resource *cnConstFUseInfoLabelRes* is set True, all the resources applying to the constant field, except for *cnConstFLabelString*, take their value from the corresponding informational label resources.

Default: False

cnHighLowLabelOverlapMode

cnHighLowLabelOverlapMode is a resource of type NhlTcnHighLowLabelOverlapMode. It controls how high and low labels are allowed to overlap other elements of the **ContourPlot** plot and has six possible settings:

`IgnoreOverlap`
: The **ContourPlot** object does nothing to prevent high and low labels from overlapping other elements of the **ContourPlot** plot.

`OmitOverHL`
: The **ContourPlot** object omits high and low labels that would overlap previously determined high or low labels.

`OmitOverVP`
: The **ContourPlot** object omits high and low labels that would overlap the edge of the viewport.

`OmitOverVPAndHL`
: The **ContourPlot** object omits high and low labels that would overlap the edge of the viewport or previously determined high or low labels.

`AdjustVP`
: The **ContourPlot** object adjusts the position of high and low labels that would overlap the edge of the viewport enough to eliminate the problem.

`AdjustVPOmitOverHL`
: The **ContourPlot** object adjusts the position of high and low labels that would overlap the edge of the viewport enough to eliminate the problem and omits high and low labels that would overlap previously determined high or low labels.

V4.1 Status Note 5

Default: `IgnoreOverlap`

cnLabelScalingMode

This resource of type NhlTScalingMode determines how to scale the numbers representing scalar field data values in the **ContourPlot** object's labels. *cnLabelScalingMode* applies to numbers that represent data values in line labels, high and low labels, the informational label, and the constant field label. There are five choices:

`ScaleFactor`
: The **ContourPlot** object divides the data value by the value of *cnLabelScaleValueF* to obtain the number that appears in the label.

`ConfineToRange`
: The **ContourPlot** object uses *cnLabelScaleValueF* as an exclusive upper bound. A scale factor is selected such that the number used to represent the data value with the maximum absolute value will be less than *cnLabelScaleValueF*, but greater than or equal to *cnLabelScaleValueF* divided by 10.0. As an example, setting *cnLabelScaleValueF* to 100.0 would result in a scaled number in the range 10.0 to 99.999....

`TrimZeros`
: The **ContourPlot** object selects a scale factor such that the number representing the data value with the maximum absolute value will have the fewest possible extra zeros. If the maximum absolute value were 245000, for instance, it would scale to 245. If it were 0.000245, it would scale to 0.245. (The zero conventionally placed before the decimal point of a number less than 1.0 is not considered.)

`MaxSigDigitsLeft`
: The **ContourPlot** object selects a scale factor such that the number representing the data value with the maximum absolute value will have its rightmost significant digit directly to the left of the decimal point. The number of significant digits is determined by the format string resource, *cnMaxDataValueFormat*.

`AllIntegers`

The **ContourPlot** object selects a scale factor such that the numbers representing labeled contour levels (as determined by examination of the values of the *cnLabelFlags* array) can be rendered as integers.

In each case, the scale factor selected is placed into the read-only resource, *cnLabelScaleFactorF*. Note that the label value multiplied by the scale factor returns the actual data value represented.

Default: `ScaleFactor`

cnLabelScaleValueF

The interpretation of *cnLabelScaleValueF* depends on the value of the enumerated resource *cnLabelScalingMode*. When *cnLabelScalingMode* is set to `ScaleFactor`, data values are divided by the value of *cnLabelScaleValueF* to obtain the numbers appearing in **ContourPlot** object labels. If *cnLabelScalingMode* is set to `ConfineToRange`, the *cnLabelScaleValueF* represents an exclusive upper bound for the numbers used to represent the data values. For other values of *cnLabelScalingMode*, the *cnLabelScaleValueF* resource is ignored.

Default: 1.0

cnLabelScaleFactorF

cnLabelScaleFactorF is a read-only resource that contains the current scale factor that applies to **ContourPlot** object labels. Multiplying the numbers in the labels by the value of *cnLabelScaleFactorF* gives the actual data values the numbers represent. The **ContourPlot** object calculates its value based on the data value with the maximum absolute value, the current mode of the enumerated resource, *cnLabelScalingMode*, and perhaps the value of *cnLabelScaleValueF*. Its value may also be influenced by the number of significant digits specified by the format specification resource, *cnMaxDataValueFormat*. You may output the value of this resource in the informational label using the the substitution string "SFU".

Default: <dynamic>

cnMaxDataValueFormat

The *cnMaxDataValueFormat* resource is a string that specifies the printing format for the maximum data value according to the HLU Floating Point Format Specification scheme. This resource serves as a master control for formatting any number representing **ScalarField** data when used in a **ContourPlot** label. If the format string for a label type sets the dynamic attribute for any of its numerical format parameters, the parameter's value is determined by the value of the corresponding parameter in the *cnMaxDataValueFormat* specifier. Thus, you can control the numerical format parameters for all labels by adjusting a single resource. For some format parameters, notably the leftmost significant digit parameter, you must use the dynamic attribute in order to apply the parameter in a meaningful way.

Default: "*+.4^sg"

cnLineLabelsOn

If this boolean resource is set False, contour line labels will not appear in the contour plot. Note that this resource overrides the values contained in the *cnLevelFlags* array resource with respect to line labels, but has nothing to do with whether contour lines appear.

Default: True

cnExplicitLineLabelsOn

This boolean resource controls the way **ContourPlot** handles the *cnLineLabelStrings* array resource. When set True, **ContourPlot** will not modify explicitly set values of the line label strings. When *cnExplicitLineLabelsOn* is set False, **ContourPlot** ignores values of *cnLineLabelStrings* set by the user.

Default: False

cnLineLabelPlacementMode

This resource of type NhlTcnLineLabelPlacementMode determines the algorithm **ContourPlot** uses to place line labels. There are three choices:

`Constant`

ContourPlot draws line labels as an integral part of the line dash pattern. As a result, the labels are equally spaced along the lines. You can control the spacing between the labels by setting the *cnLineDashSegLenF* resource.

`Randomized`

ContourPlot places labels using a randomizing algorithm to vary the distance between labels. Currently there are no native resources for controlling the density of labels when using the `Randomized` mode. However, it is possible to directly control `Conpack` parameters that have this capability by setting the low level control resource *cnConpackParams*.

`Computed`

ContourPlot uses a more complex algorithm involving the local gradient, number of contour lines in a region, cumulative change in direction, and an optimum distance value to determine the best location for line labels. This method usually gives the best-looking results. Currently there are no native resources for controlling the density of labels when using the `Computed` mode. However, it is possible to directly control `Conpack` parameters that have this capability by setting the low level control resource *cnConpackParams*.

V4.1 Status Note 3

Default: `Randomized`

cnLineLabelStrings

Each element of the array resource *cnLineLabelStrings* specifies a string to be drawn at a contour level. Note that there is an element of this array for each contour level, whether or not (based on the contents of the *cnLevelFlags* array) a label will actually be drawn. If *cnExplicitLineLabelsOn* is set False, **ContourPlot** automatically sets the contents of the array to a set of strings representing the values of the contour levels, scaled based on the value of *cnLabelScaleFactorF*, and formatted based on the contents of *cnMaxDataValueFormat* and *cnLineLabelFormat*.

If *cnExplicitLineLabelsOn* is set True, **ContourPlot** will not modify any initialized element of the array, thus allowing you to set its values to strings of your choice. If you set the resource with an array containing fewer elements than the current number of levels, the remaining elements will be assigned using the automatic method.

Default: <dynamic>

cnMonoLineLabelFontColor

When set True, all contour line labels are rendered using a single color, as specified by the value of the scalar resource *cnLineLabelFontColor*. When False, the elements of the array resource *cnLineLabelFontColors* control the color of each line label individually.

Default: True

cnLineLabelFontColor

When *cnMonoLineLabelColor* is set True, this resource of type NhlTColorIndex sets a uniform color for all contour line labels.

Default: `Foreground`

cnLineLabelFontColors

When *cnMonoLineLabelFontColor* is False, each element of this array of type NhlTColorIndexGenArray specifies the color of a contour line label. If the array is not set explicitly, it will dynamically default to a set of colors spread numerically over the range of defined HLU color indexes. The colors used will therefore depend on the number of contour levels. If the array currently contains fewer elements than *cnLevelCount*, more elements will be added to the array and given values according to the default color assignment scheme; existing elements with valid color index values will not be modified.

Default: <dynamic>

cnLineLabelFormat

The *cnLineLabelFormat* resource is a string that specifies the printing format for the numeric portion of line labels according to the HLU Floating Point Format Specification scheme. It does not apply when the user explicitly sets the elements of *cnLineLabelStrings*. If any of the numerical format parameters have the dynamic attribute on, the corresponding parameter in the *cnMaxDataValueFormat* resource determines the parameter's value. The default value uses the shorthand notation '*+' to cause all numeric format parameters to be determined dynamically.

Default: "*+^sg"

cnLineLabelFontHeightF

This resource controls the height, in NDC units, of characters used in the text of line labels. The character width scales proportionally, unless you also modify the aspect ratio using the *cnLineLabelFontAspectF* resource. The line label text height scales with changes to the viewport width, unless you explicitly set *cnLineLabelFontHeightF* during the same call.

Default: <dynamic> -- 0.012 for a viewport width of 0.6

cnLineLabelFont

This resource of type NhlTFont specifies the font used to render line labels.

Default: "pwritx"

cnLineLabelFontAspectF

This resource determines the shape of the line label characters. Values increasing from 1.0 result in thinner characters. Values decreasing from 1.0 make the characters wider. Values less than or equal to 0.0 result in a WARNING message and a restoration of the default value.

Default: 1.3125

cnLineLabelFontThicknessF

Sets the thickness of the line used to draw line label text. The value acts as a multiplier of a (device-dependent) unit thickness. This resource is ignored when the *cnLineLabelLabelFont* specifies a filled font (font indexes 21-22, 25-26, 29-30, and 33-37).

Default: 1.0

cnLineLabelFontQuality

This resource of type NhlTFontQuality determines the quality of the font used to draw **ContourPlot** line labels.

Default: `High`

cnLineLabelConstantSpacingF

Normally when *cnLineLabelFontQuality* is set to `High`, the **ContourPlot** object writes line label text with proportional spacing. Setting the *cnLineLabelConstantSpacingF* to a value greater than 0.0 overrides this behavior. Instead, the distance from the start of one character to the next is computed by multiplying a single standard character width by the value of *cnLineLabelConstantSpacingF*. When *cnLineLabelConstantSpacingF* has a value between 0.0 and 1.0, characters will overlap. A value of 1.0 implies, on average, no space between characters, while values increasing from 1.0 cause the space between characters to grow. This parameter is ignored when *cnLineLabelFontQuality* is not `High`. Values less than 0.0 result in an error and are replaced with the default value.

Default: 0.0

cnLineLabelAngleF

When *cnLineLabelAngleF* has a value less than 0.0, **ContourPlot** line labels are angled in the direction of the tangent of the contour line at the location of the label. Otherwise, *cnLineLabelAngleF* specifies the angle, in degrees, of all contour line labels. If *cnLineLabelPlacementMode* is set to `Constant`, this resource has no effect: the line labels are always placed along the contour line.

Default: -1.0

cnLineLabelFuncCode

This resource of type NhlTCharacter sets the function code character that the low level utilities will use when parsing the label string.

Default: **:**

cnLineLabelBackgroundColor

This resource sets the background color used to fill the box surrounding each line label. If you do not want the box to be filled at all, set *cnLineLabelBackgroundColor* to `Transparent` (-1). If *cnLineLabelPlacementMode* is set to `Constant`, this resource has no effect: a boxed background is not available.

Default: `Background`

cnLineLabelPerimOn

cnLineLabelPerimOn is a boolean resource that determines whether **ContourPlot** will draw an outline around the perimeter of the box surrounding contour line labels. If set False, no outline will be drawn. If *cnLineLabelPlacementMode* is set to `Constant`, this resource has no effect: no perimeter can be drawn around the label.

Default: True

cnLineLabelPerimSpaceF

cnLineLabelPerimSpaceF determines the spacing or margin between the text of the line label and the edge of the line label box as a fraction of the current line label text height. If *cnLineLabelPlacementMode* is set to `Constant`, this resource has no effect: no perimeter can be drawn around the label.

Default: 0.33

cnLineLabelPerimColor

This resource sets the HLU color index used to draw the perimeter of the line label box. If *cnLineLabelPlacementMode* is set to `Constant`, this resource has no effect: no perimeter can be drawn around the label.

Default: `Foreground`

cnLineLabelPerimThicknessF

This resource determines the thickness of the perimeter line around the line label box. The value acts as a multiplier of a (device-dependent) unit thickness. If *cnLineLabelPlacementMode* is set to `Constant`, this resource has no effect: no perimeter can be drawn around the label.

Default: 1.0

cnHighLabelsOn

If this boolean resource is set True, **ContourPlot** places labels at regions representing local maximums in the dataset.

Default: False

cnHighLabelString

Specifies the string to use when drawing high labels. The string may contain function codes and/or substitution substrings. **ContourPlot** will replace the substring "ZDV" with a number representing the value at the point where the label is to appear. *cnHighLabelFormat* determines the format of the number and *cnLabelScaleFactorF* determines the scaling of the actual data value.

Default: "H:B:ZDV:E:"

cnHighLabelFormat

The *cnHighLabelFormat* resource is a string that specifies the printing format for the numeric portion of high labels according to the HLU Floating Point Format Specification scheme. If any of the numerical format parameters have the dynamic attribute on, the corresponding parameter in the *cnMaxDataValueFormat* resource determines the parameter's value. The default value uses the shorthand notation '*+' to cause all numeric format parameters to be determined dynamically.

Default: "*+^sg"

cnHighLabelFontHeightF

This resource controls the height, in NDC units, of characters used in the text of high labels. The character width scales proportionally, unless you also modify the aspect ratio using the *cnHighLabelFontAspectF* resource. The high label text height scales with changes to the viewport width, unless you explicitly set *cnHighLabelFontHeightF* during the same call.

Default: <dynamic> -- 0.012 for a viewport width of 0.6

cnHighLabelFont

This resource of type NhlTFont specifies the font used to render high labels.

Default: "pwritx"

cnHighLabelFontColor

This resource specifies the HLU color index used to render high label text. If *cnHighUseLineLabelRes* is True, this resource takes its value from the first element of the *cnLineLabelFontColors* array.

Default: True

cnHighLabelFontAspectF

This resource determines the shape of the high label characters. Values increasing from 1.0 result in thinner characters. Values decreasing from 1.0 make the characters wider. Values less than or equal to 0.0 result in a WARNING message and a restoration of the default value.

Default: 1.3125

cnHighLabelFontThicknessF

Sets the thickness of the line used to draw high label text. The value acts as a multiplier of a (device-dependent) unit thickness. This resource is ignored when the *cnHighLabelFont* specifies a filled font (font indexes 21-22, 25-26, 29-30, and 33-37).

Default: 1.0

cnHighLabelFontQuality

This resource of type NhlTFontQuality determines the quality of the font used to draw **ContourPlot** high labels.

Default: `High`

cnHighLabelConstantSpacingF

Normally when *cnLineLabelFontQuality* is set to `High`, the **ContourPlot** object writes line label text with proportional spacing. Setting the *cnLineLabelConstantSpacingF* to a value greater than 0.0 overrides this behavior. Instead, the distance from the start of one character to the next is computed by multiplying a single standard character width by the value of *cnLineLabelConstantSpacingF*. When *cnLineLabelConstantSpacingF* has a value between 0.0 and 1.0, characters will overlap. A value of 1.0 implies, on average, no space between characters, while values increasing from 1.0 cause the space between characters to grow. This parameter is ignored when *cnLineLabelFontQuality* is not `High`. Values less than 0.0 result in an error and are replaced with the default value.

Default: 0.0

cnHighLabelAngleF

This resource specifies the angle, in degrees, of high label text and its surrounding box.

Default: 0.0

cnHighLabelFuncCode

This resource of type NhlTCharacter sets the function code character that the low level utilities will use when parsing the high label string.

Default: **:**

cnHighLabelBackgroundColor

This resource sets the background color used to fill the box surrounding each high label. If you do not want the box to be filled at all, set *cnHighLabelBackgroundColor* to `Transparent` (-1).

Default: `Background`

cnHighLabelPerimOn

cnHighLabelPerimOn is a boolean resource that determines whether **ContourPlot** will draw an outline around the perimeter of the box surrounding contour high labels. If set False, no outline will be drawn.

Default: True

cnHighLabelPerimSpaceF

cnHighLabelPerimSpaceF determines the spacing or margin between the text of the high label and the edge of the high label box as a fraction of the current high label text height.

Default: 0.33

cnHighLabelPerimColor

This resource sets the HLU color index used to draw the perimeter of the high label box.

Default: `Foreground`

cnHighLabelPerimThicknessF

This resource determines the thickness of the perimeter line around the high label box. The value acts as a multiplier of a (device-dependent) unit thickness.

Default: 1.0

cnLowLabelsOn

If this boolean resource is set True, **ContourPlot** places labels at regions representing local minimums in the dataset.

Default: True

cnLowLabelString

Specifies the string to use when drawing low labels. The string may contain function codes and/or substitution substrings. **ContourPlot** will replace the substring "ZDV" with a number representing the value at the point where the label is to appear. *cnLowLabelFormat* determines the format of the number and *cnLabelScaleFactorF* determines the scaling of the actual data value.

Default: "L:B:ZDV:E:"

cnLowLabelFormat

The *cnLowLabelFormat* resource is a string that specifies the printing format for the numeric portion of low labels according to the HLU Floating Point Format Specification scheme. If any of the numerical format parameters have the dynamic attribute on, the corresponding parameter in the *cnMaxDataValueFormat* resource determines the parameter's value. The default value uses the shorthand notation '*+' to cause all numeric format parameters to be determined dynamically.

Default: "*+^sg"

cnLowLabelFontHeightF

This resource controls the height, in NDC units, of characters used in the text of low labels. The character width scales proportionally, unless you also modify the aspect ratio using the *cnLowLabelFontAspectF* resource. The low label text height scales with changes to the viewport width, unless you explicitly set *cnLowLabelFontHeightF* during the same call.

Default: <dynamic> -- 0.012 for a viewport width of 0.6

cnLowLabelFont

This resource of type NhlTFont specifies the font used to render low labels.

Default: "pwritx"

cnLowLabelFontColor

This resource specifies the HLU color index used to render low label text.

Default: True

cnLowLabelFontAspectF

This resource determines the shape of the low label characters. Values increasing from 1.0 result in thinner characters. Values decreasing from 1.0 make the characters wider. Values less than or equal to 0.0 result in a WARNING message and a restoration of the default value.

Default: 1.3125

cnLowLabelFontThicknessF

Sets the thickness of the line used to draw low label text. The value acts as a multiplier of a (device-dependent) unit thickness. This resource is ignored when the *cnLowLabelFont* specifies a filled font (font indexes 21-22, 25-26, 29-30, and 33-37).

Default: 1.0

cnLowLabelFontQuality

This resource of type NhlTFontQuality determines the quality of the font used to draw **ContourPlot** low labels.

Default: `High`

cnLowLabelConstantSpacingF

Normally when *cnLineLabelFontQuality* is set to `High`, the **ContourPlot** object writes line label text with proportional spacing. Setting the *cnLineLabelConstantSpacingF* to a value greater than 0.0 overrides this behavior. Instead, the distance from the start of one character to the next is computed by multiplying a single standard character width by the value of *cnLineLabelConstantSpacingF*. When *cnLineLabelConstantSpacingF* has a value between 0.0 and 1.0, characters will overlap. A value of 1.0 implies, on average, no space between characters, while values increasing from 1.0 cause the space between characters to grow. This parameter is ignored when *cnLineLabelFontQuality* is not `High`. Values less than 0.0 result in an error and are replaced with the default value.

Default: 0.0

cnLowLabelAngleF

This resource specifies the angle, in degrees, of low label text and its surrounding box.

Default: 0.0

cnLowLabelFuncCode

This resource of type NhlTCharacter sets the function code character that the low level utilities will use when parsing the low label string.

Default: **:**

cnLowLabelBackgroundColor

This resource sets the background color used to fill the box surrounding each low label. If you do not want the box to be filled at all, set *cnLowLabelBackgroundColor* to `Transparent` (-1).

Default: `Background`

cnLowLabelPerimOn

cnLowLabelPerimOn is a boolean resource that determines whether **ContourPlot** will draw an outline around the perimeter of the box surrounding contour low labels. If set False, no outline will be drawn.

Default: True

cnLowLabelPerimSpaceF

cnLowLabelPerimSpaceF determines the spacing or margin between the text of the low label and the edge of the low label box as a fraction of the current low label text height.

Default: 0.33

cnLowLabelPerimColor

This resource sets the HLU color index used to draw the perimeter of the low label box.

Default: `Foreground`

cnLowLabelPerimThicknessF

This resource determines the thickness of the perimeter line around the low label box. The value acts as a multiplier of a (device-dependent) unit thickness.

Default: 1.0

cnInfoLabelOn

If this boolean resource is set False, **ContourPlot** will not draw an informational label.

Default: True

cnInfoLabelString

Specifies the string to use when drawing an informational label. The string may contain function codes and/or substitution substrings. **ContourPlot** will replace the following substrings with numeric values:

CIU
: The contour interval used (value of *cnLevelSpacing*)

CMN
: The minimum contour level (first element of *cnLevels*)

CMX
: The maximum contour level

SFU
: The scale factor used (value of *cnLabelScaleFactorF*)

ZMN
: The minimum data value

ZMX
: The maximum data value

cnInfoLabelFormat determines the format of each number and except for the number generated from SFU, all values are scaled based on the value of *cnLabelScaleFactorF*.

Default: "CONTOUR FROM CMN TO CMX BY CIU"

cnInfoLabelFormat

The *cnInfoLabelFormat* resource is a string that specifies the printing format for the numbers generated from substitution substrings in *cnInfoLabelString* according to the HLU Floating Point Format Specification scheme. If any of the numerical format parameters have the dynamic attribute on, the corresponding parameter in the *cnMaxDataValueFormat* resource determines the parameter's value. The default value uses the shorthand notation '*+' to cause all numeric format parameters to be determined dynamically.

Default: "*+^sg"

cnInfoLabelTextDirection

This resource of type NhlTTextDirection specifies the direction of the text in the informational label. The choices are:

`Down`

Each character is placed below the previous character in the text string.

`Across`

Each character is placed to the right of the previous character in the text string.

These directions apply before any rotation due to *cnInfoLabelAngleF* is applied to the informational label.

Default: `Across`

cnInfoLabelFontHeightF

This resource controls the height, in NDC units, of characters used in the text of the informational label. The character width scales proportionally, unless you also modify the aspect ratio using the *cnInfoLabelFontAspectF* resource. The info label text height scales with changes to the viewport width, unless you explicitly set *cnInfoLabelFontHeightF* during the same call.

Default: <dynamic> -- 0.012 for a viewport width of 0.6

cnInfoLabelFont

This resource of type NhlTFont specifies the font used to render the informational label.

Default: "pwritx"

cnInfoLabelFontColor

This resource specifies the HLU color index used to render informational label text.

Default: `Foreground`

cnInfoLabelFontAspectF

This resource determines the shape of the informational label characters. Values increasing from 1.0 result in thinner characters. Values decreasing from 1.0 make the characters wider. Values less than or equal to 0.0 result in a WARNING message and a restoration of the default value.

Default: 1.3125

cnInfoLabelFontThicknessF

Sets the thickness of the line used to draw informational label text. The value acts as a multiplier of a (device-dependent) unit thickness. This resource is ignored when the *cnInfoLabelFont* specifies a filled font (font indexes 21-22, 25-26, 29-30, and 33-37).

Default: 1.0

cnInfoLabelFontQuality

This resource of type NhlTFontQuality determines the quality of the font used to draw the **ContourPlot** informational label.

Default: `High`

cnInfoLabelConstantSpacingF

Normally when *cnLineLabelFontQuality* is set to `High`, the **ContourPlot** object writes informational label text with proportional spacing. Setting the *cnInfoLabelConstantSpacingF* to a value greater than 0.0 overrides this behavior. Instead, the distance from the start of one character to the next is computed by multiplying a single standard character width by the value of *cnLineLabelConstantSpacingF*. When *cnLineLabelConstantSpacingF* has a value between 0.0 and 1.0, characters will overlap. A value of 1.0 implies, on average, no space between characters, while values increasing from 1.0 cause the space between characters to grow. This parameter is ignored when *cnInfoLabelFontQuality* is not `High`. Values less than 0.0 result in an error and are replaced with the default value.

Default: 0.0

cnInfoLabelAngleF

This resource specifies the angle, in degrees, of the informational label text and its surrounding box.

Default: 0.0

cnInfoLabelFuncCode

This resource of type NhlTCharacter sets the function code character that the low level utilities will use when parsing the informational label string.

Default: **:**

cnInfoLabelBackgroundColor

This resource sets the background color used to fill the box surrounding the informational label. If you do not want the box to be filled at all, set *cnInfoLabelBackgroundColor* to `Transparent` (-1).

Default: `Background`

cnInfoLabelPerimOn

cnInfoLabelPerimOn is a boolean resource that determines whether **ContourPlot** will draw an outline around the perimeter of the box surrounding a contour informational label. If set False, no outline will be drawn.

Default: True

cnInfoLabelPerimSpaceF

cnInfoLabelPerimSpaceF determines the spacing or margin between the text of the informational label and the edge of the informational label box as a fraction of the current label text height.

Default: 0.33

cnInfoLabelPerimColor

This resource sets the HLU color index used to draw the perimeter of the informational label box.

Default: `Foreground`

cnInfoLabelPerimThicknessF

This resource determines the thickness of the perimeter line around the informational label box. The value acts as a multiplier of a (device-dependent) unit thickness.

cnInfoLabelZone

ContourPlot implements the informational label as an embedded annotation. *cnInfoLabelZone* specifies the **PlotManager** zone assigned to the informational annotation. The PlotManager

Location Control Model requires this resource to allow control of the informational label consistent with other annotations. If *cnInfoLabelZone* is set to 0, the positional origin is the center of the plot viewport; otherwise it is on or outside one of the sides of the viewport.

If you create a **ContourPlot** object without an active **PlotManager** by setting *tfPlotManagerOn* False, then **ContourPlot** manages the informational annotation by itself. In this case, the *cnInfoLabelZone* resource is not as meaningful.

Default: 3

cnInfoLabelSide

This resource of type NhlTPosition determines where to place the informational annotation in relation to the sides of the plot object. The PlotManager Location Control Model requires this resource to allow control of the informational label consistent with other annotations. Its value determines a coordinate system origin and the direction of the positional resources, *cnInfoLabelParallelPosF* and *cnInfoLabelOrthogonalPosF*. It also constrains the value of *cnInfoLabelJust* appropriately. There are four settings that behave as follows, unless *cnInfoLabelZone* is set to one of the special zones (0 or 1):

`Top`

PlotManager locates the informational label annotation relative to a line paralleling the top viewport boundary. *cnInfoLabelOrthogonalPosF* increases in the direction of increasing NDC Y-Axis values and is constrained to positive values. *cnInfoLabelParallelPosF* increases in the direction of increasing NDC X-Axis values. *cnInfoLabelJust* is constrained to `BottomRight`, `BottomCenter`, or `BottomLeft`.

`Bottom`

PlotManager locates the informational label annotation relative to a line that parallels the bottom viewport boundary. *cnInfoLabelOrthogonalPosF* increases in the direction of decreasing NDC Y-Axis values and is constrained to positive values. *cnInfoLabelParallelPosF* increases in the direction of increasing NDC X-Axis values. *cnInfoLabelJust* is constrained to `TopRight`, `TopCenter`, or `TopLeft`.

`Right`

PlotManager locates the informational label annotation relative to a line that parallels the right viewport boundary. *cnInfoLabelOrthogonalPosF* increases in the direction of increasing NDC X-Axis values and is constrained to positive values. *cnInfoLabelParallelPosF* increases in the direction of increasing NDC Y-Axis values. *cnInfoLabelJust* is constrained to `TopLeft`, `CenterLeft`, or `BottomLeft`.

`Left`

PlotManager locates the informational label annotation relative to a line that parallels the left viewport boundary. *cnInfoLabelOrthogonalPosF* increases in the direction of decreasing NDC X-Axis values and is constrained to positive values. *cnInfoLabelParallelPosF* increases in the direction of increasing NDC Y-Axis values. *cnInfoLabelJust* is constrained to `TopRight`, `CenterRight`, or `BottomRight`.

If **cnInfoLabelZone** is set to 0, the **PlotManager** locates the informational label annotation relative to the viewport center. If **cnInfoLabelZone** is 1, the direction of the *cnInfoLabelOrthogonalPosF* is opposite to the specification given above. Also if the **cnInfoLabelZone** is either 0 or 1, *cnInfoLabelJust* is not constrained, and *cnInfoLabelOrthogonalPosF* may take on negative values.

Default: `Bottom`

cnInfoLabelParallelPosF

cnInfoLabelParallelPosF specifies the coordinate of the base location of the informational label annotation parallel to the current *cnInfoLabelSide* and directed toward increasing NDC values. The PlotManager Location Control Model requires this resource to allow control of the informational label consistent with other annotations.

Default: 1.0

cnInfoLabelOrthogonalPosF

cnInfoLabelOrthogonalPosF sets the coordinate of the base location of the informational label annotation orthogonal to the current *cnInfoLabelSide* and directed away from the center of the viewport. The PlotManager Location Control Model requires this resource to allow control of the informational label consistent with other annotations.

Default: 0.02

cnInfoLabelJust

This resource of type NhlTJustification, after constraint to an appropriate value based on *cnInfoLabelSide*, sets the justification of the informational label annotation with respect to its base location. The PlotManager Location Control Model requires this resource to allow control of the informational label consistent with other annotations.

Default: `TopRight`

cnNoDataLabelOn

This boolean resource, when set True, causes a label to appear when **ContourPlot** is drawn without any data having been provided. Except for the label string, all attributes of this label, including its position, are set using resources belonging to the constant field label. When set False, no such label appears.

Default: True

cnNoDataLabelString

This resource contains the string that appears in the 'No Data' label if you draw a **ContourPlot** object without providing any data. No substitution substrings are allowed in this label, since all the substitutions depend on data being available. Except for the boolean switch that turns it on and off, all attributes of this label, including its position, are set using resources belonging to the constant field label.

Default: "NO CONTOUR DATA"

cnConstFLabelOn

The **ContourPlot** object draws a constant field label annotation only when *cnConstFLabelOn* is set True and the **ScalarField** data are determined to have a single constant value within the limits of the available precision or when no **ScalarField** data are supplied.

Default: True

cnConstFLabelString

Specifies the string to use when drawing a constant field label. The string may contain function codes and/or substitution substrings. **ContourPlot** will replace the substring ZDV with a number representing the constant field value. *cnConstFLabelFormat* determines the format of the number; its value will be scaled based on the value of *cnLabelScaleFactorF*.

Default: "CONSTANT FIELD - VALUE IS ZDV"

cnConstFLabelFormat

The *cnConstFLabelFormat* resource is a string that specifies the printing format for the number generated from a substitution substring in the *cnConstFLabelString* according to the HLU Floating Point Format Specification scheme. If any of the numerical format parameters have the dynamic attribute on, the corresponding parameter in the *cnMaxDataValueFormat* resource determines the parameter's value. The default value uses the shorthand notation '*+' to cause all numeric format parameters to be determined dynamically.

Default: "*+^sg"

cnConstFLabelTextDirection

This resource of type NhlTTextDirection specifies the direction of the text in the constant field label. The choices are:

`Down`

Each character is placed below the previous character in the text string.

`Across`

Each character is placed to the right of the previous character in the text string.

These directions apply before rotation due to *cnConstFLabelAngleF*.

Default: `Across`

cnConstFLabelFontHeightF

This resource controls the height, in NDC units, of characters used in the text of the constant field label. The character width scales proportionally, unless you also modify the aspect ratio using the *cnConstFLabelFontAspectF* resource. The constant field label text height scales with changes to the viewport width, unless you explicitly set *cnConstFLabelFontHeightF* during the same call.

Default: <dynamic> -- 0.012 for a viewport width of 0.6

cnConstFLabelFont

This resource of type NhlTFont specifies the font used to render the constant field label.

Default: "pwritx"

cnConstFLabelFontColor

This resource specifies the HLU color index used to render constant field label text.

Default: True

cnConstFLabelFontAspectF

This resource determines the shape of the constant field label characters. Values increasing from 1.0 result in thinner characters. Values decreasing from 1.0 make the characters wider. Values less than or equal to 0.0 result in a WARNING message and a restoration of the default value.

Default: 1.3125

cnConstFLabelFontThicknessF

Sets the thickness of the line used to draw constant field label text. The value acts as a multiplier of a (device-dependent) unit thickness. This resource is ignored when the *lbLabelFont* specifies a filled font (font indexes 21-22, 25-26, 29-30, and 33-37).

Default: 1.0

cnConstFLabelFontQuality

This resource of type NhlTFontQuality determines the quality of the font used to draw the **ContourPlot** constant field label.

Default: `High`

cnConstFLabelConstantSpacingF

Normally when *cnLineLabelFontQuality* is set to `High`, the **ContourPlot** object writes constant field label text with proportional spacing. Setting the *cnConstFLabelConstantSpacingF* to a value greater than 0.0 overrides this behavior. Instead, the distance from the start of one character to the next is computed by multiplying a single standard character width by the value of *cnLineLabelConstantSpacingF*. When *cnLineLabelConstantSpacingF* has a value between 0.0 and 1.0, characters will overlap. A value of 1.0 implies, on average, no space between characters, while values increasing from 1.0 cause the space between characters to grow. This parameter is ignored when *cnConstFLabelFontQuality* is not `High`. Values less than 0.0 result in an error and are replaced with the default value.

Default: 0.0

cnConstFLabelAngleF

This resource specifies the angle, in degrees, of the constant field label text and its surrounding box.

Default: 0.0

cnConstFLabelFuncCode

This resource of type NhlTCharacter sets the function code character that the low level utilities will use when parsing the constant field label string.

Default: **:**

cnConstFLabelBackgroundColor

This resource sets the background color used to fill the box surrounding the constant field label. If you do not want the box to be filled at all, set *cnConstFLabelBackgroundColor* to `Transparent` (-1).

Default: `Background`

cnConstFLabelPerimOn

cnConstFLabelPerimOn is a boolean resource that determines whether **ContourPlot** will draw an outline around the perimeter of the box surrounding the contour constant field label. If set False, no outline will be drawn.

Default: True

cnConstFLabelPerimSpaceF

cnConstFLabelPerimSpaceF determines the spacing or margin between the text of the constant field label and the edge of the constant field label box as a fraction of the current label text height.

Default: 0.33

cnConstFLabelPerimColor

This resource sets the HLU color index used to draw the perimeter of the constant field label box.

Default: `Foreground`

cnConstFLabelPerimThicknessF

This resource determines the thickness of the perimeter line around the constant field label box. The value acts as a multiplier of a (device-dependent) unit thickness.

Default: 1.0

cnConstFLabelZone

ContourPlot implements the constant field label as an embedded annotation. *cnConstFLabelZone* specifies the **PlotManager** zone assigned to the constant field annotation. The PlotManager Location Control Model requires this resource to allow control of the constant field label consistent with other annotations. If *cnConstFLabelZone* is set to 0, the positional origin is the center of the plot viewport; otherwise it is on or outside one of the sides of the viewport.

If you create a **ContourPlot** object without an active **PlotManager** by setting *tfPlotManagerOn* False, then **ContourPlot** manages the constant field annotation by itself. In this case, the *cnConstFLabelZone* resource is not as meaningful.

Default: 0

cnConstFLabelSide

This resource of type NhlTPosition determines where to place the constant field annotation in relation to the sides of the plot object. The PlotManager Location Control Model requires this resource to allow control of the constant field label consistent with other annotations. Its value determines a coordinate system origin and the direction of the positional resources, *cnConstFLabelParallelPosF* and *cnConstFLabelOrthogonalPosF*. It also constrains the value of the *cnConstFLabelJust* appropriately. There are four settings, which behave as follows, unless *cnConstFLabelZone* is set to one of the special zones (0 or 1):

`Top`

PlotManager locates the constant field label annotation relative to a line that parallels the top viewport boundary. *cnConstFLabelOrthogonalPosF* increases in the direction of increasing NDC Y-Axis values and is constrained to positive values. *cnConstFLabelParallelPosF* increases in the direction of increasing NDC X-Axis values. *cnConstFLabelJust* is constrained to `BottomRight`, `BottomCenter`, or `BottomLeft`.

`Bottom`

PlotManager locates the constant field label annotation relative to a line that parallels the bottom viewport boundary. *cnConstFLabelOrthogonalPosF* increases in the direction of decreasing NDC Y-Axis values and is constrained to positive values. *cnConstFLabelParallelPosF* increases in the direction of increasing NDC X-Axis values. *cnConstFLabelJust* is constrained to `TopRight`, `TopCenter`, or `TopLeft`.

`Right`

PlotManager locates the constant field label annotation relative to a line that parallels the right viewport boundary. *cnConstFLabelOrthogonalPosF* increases in the direction of increasing NDC X-Axis values and is constrained to positive values. *cnConstFLabelParallelPosF* increases in the direction of increasing NDC Y-Axis values. *cnConstFLabelJust* is constrained to `TopLeft`, `CenterLeft`, or `BottomLeft`.

`Left`

PlotManager locates the constant field label annotation relative to a line that parallels the left viewport boundary. *cnConstFLabelOrthogonalPosF* increases in the direction of decreasing NDC X-Axis values and is constrained to positive values. *cnConstFLabelParallelPosF* increases in the direction of increasing NDC Y-Axis values. *cnConstFLabelJust* is constrained to `TopRight`, `CenterRight`, or `BottomRight`.

If **cnConstFLabelZone** is set to 0, the **PlotManager** locates the constant field label annotation relative to the viewport center. If **cnConstFLabelZone** is 1, the direction of the *cnConstFLabelOrthogonalPosF* is opposite to the specification given above. Also if the **cnConstFLabelZone** is either 0 or 1, *cnConstFLabelJust* is not constrained, and *cnConstFLabelOrthogonalPosF* may take on negative values.

Default: `Bottom`

cnConstFLabelParallelPosF

cnConstFLabelParallelPosF specifies the coordinate of the base location of the constant field label annotation parallel to the current *cnConstFLabelSide* and directed toward increasing NDC values. The PlotManager Location Control Model requires this resource to allow control of the constant field label consistent with other annotations.

Default: 0.0

cnConstFLabelOrthogonalPosF

cnConstFLabelOrthogonalPosF sets the coordinate of the base location of the constant field label annotation orthogonal to the current *cnConstFLabelSide* and directed away from the center of the viewport. The PlotManager Location Control Model requires this resource to allow control of the constant field label consistent with other annotations.

Default: 0.0

cnConstFLabelJust

This resource of type NhlTJustification, after constraint to an appropriate value based on *cnConstFLabelSide*, sets the justification of the constant field label annotation with respect to its base location. The PlotManager Location Control Model requires this resource to allow control of the constant field label consistent with other annotations.

Default: `CenterCenter`

cnMissingValPerimOn

If set True, this boolean resource specifies that perimeter lines be drawn around missing value areas in the **ContourPlot** object.

Default: False

cnMissingValPerimThicknessF

This resource determines the thickness of perimeter lines around missing value areas. The value acts as a multiplier of a (device-dependent) unit thickness.

Default: 1.0

cnMissingValPerimDashPattern

This resource sets the HLU index of a dash pattern used to render perimeter lines around missing value areas.

Default: 0

cnMissingValPerimColor

This resource sets the HLU index of the color used to render perimeter lines around missing value areas.

Default: `Foreground`

cnMissingValFillColor

This resource sets the HLU index of the color used to fill missing value areas. If *cnMissingValFillColor* has the value `Transparent` (-1), there will be no fill of the missing value area, regardless of the setting of *cnMissingValFillPattern*.

Default: `Background`

cnMissingValFillPattern

This resource sets the HLU index of the fill pattern used to fill missing value areas. If *cnMissingValFillPattern* has the value `HollowFill` (-1), there will be no fill of the missing value area, regardless of the setting of *cnMissingValFillColor*.

Default: `HollowFill`

cnMissingValFillScaleF

The *cnMissingValFillScaleF* resource controls the scaling of the pattern used for missing values fill areas. It has no effect for solid fill (fill pattern index `Solid` or 0). Values greater than 1.0 make the pattern spacing bigger than the default spacing, resulting in a fill that appears less dense. Values less than 1.0 have the opposite effect. Values less than or equal to 0.0 are invalid, generate a warning message, and are reset to the default value, 1.0.

Default: 1.0

cnGridBoundPerimOn

If set True, this boolean resource specifies that a perimeter line be drawn around the grid boundary (extent of the **ScalarField** data) in the **ContourPlot** plot.

Default: False

cnGridBoundPerimThicknessF

This resource determines the thickness of the perimeter line around the grid boundary. The value acts as a multiplier of a (device-dependent) unit thickness.

Default: 1.0

cnGridBoundPerimDashPattern

This resource sets the HLU index of a dash pattern used to render the perimeter line around the grid boundary.

Default: 0

cnGridBoundPerimColor

This resource sets the HLU index of the color used to render the perimeter line around the grid boundary.

Default: `Foreground`

cnOutOfRangePerimOn

If set True, this boolean resource specifies that perimeter lines be drawn around areas where the **ScalarField** data falls outside the current data boundaries of the plot.

Default: False

cnOutOfRangePerimThicknessF

This resource determines the thickness of perimeter lines around the edge of out-of-range areas. The value acts as a multiplier of a (device-dependent) unit thickness.

Default: 1.0

cnOutOfRangePerimDashPattern

This resource sets the HLU index of a dash pattern used to render perimeter lines around the edge of out-of-range areas.

Default: 0

cnOutOfRangePerimColor

This resource sets the HLU index of the color used to render perimeter lines around the edge of out-of-range areas.

Default: `Foreground`

cnSmoothingOn

When the boolean resource *cnSmoothingOn* is True, the **ContourPlot** object will smooth the contours using cubic splines under tension. The resources *cnSmoothingTensionF* and *cnSmoothingDistanceF* control the parameters of the smoothing algorithm. Note that when smoothing is turned on, it is possible in some cases for adjacent contour lines to cross each other. In this case, you need to adjust the *cnSmoothingTensionF* experimentally in order to eliminate the problem.

Default: False

cnSmoothingTensionF

If you give *cnSmoothingTensionF* a value of 0.0, smoothing is turned off. If you give it a negative value, the **ContourPlot** object performs smoothing prior to any transformations of the data space; if you make it positive, smoothing takes place after any data space transformations. Small absolute values on the order of 0.001 give 'loopy' curves, which are rather likely to cross each other. Large absolute values give tight, nearly polygonal curves.

Note that if *cnMaxPointDistanceF* is set to a non-zero value, smoothing will not work properly unless it is applied prior to the transformation of the data space. Therefore you should always set *cnSmoothingTensionF* to a negative value when *cnMaxPointDistanceF* is non-zero.

V4.1 Status Note 4

Default: -2.5

cnSmoothingDistanceF

cnSmoothingDistanceF sets the distance between points used to draw smoothed contour lines. It is expressed as a fraction of the width of the window in the coordinate system in which smoothing is being performed. Note that processing time is affected greatly by the value given to the smoothing distance. Smaller values add to the processing time.

Default: 0.01

cnMaxPointDistanceF

Controls the maximum distance along both the width and height of the viewport allowed before checking for a discontinuous jump in the mapped coordinate points. The value specifies a fraction of the view width for the width check and a fraction of the view height for the height check. A value of 0.0 for this resource turns off this form of checking altogether. When non-zero, this value should not be set less than 0.001 or greater than 0.1. Smaller values increase processing time.

Note that if *cnMaxPointDistanceF* is set to a non-zero value, contour smoothing will not work properly unless it is applied prior to the transformation of the data space. Therefore you should always set *cnSmoothingTensionF* to a negative value when *cnMaxPointDistanceF* is non-zero.

Default: 0.05

cnExplicitLabelBarLabelsOn

This boolean resource allows you control the labels that appear in the **ContourPlot LabelBar** explicitly. When set True, **ContourPlot** does not block the **LabelBar** resources *lbLabelStrings* and *lbLabelAlignment*. Therefore you can directly control both the contents of the **LabelBar's** label strings and their alignment with respect to the label boxes. When *cnExplicitLabelBarLabelsOn* is set False, **ContourPlot** sets both of these resources based on the current contour line labels and the value of the resource *cnLabelBarEndLabelsOn*. If you set this resource True but do not set the *lbLabelStrings* array resource, **ContourPlot** will set it for you one time. This allows you to get an initial set of strings as a starting point for any customized tweaking you want to perform.

Default: False

cnLabelBarEndLabelsOn

When this boolean resource is set True and *cnExplicitLabelBarLabelsOn* is False, **ContourPlot** creates labels for the two ends of the **LabelBar**. The label at one end will be a string representation of the minimum value in the dataset, and at the other end will be a string representation of the maximum value in the dataset. Both strings will be formatted according to the format specification in effect for the other labels provided by **ContourPlot** to the **LabelBar**.

Default: False

cnExplicitLegendLabelsOn

This boolean resource allows you control the labels that appear in the **ContourPlot Legend** explicitly. When set True, **ContourPlot** does not block the **Legend** resource *lgLabelStrings*. Therefore you can directly control the content of the **Legend's** label strings. When *cnExplicitLegendLabelsOn* is set False, **ContourPlot** sets *lgLabelStrings* based on the value of the current contour line labels. If you set this resource True but do not set the *lbLabelStrings* array resource, **ContourPlot** will set it for you one time. This allows you to get an initial set of strings as a starting point for any customized tweaking you want to perform.

Default: False

cnLegendLevelFlags

This array resource of type NhlTcnLevelUseModeGenArray allows you to control which of the lines representing contour levels are to appear in the **ContourPlot Legend**. You can also control whether the lines that do appear are to get a line label. Although the type allows for four choices, this resource supports only three of them:

`NoLine`

Do not draw a line representing this contour level in the **Legend**.

`LineOnly`

Draw a line representing this contour level, but no label in the **Legend**.

`LabelOnly`

Currently, if an element is set to this value, it is treated the same as if it were set to `NoLine`.

`LineAndLabel`

Draw a line representing this contour level along with a line label in the **Legend**.

If you set this resource with an array containing fewer elements than the current number of contour levels, additional elements are supplied and set to the value `NoLine`.

Default: NULL

cnConpackParams

This string array resource allows you limited access to a number of parameters belonging to the LLU **Conpack** package, in order to control certain plot features for which **ContourPlot** does not yet have native resources. Each element of the array is a string consisting of a Conpack parameter name and its desired value separated by the colon character. Here is an example of its use in a resource file:

```
*cnConpackParams: (/ RC1:0.05 , RC2:0.1 , RC3:0.05 , \
        PC1:7.0 , PC2:20.0 , PC3:120.0 , PC4:0.01 , PC5:0.05 , PC6:0.05 , \
        PW1:0.0 , PW2:0.0 , PW3:0.0 , PW4:0.0 /)
```

If set programmatically, each string element would need to quoted as appropriate for the source language. The example given above includes all the parameters that affect the density of line labels when the *cnLineLabelPlacementMode* is set to `Randomized` or `Computed`. The example values should normally result in the appearance of more labels than the default settings. See the **Conpack** parameter descriptions for more information.

Note that the only **Conpack** parameters accessible using the *cnConpackParams* resource are ones that **ContourPlot** does not set internally and are unlikely to have an interaction with parameters that **ContourPlot** does set. Here is a categorized list of the **Conpack** parameters that can be set using *cnConpackParams*:

Contour hachuring control

- HCL
- HCS
- HCF

High/low label density

- HLX
- HLY

`Computed` placement mode ("Penalty scheme" in Conpack) label density

- PC1
- PC2
- PC3
- PC4
- PC5
- PC6
- PW1
- PW2
- PW3
- PW4

`Randomized` placement mode ("Regular scheme" in Conpack) label density

- RC1
- RC2
- RC3

Point interpolation for lines and edges

- PIC
- PIE

Note that you cannot get the value of *cnConpackParams*. Attempts to do so will result in an error.

Default: NULL

CoordArrTable class resource descriptions

ctCopyTables

This resource is used to determine if the **CoordArrTable** object should make its own copy of the *ctXTable* and *ctYTable* data. If this resource is False, it is important for the programmer to keep the data around without modifying it.

Default: True

ctXTable

This resource specifies the **X** values of the X/Y coordinate data. It is an array of **X** vectors. Each vector is actually a single-dimension array of type *ctXTableType*. The length of each vector array is determined by the corresponding element of the *ctXTableLengths* resource. If this resource is set, then the corresponding *ctXTableLengths* resource must also be set. If the *ctXTable* resource is not set, then all of the **Y** values set with the *ctYTable* resource will be paired with their integer array index. For example,

```
(1, (ytable[0])[0])
(2, (ytable[0])[1])
(ytable_lengths[0], (ytable[0])[ytable_lengths[0]-1])
```

If one of the vectors in the *ctXTable* resource is NULL, then it is interpreted to contain an integer array index vector as well, with the *ctXTableLengths* resource indicating the length of the vector.

If the *ctXTable* resource is not set, then the *ctYTable* resource must be set.

Default: NULL

ctXTableLengths

This resource is a one-dimensional integer array. Each element of the array indicates the length of the corresponding vector in the *ctXTable* resource. This resource must be set if the *ctXTable* resource is set.

Default: NULL

ctXTableType

This resource is used to indicate the type of each element of each vector in the *ctXTable* resource.

It is a string value of the type's name. For example, "Integer". There are symbols provided to the C interface for all the available types. "Integer" would be set with *NhlTInteger*.

Default: NULL
This resource must be set if the *ctXTable* resource is set.

ctXElementSize

This resource is used to indicate the size of each element of each vector in the *ctXTable* resource. If the *ctXTable* resource is set, and this one is not, then the **CoordArrTable** object will use the *ctXTableType* resource to try to determine the size of each element; therefore this resource is usually not necessary.

Default: <dynamic>
Determined from *ctXTableType* if possible.

ctXMissingV

This resource indicates a **missing** value for elements in the *ctXTable* resource. When the HLU library is parsing the data in the *ctXTable* elements, it will treat any element with this value as **missing** data. This resource has a *dynamic* type so that elements of any type can be set to it. It is important to realize that the elements in *ctXTable* will be converted to float, and the contents of this resource will be converted to float, before the elements are compared with the value of this resource.

Default: None

ctXMaxV

This resource is used to tell the **CoordArrTable** object the maximum value contained in the *ctXTable* resource. If it is not specified, the **CoordArrTable** object will compute it. This resource has a *dynamic* type so that elements of any type can be set to it.

Default: <*dynamic*>
The value will be computed if it is not set.

ctXMinV

This resource is used to tell the **CoordArrTable** object the minimum value contained in the *ctXTable* resource. If it is not specified, the **CoordArrTable** object will compute it. This resource has a *dynamic* type so that elements of any type can be set to it.

Default: <*dynamic*>
The value will be computed if it is not set.

ctYTable

This resource specifies the **Y** values of the X/Y coordinate data. It is an array of **Y** vectors. Each vector is actually a single-dimension array of type *ctYTableType*. The length of each vector array is determined by the corresponding element of the *ctYTableLengths* resource. If this resource is set, then the corresponding *ctYTableLengths* resource must also be set. If the *ctYTable* resource is not set, then all of the **X** values set with the *ctXTable* resource will be paired with their integer array index. For example,

```
((xtable[0])[0], 1)
((xtable[0])[1]), 2)
((xtable[0])[xtable_lengths[0]-1],xtable_lengths[0])
```

If one of the vectors in the *ctYTable* resource is NULL, then it is interpreted to contain an integer array index vector as well, with the *ctYTableLengths* resource indicating the length of the vector.

If the *ctYTable* resource is not set, then the *ctXTable* resource must be set.

Default: NULL

ctYTableLengths

This resource is a one-dimensional integer array. Each element of the array indicates the length of the corresponding vector in the *ctYTable* resource. This resource must be set if the *ctYTable* resource is set.

Default: NULL

ctYTableType

This resource is used to indicate the type of each element of each vector in the *ctYTable* resource. It is a string value of the type's name. For example, "Integer". There are symbols provided to the C interface for all the available types. "Integer" would be set with *NhlTInteger*.

Default: NULL
This resource must be set if the *ctYTable* resource is set.

ctYElementSize

This resource is used to indicate the size of each element of each vector in the *ctYTable* resource. If the *ctYTable* resource is set, and this one is not, then the **CoordArrTable** object will use the *ctYTableType* resource to try to determine the size of each element; therefore this resource is not usually necessary.

Default: <dynamic>
Determined from *ctYTableType* if possible.

ctYMissingV

This resource indicates a **missing** value for elements in the *ctYTable* resource. When the HLU library is parsing the data in the *ctYTable* elements, it will treat any element with this value as **missing** data. This resource has a *dynamic* type so that elements of any type can be set to it. It is important to realize that the elements in *ctYTable* will be converted to float, and the contents of this resource will be converted to float, before the elements are compared with the value of this resource.

Default: None

ctYMaxV

This resource is used to tell the **CoordArrTable** object the maximum value contained in the *ctYTable* resource. If it is not specified, the **CoordArrTable** object will compute it. This resource has a *dynamic* type so that elements of any type can be set to it.

Default: <*dynamic*>
The value will be computed if it is not set.

ctYMinV

This resource is used to tell the **CoordArrTable** object the minimum value contained in the *ctYTable* resource. If it is not specified, the **CoordArrTable** object will compute it. This resource has a *dynamic* type so that elements of any type can be set to it.

Default: <*dynamic*>

The value will be computed if it is not set.

DataComm class resource descriptions

dcDelayCompute

This resource is used to indicate if the **DataComm** object should buffer its response to changes in **DataItem** objects that are associated with data resources in the **DataComm** object. If this resource is False, then the **DataComm** object will immediately update its internal state to changes the user makes to an associated **DataItem**. If this resource is True, then the **DataComm** object will wait until the user tries to draw the **DataComm** object, or they call the **UpdateData** function.

It may be useful to set this resource True, if a particular data resource has a number of **DataItem**s associated with it and each one of the **DataItem**s are modified. In this case, if the *dcDelayCompute* resource is True, then the **DataComm** object will only recompute data extents once, instead of recomputing them every single time each of the associated **DataItem** objects is modified.

Default: False

Error class resource descriptions

errBuffer

This resource determines if error messages should be buffered so they can be extracted by the application programmer at a later time.

Default: False

errLevel

Specifies the error level that should be reported. Any error message that is less severe than this level will not be reported or buffered.

Default: WARNING

errPrint

Used to determine if error messages should be printed or not. If they are, they are printed to the file indicated by the *errFileName*, *errFilePtr* and *errUnitNumber* resources.

Default: True

errFileName

Specifies the file to print error messages to. The error object understands the file names "stderr" and "stdout" to be the streams associated with those standard *FILE* pointers in the UNIX environment. If the **Error** object is in Fortran mode, it will still interpret the "stderr" and "stdout" file names. If the *errUnitNumber* resource is not set, then the library will use the appropriate unit numbers for these standard UNIX streams. The error object will try to create a file of any other name relative to the current directory or append to that file if it already exists.

Default: "stderr"
Note: If the **Error** object is in C mode, and the *errFilePtr* resource is set, then this resource is ignored. Also, if the **Error** object is in Fortran mode, and the *errUnitNumber* is set to an Open unit, this resource is ignored.

errFilePtr

This resource is only used if the HLU library is initialized with one of the C functions **NhlInitialize** or **NhlOpen**. This resource specifies the C file pointer to print the error messages to, if the *errPrint* resource is True. If this resource is set, then the *errFileName* resource is ignored.

Default: NULL

errUnitNumber

This resource is only used if the HLU library is initialized with one of the Fortran routines **NHLFINITIALIZE** or **NHLFOPEN**. This resource specifies the Fortran unit number to print the error messages to if the *errPrint* resource is True. If the unit number has already been opened, then the **Error** object ignores the *errFileName* resource. If the unit number has not been opened yet, then the **Error** object attempts to open the file. If the *errFileName* resource has been set, then it uses that name, otherwise it doesn't. If this resource is not set, then the **Error** object just uses the *errFileName* resource to determine where it should print error messages.

Default: 74
Note: If the *errFileName* resource is set to either "stdout" or "stderr", then this defaults to the appropriate unit numbers for those standard UNIX streams, usually 6 and 0 respectively.

GraphicStyle class resource descriptions

gsClipOn

NOT YET IMPLEMENTED

Default: True

gsLineDashPattern

This resource of type NhlTDashIndex specifies the dash pattern for line primitives associated with the **GraphicStyle** object.

Default: 0

gsLineDashSegLenF

This resource specifies the length of each segment of the dash pattern used for line primitives associated with the **GraphicStyle** object.

Default: 0.15

gsLineColor

This resource of type NhlTColorIndex specifies the color of line primitives associated with the **GraphicStyle** object.

Default: `Foreground`

gsLineThicknessF

This resource specifies the thickness of line primitives associated with the **GraphicStyle** object. The value acts as a multiplier of a (device-dependent) unit thickness.

Default: 1.0

gsLineLabelString

This resource specifies the string used as the line label for line primitives associated with the **GraphicStyle** object.

Default: <dynamic>

gsLineLabelFont

This resource of type NhlTFont specifies the line label font for line primitives associated with the **GraphicStyle** object.

Default: 0

gsLineLabelFontColor

This resource of type NhlTColorIndex specifies the line label color for line primitives associated with the **GraphicStyle** object.

Default: `Foreground`

gsLineLabelFontHeightF

This resource specifies the line label font height for line primitives associated with the **GraphicStyle** object.

Default: 0.0125

gsLineLabelFontAspectF

This resource specifies the line label font aspect ratio for line primitives associated with the **GraphicStyle** object.

Default: 1.3125

gsLineLabelFontThicknessF

This resource specifies the line label font thickness for line primitives associated with the **GraphicStyle** object. The value acts as a multiplier of a (device-dependent) unit thickness.

Default: 1.0

gsLineLabelFontQuality

This resource of type NhlTFontQuality specifies the quality of the line label font for line primitives associated with the **GraphicStyle** object.

Default: `High`

gsLineLabelConstantSpacingF

Normally when *gsLineLabelFontQuality* is set to `High`, line labels for line primitives associated with the **GraphicStyle** object are rendered using proportional spacing. Setting the *gsLineLabelConstantSpacingF* to a value greater than 0.0 overrides this behavior. Instead, the distance from the start of one character to the next is computed by multiplying a single standard character width by the value of *gsLineLabelConstantSpacingF*. When *gsLineLabelConstantSpacingF* has a value between 0.0 and 1.0, characters will overlap. A value of 1.0 implies, on average, no space between characters, while values increasing from 1.0 cause the space between characters to grow. This parameter is ignored when *gsLineLabelFontQuality* is not `High`. Values less than 0.0 result in an error and are replaced with the default value.

Default: 0.0

gsLineLabelFuncCode

This resource of type NhlTCharacter sets the line label function code used for line primitives associated with the **GraphicStyle** object.

Default: **:**

gsFillIndex

This resource of type NhlTFillIndex sets the fill pattern used for fill primitives associated with the **GraphicStyle** object.

Default: 0

gsFillColor

This resource of type NhlTColorIndex sets the color used for fill primitives associated with the **GraphicStyle** object.

Default: `Foreground`

gsFillBackgroundColor

This resource of type NhlTColorIndex sets the color used as the background of fill primitives associated with the **GraphicStyle** object.

Default: `Transparent`

gsFillScaleF

This resource specifies the fill pattern scaling factor for fill primitives associated with the **GraphicStyle** object.

Default: 1.0

gsFillLineThicknessF

This resource specifies the thickness of lines used within the fill patterns of fill primitives associated with the **GraphicStyle** object.

Default: 1.0

gsEdgesOn

If this boolean resource is set True, fill primitives associated with the **GraphicStyle** object are rendered with a line drawn around all visible edges.

Default: False

gsEdgeDashPattern

This resource of type NhlTDashIndex specifies the dash pattern for the edges of fill primitives associated with the **GraphicStyle** object.

Default: 0

gsEdgeThicknessF

This resource specifies the thickness of the edges of fill primitives associated with the **GraphicStyle** object. The value acts as a multiplier of a (device-dependent) unit thickness.

Default: 1.0

gsEdgeDashSegLenF

This resource specifies the length of each segment of the dash pattern used for the edges of fill primitives associated with the **GraphicStyle** object.

Default: .15

gsEdgeColor

This resource of type NhlTColorIndex specifies the color used for the edges of fill primitives associated with the **GraphicStyle** object.

Default: `Foreground`

gsMarkerIndex

This resource of type NhlTMarkerIndex specifies the marker glyph for marker primitives associated with the **GraphicStyle** object.

Default: "asterisk"

gsMarkerColor

This resource of type NhlTColorIndex specifies the color of marker primitives associated with the **GraphicStyle** object.

Default: `Foreground`

gsMarkerSizeF

This resource specifies the NDC size of marker primitives associated with the **GraphicStyle** object.

Default: 0.007

gsMarkerThicknessF

This resource specifies the thickness of the lines used to render marker primitives associated with the **GraphicStyle** object. The value acts as a multiplier of a (device-dependent) unit thickness.

Default: 1.0

gsTextAngleF

This resource specifies the rotation angle around the justification point of text primitives associated with the **GraphicStyle** object.

Default: 0.0

gsFont

This resource specifies the font used to render text primitives associated with the **GraphicStyle** object.

Default: `pwritx`

gsTextJustification

This resource of type NhlJustification sets the justification point for text primitives associated with the **GraphicStyle** object.

Default: `CenterCenter`

gsFontQuality

This resource of type NhlFontQuality determines the quality of the font used to render text primitives associated with the **GraphicStyle** object. There are three choices:

`High`
: Draw characters using any of the stoked or filled fonts.

`Medium`
: Draw characters using a 94-character stroked font that is simpler than the `High` quality fonts, resulting in somewhat smaller metafiles. The *gsFont* resource is ignored.

`Low`
: The characters are output as a string into the metafile. The quality of the output therefore depends on the fonts available to the metafile translator. When NCAR Graphics translators are used, the font quality is similar to that of `Medium` text. The *gsFont* resource is ignored.

Default: `High`

gsFontColor

This resource of type NhlTColorIndex specifies the color of text primitives associated with the **GraphicStyle** object.

Default: `Foreground`

gsFontHeightF

This resource specifies the font height for text primitives associated with the **GraphicStyle** object.

Default: 0.015

gsFontAspectF

This resource specifies the font aspect ratio for text primitives associated with the **GraphicStyle** object.

Default: 1.3125

gsFontThicknessF

This resource specifies the font thickness for text primitives associated with the **GraphicStyle** object. The value acts as a multiplier of a (device-dependent) unit thickness.

Default: 1.0

gsTextConstantSpacingF

Normally when *gsFontQuality* is set to `High`, text primitives associated with the **GraphicStyle** object are rendered using proportional spacing. Setting the *gsTextConstantSpacingF* to a value greater than 0.0 overrides this behavior. Instead, the distance from the start of one character to the next is computed by multiplying a single standard character width by the value of *gsTextConstantSpacingF*. When *gsTextConstantSpacingF* has a value between 0.0 and 1.0, characters will overlap. A value of 1.0 implies, on average, no space between characters, while values increasing from 1.0 cause the space between characters to grow. This parameter is ignored when *gsFontQuality* is not `High`. Values less than 0.0 result in an error and are replaced with the default value.

Default: 0.0

gsTextDirection

This resource of type NhlTextDirection specifies the direction of the text primitives associated with the **GraphicStyle** object. There are two choices:

`Down`

Each character is placed below the previous character in the text string.

`Across`

Each character is placed to the right of the previous character in the text string.

These descriptions apply before any rotation due to *gsTextAngleF* is applied to the text primitive.

Default: `Across`

gsTextFuncCode

This resource of type NhlTCharacter specifies the function code used for text primitives associated with the **GraphicStyle** object.

Default: **:**

LabelBar class resource descriptions

lbLabelBarOn

A boolean flag that determines whether the **LabelBar** should appear. Primarily useful as a forwarded resource when the **LabelBar** is a child of a higher level object.

This resource may be intercepted or disabled by:

- **PlotManager**

Default: True

lbOrientation

This resource of type NhlTOrientation specifies whether the labelbar boxes are arranged horizontally in a row or vertically in a column. The *major axis* of the **LabelBar** instance is parallel to the orientation and the *minor axis* is perpendicular to the orientation.

This resource may be intercepted or disabled by:

- **PlotManager**

Default: `Vertical`

lbJustification

When the labelbar changes size, the justification determines a fixed point about which the size change occurs. Any of the corners, the center of any edge, or the current center of the **LabelBar** may be set to the fixed justification point.

This resource may be intercepted or disabled by:

- **PlotManager**

Default: `BottomLeft`

lbBoxMajorExtentF

Determines the amount of the area allotted to each box of the **LabelBar** in the direction of *lbOrientation* is actually occupied by the box. When set to 1.0, the boxes touch each other. If set to 0.0, the boxes disappear entirely. Intermediate values create separated boxes.

Default: 1.0

lbBoxMinorExtentF

When the *lbAutoManage* feature is turned on, this resource determines the fraction of the distance (less the margins) across the axis perpendicular to the orientation (the minor axis) occupied by the boxes of the **LabelBar**. If set to 1.0, the boxes entirely crowd out their associated labels. If *lbTitlePosition* is set to a side parallel with the major axis, the *lbBoxMinorExtentF* cannot exceed 1.0 minus the amount of space used for the title, as set by the resource *lbTitleExtentF*.

When *lbAutoManage* is False and *lbTitlePosition* is set to a side perpendicular to the major axis, the axis extent from which the box minor extent is calculated includes any extra extent added due to an increased value given to *lbTitleFontHeightF*. However, it does not include extra extent due to increased value given to the *lbLabelFontHeightF* resource.

Default: 0.33

lbBoxCount

Number of boxes in the labelbar. All the **LabelBar** array resources, when specified, are required to have a number of elements related to the number of boxes. The arrays specified by *lbFillPatterns*, *lbFillColors*, and *lbFillScales* must have at least as many elements as the box count. The minimum size of the *lbLabelStrings* array may be the box count, one element less than box count, or one element more than box count, depending on the setting of the *lbLabelAlignment* resource. The *lbBoxFractions* array, when set, always requires one element more than box count.

This resource may be intercepted or disabled by:

- **ContourPlot**
- **VectorPlot**

Default: 16

lbBoxSizing

When set to UniformSizing, all the boxes in the **LabelBar** have the same size. When set to ExplicitSizing, the values in the array, *lbBoxFractions*, determine the relative size of each box along the major axis (the axis of orientation).

Default: `UniformSizing`

lbAutoManage

The *lbAutoManage* switch determines how **LabelBar** operates; when True, **LabelBar** manages the sizing of the title and the label text. The title is always sized to fit within the currently set boundaries of the **LabelBar** given any text angle, aspect ratio, etc. The labels also are sized to fit within the current boundary. Additionally, the sizing of the labels is managed so that under any rotation, the labels will not overlap. Also the label justification is managed such that, given any rotation, the end of the label string aligns with the correct **LabelBar** box. When off, you may directly size the labels and text as you please. However, under rotation, the justification of the labels does not change, and, although the text is moved out of the way of the **LabelBar** boxes, it will not necessarily line up correctly. In practice, when working interactively, a good method is to create a basic **LabelBar** layout close to the desired size with the *lbAutoManage* mode on, then switch it off to tune the text size precisely to your taste.

Currently, when the text of the labels is rotated, the size of the **LabelBar** may increase slightly along the axis of orientation.

Default: True

lbLabelOffsetF

Defines an offset, specified as a fraction of the length of the minor labelbar axis (perpendicular to the axis of orientation), between the **LabelBar** boxes and the labels.

Default: 0.1

lbTitleOffsetF

This resource defines an offset specified as a fraction of the length of the axis (minus the margins) perpendicular to the side specified by *lbTitlePosition*. This offset separates the title extent, as specified by *lgTitleExtentF*, from the other elements of the *LabelBar*.

Default: 0.03

lbLeftMarginF

Defines an offset, specified as a fraction of whichever **LabelBar** axis is smallest, between the leftmost **LabelBar** element and the left edge of the **LabelBar** perimeter. It is always subtracted from the current **LabelBar** extent.

Default: 0.05

lbRightMarginF

Defines an offset, specified as a fraction of whichever **LabelBar** axis is smallest, between the rightmost **LabelBar** element and the right edge of the **LabelBar** perimeter. It is always subtracted from the current **LabelBar** extent.

Default: 0.05

lbBottomMarginF

Defines an offset, specified as a fraction of whichever **LabelBar** axis is smallest, between the rightmost **LabelBar** element and the right edge of the **LabelBar** perimeter. It is always subtracted from the current **LabelBar** extent.

Default: 0.05

lbTopMarginF

Defines an offset, specified as a fraction of whichever **LabelBar** axis is smallest, between the rightmost **LabelBar** element and the right edge of the **LabelBar** perimeter. It is always subtracted from the current **LabelBar** extent.

Default: 0.05

lbMonoFillColor

When set True, all **LabelBar** boxes are set to a single color, as specified by the value of the scalar resource *lbFillColor*. When False, the elements of the array resource *lbFillColors* control the color of each box individually.

This resource may be intercepted or disabled by:

- **ContourPlot**
- **VectorPlot**

Default: False

lbFillColor

When *lbMonoFillColor* is set True, this resource of type NhlTColorIndex sets a uniform fill color for all the **LabelBar** boxes.

This resource may be intercepted or disabled by:

- **ContourPlot**
- **VectorPlot**

Default: `Foreground`

lbFillColors

This array resource of type NhlTColorIndexGenArray individually sets the color of each box in the **LabelBar** when *lbMonoFillColor* is set False. The **LabelBar** ensures that this array contains at least as many elements as the current value of *lbBoxCount*. You may cause a box to appear empty by setting the appropriate array element to the value `Transparent`.

This resource may be intercepted or disabled by:

- **ContourPlot**
- **VectorPlot**

Default: By default, each box is assigned to the next succeeding color in the hlu color table, up to the number of defined colors. Additional boxes are assigned the current value of *wkForegroundColor*.

lbMonoFillPattern

When set True, all the boxes in the labelbar are set to a single pattern, as specified by the value of the scalar resource *lbFillPattern*.

This resource may be intercepted or disabled by:

- **ContourPlot**
- **VectorPlot**

Default: False

lbFillPattern

When *lbMonoFillPattern* is set True, this resource of type NhlTFillIndex sets a uniform fill pattern for all the **LabelBar** boxes.

This resource may be intercepted or disabled by:

- **ContourPlot**
- **VectorPlot**

Default: `SolidFill`

lbFillPatterns

This array resource of type NhlTFillIndexGenArray individually sets the fill pattern of each box in the **LabelBar** when *lbMonoFillPattern* is set False. The **LabelBar** ensures that this array contains at least as many elements as the current value of *lbBoxCount*. You can cause any box to appear empty by setting the appropriate array element to the value `HollowFill` (-1). Note that you can use the scalar resource *lbFillBackground* to set a uniform solid-fill background color the fill patterns.

This resource may be intercepted or disabled by:

- **ContourPlot**
- **VectorPlot**

Default: All array elements above those specified by the user are assigned values according to the formula: element_index MOD *wkFillTableLength* + 1.

lbMonoFillScale

When set True, the patterns applied to each box in the **LabelBar** are scaled by a single factor, as specified by the value the scalar resource *lbFillScaleF*.

This resource may be intercepted or disabled by:

- **ContourPlot**
- **VectorPlot**

Default: True

lbFillScaleF

When *lbMonoFillScale* is set True, *lbFillScaleF* sets a uniform fill scale that applies to all patterns in the **LabelBar** boxes.

This resource may be intercepted or disabled by:

- **ContourPlot**
- **VectorPlot**

Default: 1.0

lbFillScales

When *lbMonoFillScale* is False, each element of this array resource contains an individual scale value that is applied to the pattern assigned to the corresponding box in the **LabelBar**. When the scale value is 1.0, all lines in the currently defined patterns are nominally spaced at about 0.01 NDC units. The scale value is applied as a factor to this spacing.

This resource may be intercepted or disabled by:

- **ContourPlot**
- **VectorPlot**

Default: 1.0 for all elements

lbLabelStrings

An array containing the strings comprising each label in the **LabelBar**.

This resource may be intercepted or disabled by:

- **ContourPlot**
- **VectorPlot**

Default: Label_<label element number>

lbBoxFractions

An array that specifies sizing of each box in the **LabelBar** when the box sizing mode is set to `ExplicitSizing`. There must be one more element in this array than the number of items specified by the resource *lgBoxCount*. Each element of the array must eventually contain a number in the range 0.0 to 1.0, with succeeding elements increasing monotonically. The first element must be 0.0 and the last 1.0. If invalid values are discovered when the array is checked, it is not considered an error. Instead, the code simply supplies linearly interpolated values for all adjacent elements containing out- of-bounds elements. The interpolation is performed relative to the two closest bounding elements containing valid values, or 0.0 or 1.0 respectively if the first or last element contains invalid data. The values thus obtained represent the beginnings and endings of the **LabelBar** boxes.

Default: NULL

lbLabelsOn

A boolean flag determining whether labels should appear in the **LabelBar**.

Default: True

lbLabelPosition

This resource of type NhlTPosition determines the position of the labels with respect to the **LabelBar** boxes. If the orientation of the **LabelBar** is `Horizontal`, valid values are `Top`, `Center`, and `Bottom`. If the orientation is `Vertical`, valid values are `Left`, `Center`, and `Right`. If a value inappropriate for the orientation is assigned, the value is silently converted as follows: `Bottom` becomes `Left`, `Top` becomes `Right`, and vice versa. When set to `Center` the labels are centered on, and when the auto-manage feature is on, sized to fit within, each respective label box.

Default: `Right`

lbLabelAngleF

The angle of the text of the labels. When the auto-manage resource is turned on, both the size and justification mode of the label text may change in response to changes of the label angle.

Default: 0.0

lbLabelAlignment

How the labels align with respect to the label boxes. If set to `BoxCenters`, the labels align with the centers of each box, and the number of labels is equal to the number of boxes. If set to `InteriorEdges`, the labels align with the internal separators between the boxes, and there is one fewer label than the number of boxes. If set to `ExternalEdges`, the labels align with the external edges as well as the interior separators between the boxes, and there is one more label than boxes.

This resource may be intercepted or disabled by:

- **ContourPlot**
- **VectorPlot**

Default: `BoxCenters`

lbLabelDirection

This resource of type NhlTTextDirection specifies the direction of the label text.

Default: `Across`

lbLabelJust

The justification of the label text. When the auto-manage feature is on, the justification may be changed internally in response to changes in the label angle. Therefore in order to control the label justification explicitly, you must first turn off the auto-manage feature.

Default: `CenterCenter`

lbLabelFont

This resource of type NhlTFont specifies the font used to render the **LabelBar** labels.

Default: "pwritx"

lbLabelFontColor

The hlu color index used for drawing the label text.

Default: `Foreground`

lbLabelFontHeightF

The height in NDC coordinates of the text used to draw the labels. When *lbAutoManage* is set True, the user cannot directly set the label font height. Rather, it is set in response to other factors, such as the current size and shape of the **LabelBar**, the current setting of *lbBoxMinorExtent*, the current text angle of the labels, and how much space there is between the labels. Set *lbAutoManage* False if you wish to control the label font height directly.

Default: 0.02

lbLabelFontAspectF

Determines the shape of the label font text. Values greater than 1.0 make the text tall and skinny. Values less than one make the text short and wide.

Default: 1.0

lbLabelFontThicknessF

Sets the thickness of the line used to draw the Label text. The value acts as a multiplier of a (device-dependent) unit thickness. This resource is ignored when the *lbLabelFont* specifies a filled font (font indexes 21-22, 25-26, 29-30, and 33-37).

Default: 1.0

lbLabelFontQuality

Determines the text quality used to draw the label text.

Default: `High`

lbLabelConstantSpacingF

Normally when *lbLabelFontQuality* is set to `High`, the**LabelBar** writes line label text with proportional spacing. Setting the *lbLabelConstantSpacingF* to a value greater than 0.0 overrides this behavior and instead begins each character a distance of *lbLabelConstantSpacingF* times the nominal character size from the beginning of the previous character. This implies that values between 0.0 and 1.0 will cause the characters to overlap each other, while a value of 1.0 implies no space between two nominally sized characters. This parameter is ignored when *lbLabelFontQuality* is not `Low` or `Medium`. Values less than 0.0 result in an error and are replaced with the default value.

Default: 0.0

lbLabelFuncCode

Determines the function code character used when parsing the label string.

This resource may be intercepted or disabled by:

- **ContourPlot**

- **VectorPlot**

Default: **:**

lbLabelStride

Determines which labels actually are rendered the **LabelBar** is drawn. For example, if the stride is set to 2, only every other label will be drawn, starting with the first label.

Default: 1

lbTitleString

A string containing the text used for the **LabelBar** title. If *lbTitleString* is set when the LabelBar object created, the boolean resource *lbTitleOn* defaults to True, causing the title to appear. Otherwise it defaults to False. If you explicitly set *lbTitleOn* True without setting *lbTitleString*, **LabelBar** supplies a title consisting of the name of the current instantiation of the object.

Default: <dynamic>

lbTitleOn

A boolean flag determining whether the title should appear in the **LabelBar**. If *lbTitleString* is set when the LabelBar object created, *lbTitleOn* defaults to True. Otherwise it defaults to False.

Default: True

lbTitlePosition

This resource of type NhlTPosition determines the position of the title with respect to the other elements of the **LabelBar**. Valid positions are `Top`, `Bottom`, `Left`, and `Right`. When you set the title position, **LabelBar** automatically adjusts the title direction, unless you explicitly set *lbTitleDirection*in the same call. When you set the position to `Top` or `Bottom`, the title direction is set to `Across`; when the position is set to `Left` or `Right`, the title direction is set to `Down`.

Default: `Top`

lbTitleExtentF

The **LabelBar** title occupies a rectangular portion of the **LabelBar** viewport bounded on three sides by edges of the viewport and on the fourth by a line determined by the value of this resource. *lbTitleExtentF* specifies a fraction of the length (minus the margins) of the **LabelBar** axis perpendicular to *lbTitleSide*. At this point along the length of the axis the fourth side of the title extent rectangle is constructed parallel to the side specified by *lbTitleSide*. The sum of the values given to *lbTitleExtentF* and *lbTitleOffsetF* cannot exceed 0.5 (half the length of the axis). If the sum does exceed 0.5, a warning is issued and both values are reset to their default values.

If *lbAutoManage* is set False, and *lbTitleFontHeightF* is set such that the title extent rectangle cannot accommodate the full extent of the title text, the viewport of the **LabelBar** instance is expanded to fit the title text extent. However, the **LabelBar** treats this additional extent as 'extra'. The title extent rectangle does *not* change its size as long as the **LabelBar** view width or height is not explicitly modified. This means that as you set *lbTitleFontHeightF* to smaller values, the **LabelBar** viewport will shrink until its size matches the size it would have had if the text extent fit within the originally set title extent.

Default: 0.15

lbTitleAngleF

The angle of the title text. When the auto-manage feature is on, the title size changes as the text rotates.

Default: 0.0

lbTitleDirection

This resource of type NhlTTextDirection specifies the direction of the title text. When the title position, as set by the resource *lbTitlePosition*, is `Top` or `Bottom` the direction is set by default to `Across`. When title position is `Left` or `Right` the text is set by default to `Down`.

Default: `Across`

lbTitleFont

This resource of type NhlTFont specifies the font used to render the **LabelBar** title.

Default: "pwritx"

lbTitleFontColor

The hlu index of the color used for the title text.

Default: `Foreground`

lbTitleJust

The justification used for the title text.

Default: `CenterCenter`

lbTitleFontHeightF

The font height in NDC units used for the title text. If *lbAutoManage* is set True, the **LabelBar** sets this resource automatically based on the space available and the value of other title font attributes including *lbTitleAngleF*, *lbTitleConstantSpacingF* and *lbTitleFontAspectF*. The available space is determined from the size of the **LabelBar** viewport and the setting of the resource *lbTitleExtentF*. When *lbAutoManage* is True, attempts by the user to set this resource are simply ignored.

If *lbAutoManage* is False, the **LabelBar** instance will honor the set value of *lbTitleFontHeightF*, even if it must increase the size of the viewport in order to encompass the full extent of the title text. However, space added in this manner is considered an addition to the 'fundamental' size of the **LabelBar**. If the *lbTitleFontHeightF* is reduced to a value less than or equal to the value that would be used if *lbAutoManage* were True, then the **LabelBar** will resize itself to its 'fundamental' size. If you resize the **LabelBar** by setting the width or height of its viewport, *lbTitleFontHeightF* and the 'fundamental' size both adjust themselves proportionally.

Default: 0.025

lbTitleFontAspectF

Determines the shape of the title font text. Values greater than 1.0 make the text tall and skinny. Values less than one make the text short and wide.

Default: 1.0

lbTitleFontThicknessF

Determines the thickness of the line used to draw the Label text. This resource only affects the Hershey fonts.

Default: 1.0

lbTitleFontQuality

Determines the text quality used to draw the title text.

Default: `High`

lbTitleConstantSpacingF

Determines a constant amount of extra space that is placed between each character of the title text. Values less than 0.0 result in an error and are replaced with the default value.

Default: 0.0

lbTitleFuncCode

Determines the function code character used when parsing the label string.

Default: **:**

lbBoxLinesOn

A boolean flag determining whether lines should appear around the boxes in the **LabelBar**.

Default: True

lbBoxLineColor

The hlu index of the color used to draw lines around the boxes in the **LabelBar**

Default: `Foreground`

lbBoxLineThicknessF

Determines the thickness of the lines used around the boxes.

Default: 1.0

lbBoxLineDashPattern

The hlu index of the dash pattern used for the lines around the boxes of the **LabelBar**.

Default: 0

lbBoxLineDashSegLenF

The length in NDC units of the dash pattern used for the lines around the boxes of the **LabelBar**.

Default: 0.15

lbPerimOn

A boolean flag determining whether a line is drawn around the perimeter of the **LabelBar**.

Default: True

lbPerimColor

The hlu index of the color used for the line around the perimeter of **LabelBar**.

Default: `Foreground`

lbPerimFill

The hlu index of the pattern used to fill the background of the **LabelBar** area. Only has an effect when the *lbPerimFillColor* has set to a value greater than `Transparent` (-1).

Default: `HollowFill`

lbPerimFillColor

The hlu index of the color used to fill the background of the **Legend** area. Only has an effect when the *lgPerimFill* has a value greater than `HollowFill` (-1).

Default: `Background`

lbPerimThicknessF

Specifies the thickness of the line used to draw the perimeter of the **LabelBar**.

Default: 1.0

lbPerimDashPattern

Specifies the hlu index of the dash pattern used to draw the perimeter of the **LabelBar**.

Default: 0, specifying a solid line

lbPerimDashSegLenF

The length in NDC units of the dash pattern used to draw the perimeter of the **LabelBar**.

Default: 0.15

lbFillBackground

The color index used for the background of all the boxes in the **LabelBar**. By default it is set to `Transparent` (-1), specifying that the background of the boxes is transparent to whatever it overlays. Note that the box background is only observable when the fill pattern is not solid. This resource also applies to the background of the fill pattern set with the *lbPerimFill* resource.

Default: `Transparent`

lbFillLineThicknessF

The line thickness used for the lines that comprise the fill pattern within the label boxes.

Default: 1.0

Legend class resource descriptions

lgLegendOn

A boolean flag that determines whether the **Legend** should appear.

This resource may be intercepted or disabled by:

- **PlotManager**

Default: True

lgOrientation

This resource of type NhlTOrientation specifies whether the legend items are arranged horizontally in a row or vertically in a column. The *major axis* of the **Legend** instance is parallel to the orientation and the *minor axis* is perpendicular to the orientation.

Default: `Vertical`

lgJustification

When the **Legend** changes size the justification determines a fixed point about which the size change occurs. Any of the corners, the center of any edge, or the current center of the **Legend** may be set to be the fixed justification point.

This resource may be intercepted or disabled by:

- **PlotManager**

Default: `BottomLeft`

lgBoxMajorExtentF

Determines the amount of the area allotted to the boxes surrounding each item of the **Legend** in the direction of *lgOrientation* is actually occupied by the box. This will be visible only when the boxes outlining the space occupied by each item are turned on. When set to 1.0 the boxes touch each other. If set to 0.0 the boxes disappear entirely. Intermediate values create discreet separated boxes. The items themselves are not affected by the value given to *lgBoxMajorExtentF*.

Default: 0.5

lgBoxMinorExtentF

When the *lgAutoManage* feature is turned on, this resource determines the fraction of the distance (less the margins) across the axis perpendicular to the orientation (the minor axis) occupied by the boxes surrounding the each item of the **Legend**. If set to 1.0, the explanatory labels are completely eliminated from the **Legend** area. If the title also extends along the minor axis, the *lgBoxMinorExtentF* cannot exceed 1.0 minus the amount of space requested by the title, as set by the resource *lgMaxTitleExtentF*. The value of the *lgBoxMinorExtentF* controls how long a line segment is when the item type is set to `Lines`, but it does not clip or affect the appearance of item type `Markers` in any way.

When *lbAutoManage* is False and *lgTitlePosition* is set to a side perpendicular to the major axis, the axis extent from which the box minor extent is calculated includes any extra extent added due to an increased value given to *lgTitleFontHeightF*. However, it does not include extra extent due to increased value given to the *lgLabelFontHeightF* resource.

Default: 0.6

lgItemCount

Number of items in the **Legend**. All the **Legend** array resources, when specified, are required to have at least as many elements as *lgItemCount*.

This resource may be intercepted or disabled by:

- **ContourPlot**
- **XyPlot**

Default: 16

lgItemPlacement

When set to `UniformPlacement` all **Legend** items are evenly spaced within the Legend. If set to `ExplicitPlacement`, the distance between each item is controlled by the *lgItemPositions* array.

Default: `UniformPlacement`

lgBoxBackground

The color index used for the background of the boxes enclosing the **Legend** items. By default it is set to -1 (`Transparent`), specifying that the background of the boxes is transparent to whatever it overlays.

Default:`Transparent`

lgAutoManage

The *lgAutoManage* switch determines how **Legend** operates. When it is on, **Legend** manages the sizing of the title and the label text. The title is always sized to fit within the currently set boundaries of the **Legend** given any text angle, aspect ratio, etc. The labels also are sized to fit within the current boundary, with the exception that when the text of the labels is rotated, the **Legend** as a whole may grow along the axis of orientation. Additionally the sizing of the labels is managed so that under any rotation, the labels will not overlap. Also the label justification is managed such that, given any rotation the end of the label string aligns with the correct **Legend** item. When off, you may directly size the labels and text as you please. However, under rotation, the justification of the labels does not change, and, although the text is moved out of the way of the **Legend** boxes, it will not necessarily line up correctly. In practice, when working interactively, a good method is to create a basic **Legend** layout close to the desired size with the *lgAutoManage* mode on, then switch it off to tune the text size precisely to your taste.

Default: True

lgLabelOffsetF

Defines an offset, specified as a fraction of the length of the minor **Legend** axis (perpendicular to the axis of orientation), between the **Legend** boxes and the labels. The offset may be positive or negative and the offset direction depends on the current value of *lgLabelJust*. The positive direction is from the justified edge toward the body of the text (i.e. positive for right-justified text is toward the left). The primary use of negative offsets for labels is when the label alignment is set either to `AboveBoxes` or `BelowBoxes`. In these cases, you may begin or end the text at some point under or over the **Legend** items, thus allowing for more text within a narrower **Legend**. If the label alignment is set to `BoxCenters`, negative offsets may result in text appearing overlaid on the actual **Legend** items. When positive the label offset quantity is subtracted from the amount allotted to the labels as a whole.

Default: 0.02

lgTitleOffsetF

This resource defines an offset specified as a fraction of the length of the axis (minus the margins) perpendicular to the side specified by *lgTitlePosition*. This offset separates the title extent, as specified by *lgTitleExtentF*, from the other elements of the *Legend*.

Default: 0.03

lgLeftMarginF

Defines an offset, specified as a fraction of whichever **Legend** axis is smallest, between the leftmost **Legend** element and the left edge of the **Legend** perimeter. It is always subtracted from the current **Legend** extent.

Default: 0.05

lgRightMarginF

Defines an offset, specified as a fraction of whichever **Legend** axis is smallest, between the rightmost **Legend** element and the right edge of the **Legend** perimeter. It is always subtracted from the current **Legend** extent.

Default: 0.05

lgBottomMarginF

Defines an offset, specified as a fraction of whichever **Legend** axis is smallest, between the rightmost **Legend** element and the right edge of the **Legend** perimeter. It is always subtracted from the current **Legend** extent.

Default: 0.05

lgTopMarginF

Defines an offset, specified as a fraction of whichever **Legend** axis is smallest, between the rightmost **Legend** element and the right edge of the **Legend** perimeter. It is always subtracted from the current **Legend** extent.

Default: 0.05

lgMonoItemType

When set True, the scalar resource *lgItemType* sets all items in the legend to the same NhlTMarkLineMode. This indicates if the item should contain a line, a marker, or both. Otherwise, the array resource *lgItemTypes* controls each legend item individually.

This resource may be intercepted or disabled by:

- **ContourPlot**
- **XyPlot**

Default: True

lgItemType

When *lgMonoItemType* is set True, this resource sets all the items in the legend. It indicates if they should contain a line, a marker, or both.

This resource may be intercepted or disabled by:

- **ContourPlot**
- **XyPlot**

Default: `Lines`

lgItemTypes

When *lgMonoItemType* is set False, this array sets an NhlTMarkLineMode value for each **Legend** item individually. This means that each item in the **Legend** can have a line, a marker, or both based on the value of each element of this array.

This resource may be intercepted or disabled by:

- **ContourPlot**
- **XyPlot**

Default: NULL

lgMonoDashIndex

When set True, the scalar resource *lgDashIndex* sets the dash pattern index for all lines drawn in items in the legend to a single index value. Otherwise, the array resource *lgDashIndexes* controls the dash index of each legend item individually.

This resource may be intercepted or disabled by:
- **ContourPlot**
- **XyPlot**

Default: False

lgDashIndex

When *lgMonoDashIndex* is set True, this resource sets a uniform dash pattern index for the lines drawn in all items in the legend.

This resource may be intercepted or disabled by:
- **ContourPlot**
- **XyPlot**

Default: `SolidLine`

lgDashIndexes

When *lgMonoDashIndex* is set False, this array sets a dash pattern index value for each **Legend** item individually. If a positive index greater than the number of dash patterns is specified, the dash pattern routines automatically use modular arithmetic to arrive at an index value within the defined range of indexes.

This resource may be intercepted or disabled by:
- **ContourPlot**
- **XyPlot**

Default: Array elements 0 through the item count - 1 are assigned indexes 1 through item count.

lgMonoMarkerIndex

When set True, the scalar resource *lgMarkerIndex* sets the marker index for all lines drawn in items in the legend to a single index value. Otherwise, the array resource *lgMarkerIndexes* controls the marker index of each legend item individually.

This resource may be intercepted or disabled by:
- **ContourPlot**
- **XyPlot**

Default: False

lgMarkerIndex

When *lgMonoMarkerIndex* is set True, this resource sets a uniform marker for all items in the legend.

This resource may be intercepted or disabled by:
- **ContourPlot**
- **XyPlot**

Default:

lgMarkerIndexes

When *lgMonoMarkerIndex* is set False, this array sets a marker index value for each **Legend** item individually. If a positive index greater than the number of markers is specified, the marker routines automatically use modular arithmetic to arrive at an index value within the defined range of indexes.

This resource may be intercepted or disabled by:

- **ContourPlot**
- **XyPlot**

Default: Array elements 0 through the item count - 1 are assigned indexes 1 through item count.

lgLineLabelStrings

An array of strings. Each string determines the contents of the label used in the rendering of the line specified by the corresponding element in the *lgDashIndexes* resource, if the corresponding element in the *lgItemTypes* resource indicates "Lines". To have an item line without a line label, specify that element with NULL using the C interface, or an empty string ("") from FORTRAN or NCL.

This resource may be intercepted or disabled by:

- **ContourPlot**
- **XyPlot**

Default: NULL

lgMonoLineColor

When set True, all item lines in the **Legend** are set to a single color, as specified by the value of the scalar resource *lgLineColor*. Otherwise, the array resource *lgLineColors* controls the color of each item line individually.

This resource may be intercepted or disabled by:

- **ContourPlot**
- **XyPlot**

Default: False

lgLineColor

When *lgMonoLineColor* is True, this resource of type NhlTColorIndex sets the color of each **Legend** item line to a single value. Note that color of the line labels are controlled by a different set of resources.

This resource may be intercepted or disabled by:

- **ContourPlot**
- **XyPlot**

Default: `Foreground`

lgLineColors

When *lgMonoLineColor* is False, this resource of type NhlTColorIndexGenArray individually sets the color assigned to each **Legend** item line. Note that color of the line labels are controlled by a different set of resources.

This resource may be intercepted or disabled by:

- **ContourPlot**

- **XyPlot**

Default: By default each item line is assigned to the next succeeding color in the hlu colortable, up to the number of defined colors. Additional boxes are assigned the value, "Foreground" (1).

lgMonoMarkerColor

When set True, all items markers in the **Legend** are set to a single color, as specified by the value of the scalar resource *lgMarkerColor*. Otherwise, the array resource *lgMarkerColors* controls the color of each item marker individually.

This resource may be intercepted or disabled by:

- **ContourPlot**
- **XyPlot**

Default: False

lgMarkerColor

When *lgMonoMarkerColor* is True, this resource of type NhlTColorIndex sets the color of each **Legend** item marker to a single value.

This resource may be intercepted or disabled by:

- **ContourPlot**
- **XyPlot**

Default: `Foreground`

lgMarkerColors

When *lgMonoMarkerColor* is False, this resource of type NhlTColorIndexGenArray individually sets the color assigned to each **Legend** item marker.

This resource may be intercepted or disabled by:

- **ContourPlot**
- **XyPlot**

Default: By default each item marker is assigned to the next succeeding color in the hlu colortable, up to the number of defined colors. Additional boxes are assigned the value, "Foreground" (1).

lgMonoLineDashSegLen

When set True, a single line segment length is used for all item lines int the **Legend** object, as specified by the scalar resource *lgLineDashSegLenF*. Otherwise the array element of *lgLineDashSegLens* controls the segment length of each individual **Legend** item line.

This resource may be intercepted or disabled by:

- **ContourPlot**
- **XyPlot**

Default: True

lgLineDashSegLenF

The length in NDC units of the dash pattern used for **Legend** line items. This is the length before the dash pattern repeats.

This resource may be intercepted or disabled by:
- **ContourPlot**
- **XyPlot**

Default: 0.15

lgLineDashSegLens

When *lgMonoLineDashSegLen* is False, this array resource individually controls the segment length value applied to the each **Legend** item line.

This resource may be intercepted or disabled by:
- **ContourPlot**
- **XyPlot**

Default: NULL

lgMonoLineThickness

When set True, a single line thickness is used for all item lines in the **Legend** object, as specified by the scalar resource *lgLineThicknessF*. Otherwise the array element of *lgLineThicknesses* controls the thickness of each individual **Legend** item line.

This resource may be intercepted or disabled by:
- **ContourPlot**
- **XyPlot**

Default: True

lgLineThicknessF

When *lgMonoLineThickness* is True, this resource sets a singles thickness value for all **Legend** item lines. It does not affect the thickness of lines used to draw the line labels in the line items.

This resource may be intercepted or disabled by:
- **ContourPlot**
- **XyPlot**

Default: 1.0

lgLineThicknesses

When *lgMonoLineThickness* is False, this array resource individually controls the thickness value applied to the each **Legend** item line. The thickness of the characters in the line labels are not affected.

This resource may be intercepted or disabled by:
- **ContourPlot**
- **XyPlot**

Default: NULL

lgMonoMarkerThickness

When set True, a single thickness is used for all item markerss in the **Legend** object, as specified by the scalar resource *lgMarkerThicknessF*. Otherwise the array elements of *lgMarkerThicknesses* controls the thickness of each individual **Legend** marker.

This resource may be intercepted or disabled by:

- **ContourPlot**
- **XyPlot**

Default: True

lgMarkerThicknessF

When *lgMonoMarkerThickness* is True, this resource sets a singles thickness value for all **Legend** markers. This value affects the thickness of the lines in the glyphs used to represent the markers.

This resource may be intercepted or disabled by:

- **ContourPlot**
- **XyPlot**

Default:

lgMarkerThicknesses

When *lgMonoMarkerThickness* is False, this array resource individually controls the thickness value applied to the each **Legend** marker. The value affects the thickness of the lines in the glyphs used to represent the markers.

This resource may be intercepted or disabled by:

- **ContourPlot**
- **XyPlot**

Default: NULL

lgMonoLineLabelFontHeight

When set True, the font height of all line labels of line type items are set to a single size, as specified by the scalar resource *lgLineLabelFontHeightF*. Otherwise, the font height is individually controlled by the array resource *lgLineLabelFontHeights*.

This resource may be intercepted or disabled by:

- **ContourPlot**
- **XyPlot**

Default: True

lgLineLabelFontHeightF

When *lgMonoLineLabelFontHeight* is True, this resource sets the height in NDC units of all the internal line labels used for line type items. Note that unlike many other text font height type resources, the values of this resource does *not* automatically scale when the size of the **Legend** is changed. This is because the size may be of significance in distinguishing the items for which the **Legend** serves as the key. Also, note that setting *lgAutoManage* True does not disable *lgLineLabelFontHeightF*. The **Legend** object currently does nothing to prevent the internal line labels from overlapping other **Legend** items, when the value of this resource is large enough.

This resource may be intercepted or disabled by:

- **ContourPlot**
- **XyPlot**

Default: 0.01

lgLineLabelFontHeights

When *lgMonoLineLabelFontHeight* is False, the elements of this array individually specify the height in NDC units of the internal line labels used with line type items. Note that unlike many other text font height type resources, the values of this resource does *not* automatically scale when the size of the **Legend** is changed. This is because the size may be of significance in distinguishing the items for which the **Legend** serves as the key. Also, note that setting *lgAutoManage* True does not disable *lgLineLabelFontHeights*. The **Legend** object currently does nothing to prevent the internal line labels from overlapping other **Legend** items, when the value of this resource is large enough.

This resource may be intercepted or disabled by:

- **ContourPlot**
- **XyPlot**

Default: NULL

lgMonoMarkerSize

When set True, the size of all marker type items is set to a single size, as specified by the scalar resource *lgMarkerSizeF*. Otherwise, the size is individually controlled by the array resource *lgMarkerSizes*.

This resource may be intercepted or disabled by:

- **ContourPlot**
- **XyPlot**

Default: True

lgMarkerSizeF

When *lgMonoMarkerSize* is True, this resource sets the height in NDC units of all marker items.

This resource may be intercepted or disabled by:

- **ContourPlot**
- **XyPlot**

Default: 0.01

lgMarkerSizes

When *lgMonoMarkerSize* is False, the elements of this array individually specify the height in NDC units of marker items.

This resource may be intercepted or disabled by:

- **ContourPlot**
- **XyPlot**

Default: NULL

lgLabelStrings

An array containing the strings that comprise the label associated with each item of the **Legend**.

This resource may be intercepted or disabled by:

- **ContourPlot**

Default: Label_<label element number>

lgItemPositions

When *lgItemPlacement* is set to `ExplicitPlacement`, the elements of this array control the relative position within the **Legend** of each item. The array should be a monotonically increasing sequence in the range 0.0 - 1.0 inclusive (except as noted below). Each element corresponds to the position of the **Legend** item with the corresponding index. The value indicates the distance of each particular item from the left or bottom (depending on the value of *lgOrientation*) of the **Legend** item area. A value of 1.0 indicates the right or top of the item area, while 0.5 would indicate the center of the item area. You can cause a subgroup of the items to be evenly spaced by giving the value -1.0 to the elements between two elements containing valid monotonically increasing values. The **Legend** object will replace the -1.0 values with monotonically increasing values evenly spaced between the two given valid values. Actually it will replace *any* invalid values in the same manner, although values other than -1.0 will cause a WARNING error.

Default: NULL

lgMonoLineLabelFontColor

When set True, all internal line labels have the same color, as specified by the scalar resource *lgLineLabelFontColor*. Otherwise, the colors of the line labels are controlled individually based on the elements of the array resource *lgLineLabelFontColors*.

This resource may be intercepted or disabled by:

- **ContourPlot**
- **XyPlot**

Default: False

lgLineLabelFontColor

When *lgMonoLineLabelFontColor* is True, this resource of type NhlTColorIndex sets the color of all line labels used in line type items to a single value.

This resource may be intercepted or disabled by:

- **ContourPlot**
- **XyPlot**

Default:

lgLineLabelFontColors

When *lgMonoLineLabelFontColor* is False, this resource of type NhlTColorIndexGenArray individually sets the color index used to render the line label for each line type **Legend** item.

This resource may be intercepted or disabled by:

- **ContourPlot**
- **XyPlot**

Default: NULL

lgLineLabelsOn

A boolean flag determining whether to draw line labels in line type **Legend** items.

Default: True

lgLineLabelFont

This resource of type NhlTFont specifies the font used to render the **Legend** line labels.

Default: "pwritx"

lgLineLabelFontAspectF

Determines the shape of the internal line label font text. Values greater than 1.0 make the text tall and skinny. Values less than one make the text short and wide.

This resource may be intercepted or disabled by:

- **ContourPlot**
- **XyPlot**

Default: 1.0

lgLineLabelFontThicknessF

Determines the thickness of the line used to draw the internal line label text. This resource only affects the Hershey fonts.

This resource may be intercepted or disabled by:

- **ContourPlot**
- **XyPlot**

Default: 1.0

lgLineLabelFontQuality

Determines the text quality used to draw the line label text.

This resource may be intercepted or disabled by:

- **ContourPlot**
- **XyPlot**

Default: `High`

lgLineLabelConstantSpacingF

Determines a constant amount of extra space that is placed between each character of the line label text. Values less than 0.0 result in an error and are replaced with the default value.

This resource may be intercepted or disabled by:

- **ContourPlot**
- **XyPlot**

Default: 0.0

lgLineLabelFuncCode

Determines the function code character used when parsing the line label string.

This resource may be intercepted or disabled by:

- **ContourPlot**
- **XyPlot**

Default: **:**

lgLabelsOn

A boolean flag determining whether labels should appear in the **Legend**.

Default: True

lgLabelPosition

This resource of type NhlTPosition determines the position of the labels with respect to the **Legend** boxes. If the orientation of the **Legend** is `Horizontal`, valid values are `Top`, `Center`, and `Bottom`. If the orientation is `Vertical`, valid values are `Left`, `Center`, and `Right`. If a value inappropriate for the orientation is assigned, the value is silently converted as follows: `Bottom` becomes `Left`, `Top` becomes `Right`, and vice versa. When set to `Center` the labels are centered on, and when the auto-manage feature is on, sized to fit in the same space as each **Legend** item. To avoid overlap, the *lgLabelAlignment* resource must be set properly when the label position is `Center`.

Default: `Right`

lgLabelAngleF

The angle of the text of the labels. When the auto-manage resource is turned on, both the size and justification mode of the label text may change in response to changes of the label angle.

Default: 0.0

lgLabelAlignment

How the labels align with respect their corresponding items along the minor axis of the **Legend** object. When set to `ItemCenters`, the labels align with the centers of each item. If set to `AboveItems`, the labels appear above the item, while if set to `BelowItems`, the labels appear below the item. To avoid overlap use alignment modes of above or below when the label offset is negative or the label position is set to `Center`.

Default: `ItemCenters`

lgLabelDirection

This resource of type NhlTTextDirection specifies the direction of the label text.

Default: `Across`

lgLabelJust

The justification of the label text. When the auto-manage feature is on, the **Legend** object manages the justification internally. Therefore in order to control the label justification explicitly you must first turn off the auto-manage feature.

Default: `CenterCenter`

lgLabelFont

This resource of type NhlTFont specifies the font used to render the **Legend** labels.

Default: "pwritx"

lgLabelFontColor

The hlu color index used for drawing the label text.

Default: `Foreground`

lgLabelFontHeightF

The height in NDC coordinates of the text used to draw the labels. When *lgAutoManage* is set True, the user cannot directly set the label font height. Rather it is set in response to other factors, such as the current size and shape of the **Legend**, the current setting of *lgBoxMinorExtent*, the current text angle of the labels, and how much space there is between the labels. Set *lgAutoManage* False if you wish to control the label font height directly.

Default: 0.02

lgLabelFontAspectF

Determines the shape of the label font text. Values greater than 1.0 make the text tall and skinny. Values less than one make the text short and wide.

Default: 1.0

lgLabelFontThicknessF

Determines the thickness of the line used to draw the Label text. This resource only affects the Hershey fonts.

Default: 1.0

lgLabelFontQuality

Determines the text quality used to draw the label text.

Default: `High`

lgLabelConstantSpacingF

Determines a constant amount of extra space that is placed between each character of the label text. Values less than 0.0 result in an error and are replaced with the default value.

Default: 0.0

lgLabelFuncCode

Determines the function code character used when parsing the label string.

This resource may be intercepted or disabled by:

- **ContourPlot**

Default: **:**

lgLabelStride

Determines which labels actually are rendered the **Legend** is drawn. For example, if the stride is set to 2, only every other label will be drawn, starting with the first label.

Default: 1

lgTitleString

A string containing the text used for the **Legend** title.

Default: <dynamic>

The name of the **Legend** object is used if this resource is not set.

lgTitleOn

A boolean flag determining whether the title should appear in the **Legend**.

Default: True

lgTitlePosition

This resource of type NhlTPosition determines the position of the title with respect to the other elements of the **Legend**. Valid positions are `Top`, `Bottom`, `Left`, and `Right`. When you set the title position, **Legend** automatically adjusts the title direction, unless you explicitly set *lgTitleDirection*in the same call. When you set the position to `Top` or `Bottom`, the title direction is set to `Across`; when the position is set to `Left` or `Right`, the title direction is set to `Down`.

Default: `Top`

lgTitleExtentF

The **Legend** title occupies a rectangular portion of the **Legend** viewport bounded on three sides by edges of the viewport and on the fourth by a line determined by the value of this resource. *lgTitleExtentF* specifies a fraction of the length (minus the margins) of the **Legend** axis perpendicular to *lgTitleSide*. At this point along the length of the axis the fourth side of the title extent rectangle is constructed parallel to the side specified by *lgTitleSide*. The sum of the values given to *lgTitleExtentF* and *lgTitleOffsetF* cannot exceed 0.5 (half the length of the axis). If the sum does exceed 0.5, a warning is issued and both values are reset to their default values.

If *lgAutoManage* is set False, and *lgTitleFontHeightF* is set such that the title extent rectangle cannot accommodate the full extent of the title text, the viewport of the **Legend** instance is expanded to fit the title text extent. However, the **Legend** treats this additional extent as 'extra'. The title extent rectangle does *not* change its size as long as the **Legend** view width or height is not explicitly modified. This means that as you set *lgTitleFontHeightF* to smaller values, the **Legend** viewport will shrink until its size matches the size it would have had if the text extent fit within the originally set title extent.

Default: 0.15

lgTitleAngleF

The angle of the title text. When the auto-manage feature is on, the title size changes as the text rotates.

Default: 0.0

lgTitleDirection

This resource of type NhlTTextDirection specifies the direction of the title text. When the title position, as set by the resource *lgTitlePosition*, is `Top` or `Bottom` the direction is set by default to `Across`. When title position is `Left` or `Right` the text is set by default to `Down`.

Default: `Across`

lgTitleFont

This resource of type NhlTFont specifies the font used to render the **Legend** title.

Default: "pwritx"

lgTitleFontColor

The hlu index of the color used for the title text.

Default: `Foreground`

lgTitleJust

The justification used for the title text.

Default: `CenterCenter`

lgTitleFontHeightF

The font height in NDC units used for the title text. If *lgAutoManage* is set True, the **Legend** sets this resource automatically based on the space available and the value of other title font attributes including *lgTitleAngleF*, *lgTitleConstantSpacingF* and *lgTitleFontAspectF*. The available space is determined from the size of the **Legend** viewport and the setting of the resource *lgTitleExtentF*. When *lgAutoManage* is True, attempts by the user to set this resource are simply ignored.

If *lgAutoManage* is False, the **Legend** instance will honor the set value of *lgTitleFontHeightF*, even if it must increase the size of the viewport in order to encompass the full extent of the title text. However, space added in this manner is considered an addition to the 'fundamental' size of the **Legend**. If the *lgTitleFontHeightF* is reduced to a value less than or equal to the value that would be used if *lgAutoManage* were True, then the **Legend** will resize itself to its 'fundamental' size. If you resize the **Legend** by setting the width or height of its viewport, *lgTitleFontHeightF* and the 'fundamental' size both adjust themselves proportionally.

Default: 0.025

lgTitleFontAspectF

Determines the shape of the title font text. Values greater than 1.0 make the text tall and skinny. Values less than one make the text short and wide.

Default: 1.0

lgTitleFontThicknessF

Determines the thickness of the line used to draw the Label text. This resource only affects the Hershey fonts.

Default: 1.0

lgTitleFontQuality

Determines the text quality used to draw the title text.

Default: `High`

lgTitleConstantSpacingF

Determines a constant amount of extra space that is placed between each character of the title text. Values less than 0.0 result in an error and are replaced with the default value.

Default: 0.0

lgTitleFuncCode

Determines the function code character when parsing the title string.

Default: **:**

lgBoxLinesOn

A boolean flag determining whether boxes should appear around **Legend** items.

Default: False

lgBoxLineColor

The hlu index of the color used to draw the box lines around the **Legend** items.

Default: `Foreground`

lgBoxLineThicknessF

Determines the thickness of the box lines drawn around the **Legend** items.

Default: 1.0

lgBoxLineDashPattern
The hlu index of the dash pattern used for the lines drawn around the **Legend** items.

Default: `Solidline`

lgBoxLineDashSegLenF
The length in NDC units of the dash pattern used for the box lines around the **Legend** items.

Default: 0.15

lgPerimOn
A boolean flag determining whether to draw a line around the perimeter of the **Legend**.

Default: True

lgPerimColor
The hlu index of the color used for the perimeter line.

Default: `Foreground`

lgPerimFill
The hlu index of the pattern used to fill the background of the **Legend** area. Only has an effect when the *lgPerimFillColor* has set to a value greater than NhlTRANSPARENT (-1).

Default: `HollowFill`

lgPerimFillColor
The hlu index of the color used to fill the background of the **Legend** area. Only has an effect when the *lgPerimFill* has a value greater than `HollowFill` (-1).

Default: `Background`

lgPerimThicknessF
Specifies the thickness of the perimeter line.

Default: 1.0

lgPerimDashPattern
Specifies the hlu index of the dash pattern used to draw the perimeter of the **Legend**.

Default: `SolidLine`

lgPerimDashSegLenF
The length in NDC units of the dash pattern used to draw the perimeter of the **Legend**.

Default: 0.15

MapPlot class resource descriptions

mpDataBaseVersion
This resource of type NhlTMapDataBaseVersion determines which database version **MapPlot** should use. Currently there are two choices:

`Ncarg4_0`

Use the original database as supplied with **MapPlot** in version 4.0. This database is out of date with respect to political boundaries and has much lower resolution than the Ncarg4_1 database. Nevertheless, for global scale plots that involve only geophysical boundaries, this database is the recommended one to use. Because there are so many fewer points, **MapPlot** can render the data much faster, particularly when fill is involved. This database contains 583 named geographical entities.

`Ncarg4_1`

Use the database created for version 4.1. This database contains 1085 named geographical entities arranged in an extensible hierarchical fashion. The database is organized in such a way that new subdivisions of existing areas can easily be added. The resolution is much higher than the original database, and the political boundaries are up-to-date as of mid-year 1998. Use of this database is recommended when plotting limited areas of the earth's surface, or any time up-to-date political boundaries are required.

Note that the set of names provided with the 4.1 database is different from set provided with the earlier database. Although you can switch between databases after creating a **mapplot**, if you have set any of the specifier resources (*mpFillAreaSpecifiers*, *mpOutlineSpecifiers*, or *mpMaskAreaSpecifiers*), you will be likely to encounter warning messages. These occur because names found in one database often do not occur in the same form in the other database. In particular, a hyphen character always separates multi-word names in the 4.0 database. The 4.1 database uses spaces except where a hyphen is part of the natural spelling of the name.

Default: `Ncarg4_0`

mpShapeMode

This resource of type NhlTMapShapeMode allows you to control whether the map projection preserves its intrinsic shape (*aspect ratio*), and in addition, if the aspect ratio is to be preserved, whether the **MapPlot** bounding box should be resized to fit the map projection extent exactly. It has three possible settings:

`FreeAspect`

Do not preserve the map projection aspect ratio. The projected area is "stretched" along either the horizontal or the vertical **View** axis such that it fills the specified **View** area.

`FixedAspectFitBB`

Preserve the map projection aspect ratio. The projected area is made as large as possible and centered within the **View** area. Then **MapPlot** resizes its **View** area such that its perimeter matches the bounding box of the projected area.

`FixedAspectNoFitBB`

Preserve the map projection aspect ratio. The projected area is made as large as possible and centered within the **View** area. **MapPlot** does not resize its **View** area.

Default: `FixedAspectFitBB`

mpAreaNames

mpAreaNames is a string array resource you can use to retrieve the names of all areas in a **MapPlot** database. You need to know these names, including specific details of their spelling, in order to use the specifier resources (*mpFillAreaSpecifiers*, *mpOutlineSpecifiers*, or *mpMaskAreaSpecifiers*). There is a unique name for each distinct area. However, the form of the names differs depending on the database in use.

You can also set this resource to rename the area names, perhaps to better represent the conventions of another locale. You must set the complete list of names at the same time. Keep in mind that the order of areas is fixed: a particular element of the array always refers to the same area in the database you are using regardless of the name you give it.

When using the Ncarg4_0 database:

All names that are compounds of several words use the hyphen character ('-') to separate the individual words. In several cases where the names would otherwise not be unique, an underscore character ('_') followed by a digit is appended to the end of the name. If you use a **SetValues** call to rename the areas, it is your responsibility to ensure that you do not duplicate the names of any areas. Only one of a group of identically named areas will remain accessible. See the Ncarg4_0 database table for the list of area names.

When using the Ncarg4_1 database:

Names that are compounds of several words use the space character (' ') to separate the individual words unless some or all of them are normally hyphenated. See the Ncarg4_1 database table for the list of area names.

Default: <dynamic>

mpAreaTypes

This read-only integer array resource contains the integer value of the type of each area in a **MapPlot** database. Each element gives the type of the area named by the corresponding element of the *mpAreaNames* array. The meaning and value of these types vary depending on the value of *mpMapDataBaseVersion*.

When using the Ncarg4_0 database:

Areas are classified into eight types. These types are as follows:

- (0) mpOcean
- (1) mpContinent
- (2) mpLargeIsland
- (3) mpSmallIsland
- (4) mpInlandWater
- (5) mpNational
- (6) mpUSStateLand
- (7) mpUSStateWater

See the Ncarg4_0 database table to find the area type associated with a particular area name.

When using the Ncarg4_1 database:

Areas are classified into four types. These types represent the four levels of the hierarchical classification scheme currently represented in the database. More types may be added if and when areas at other levels of the hierarchy are added to the database. The current types are as follows:

- (1) mpGeophysical
- (2) mpContinental
- (3) mpNational
- (4) mpState

See the Ncarg4_1 database table to find the area type associated with a particular area name.

Default: <dynamic>

mpFixedAreaGroups

MapPlot maintains two arrays that assign area group numbers to each area in a **MapPlot** database.

MapPlot uses the area's group numbers to determine its fill attributes, assuming *mpFillOn* is True and you do not explicitly specify fill attributes for the area. The *mpFixedAreaGroups* resource allows read-only access for informational purposes only to the fixed area group array, which, as its name implies, you cannot modify. The elements of this array have the same order as the *mpAreaNames* array. Within the array each area in the database is assigned to area group 1, 2, or 3. These are respectively the *Ocean* group, the *Land* group and the *InlandWater* group. For this reason *mpFixedAreaGroups* is sometimes known as the *geophysical* area groups array, since a map whose fill attributes are based on these area groups shows only physical as opposed to *political* features.

If fill is turned on and you set *mpFillBoundarySets* to the value `Geophysical` and do not explicitly specify any areas using the *mpFillAreaSpecifiers* resource, **MapPlot** will use the *mpFixedAreaGroups* group numbers to determine fill attributes for all areas of the map. For each of the three fill attribute types that has its *Mono* flag resource set False, the area's group number is used as an index into the fill attribute array resource. This yields, for *mpFillColors*, an HLU color index, for *mpFillPatterns*, an HLU fill pattern index, and for *mpFillScales*, a fill pattern scaling factor. A more complete description of the situations that result in **MapPlot** indexing the fill attribute array resources based on the contents of the *mpFixedAreaGroups* array is found in the discussion of area fill in the **MapPlot** object description.

Note that you can set the fill attributes of the area groups assigned by the *mpFixedAreaGroups* array using a set of convenience resources that act as named aliases for the *Ocean*, *Land*, and *InlandWater* elements of *mpFillColors*, *mpFillPatterns*, and *mpFillScales*. These resources are as follows: *mpOceanFillColor*, *mpOceanFillPattern*, *mpOceanFillScaleF*, *mpLandFillColor*, *mpLandFillPattern*, *mpLandFillScaleF*, *mpInlandWaterFillColor*, *mpInlandWaterFillPattern*, and *mpInlandWaterFillScaleF*.

When using the Ncarg4_0 database:
: See the Ncarg4_0 database table to find the fixed area group associated with a particular area name.

When using the Ncarg4_1 database:
: See the Ncarg4_1 database table to find the fixed area group associated with a particular area name.

Default: <dynamic>

mpDynamicAreaGroups

MapPlot maintains two arrays that assign area group numbers to each area in a **MapPlot** database. **MapPlot** uses the area's group numbers to determine its fill attributes, assuming *mpFillOn* is True and you do not explicitly specify fill attributes for the area. The *mpDynamicAreaGroups* resource allows you to access and, if desired, set the elements of the dynamic area group array. The elements of this array have the same order as the *mpAreaNames* array. By default, within the dynamic group arrray **MapPlot** assigns each area a group number such that adjoining politically distinct land areas are guaranteed to belong to a different group. For this reason the *mpDyanamicAreaGroups* array is sometimes known as the *political* groups array, since a map whose fill attributes are based on these area groups shows national and perhaps state boundaries as well as physical features. Anytime *mpFillBoundarySets* is set in such a way that national or U.S. state areas must be distinguished, and the fill attributes are not set explicitly, **MapPlot** uses the group numbers in the *mpDyanamicAreaGroups* array as indexes into the fill attribute array resources whose associated *Mono* flag resource is set False. A more thorough description of the situations that result in **MapPlot** indexing the fill attribute array resources based on the contents of the *mpDyanamicAreaGroups* array is found in the discussion of area fill in the **MapPlot** object description.

You may reassign the group numbers in the *mpDyanamicAreaGroups* array in any manner you please. At minimum and by default there are 10 area groups, but by setting the resource *mpAreaGroupCount* you can increase the number of area groups to any number up to a maximum of 255.

When using the Ncarg4_0 database:
    See the Ncarg4_0 database table to find the dynamic area group associated with a particular area name.
When using the Ncarg4_1 database:
    See the Ncarg4_1 database table to find the dynamic area group associated with a particular area name.

Default: <dynamic>

mpAreaGroupCount

mpAreaGroupCount specifies the total number of area groups, including the "default" group, the three fixed (or geophysical) groups, and the dynamic (or political) groups (a minimum of six). The minimum, as well as the default, value of *mpAreaGroupCount* is 10. The maximum value is 255. Attempts to set *mpAreaGroupCount* to values outside this range result in a warning and a restoration of the default value.

Default: 10

mpOutlineOn

This boolean resource is the master switch for drawing of **MapPlot** area outlines. If True, the **MapPlot** enables drawing of map area outlines. Otherwise, these outlines will not appear, regardless of the setting of any other area-outline-related resources. However, note that this resource does not apply to **MapPlot** grid or limb lines or to the perimeter line.

Default: True

mpOutlineDrawOrder

This resource of type NhlTDrawOrder determines when **MapPlot** area outlines are drawn relative to other elements of the plot. There are three choices:

`PreDraw`
    Draw the area outlines before the standard draw phase; the lines will overlay **MapPlot** fill and grid or limb lines also drawn during the predraw phase but will underlie **MapPlot** labels and perimeter lines drawn during the predraw phase and anything drawn during the draw and postdraw phases.

`Draw`
    Draw the area outlines during the standard draw; the lines will overlay any elements drawn during the predraw phase, as well as **MapPlot** fill and grid or limb lines drawn during the standard draw phase, but will underlie **MapPlot** labels and perimeter lines drawn during the draw phase and anything drawn during the postdraw phase.

`PostDraw`
    Draw the lines after the standard draw; the lines will overlay all **MapPlot** elements except

for labels and perimeter lines drawn during the postdraw phase.

Default: `PostDraw`

mpOutlineBoundarySets

You set this resource of type NhlTMapBoundarySets to specify the basic boundary set used to render area outlines. Given the basic boundaries, you can then add other outline areas by specifying area names in the *mpOutlineSpecifiers* string array resource. There are six choices:

`NoBoundaries`

MapPlot will draw area outlines only for areas specified by name in *mpOutlineSpecifiers*.

`Geophysical`

MapPlot will draw area outlines for geophysical features: continents, oceans, islands, and inland water areas. Specific national boundary lines may be added by name in *mpOutlineSpecifiers*.

`National`

MapPlot will draw area outlines delineating all national boundary areas as contained in the **MapPlot** database being used. All the geophysical features specified by `Geophysical` are implicitly included. You may add U.S. State area outlines by name using the *mpOutlineSpecifiers* resource.

`USStates`

MapPlot will draw area outlines delineating the states of the United States. When using the Ncarg4_0 database, Alaska and Hawaii are not included, although a few areas outside the U.S. proper, including the Bahamas and a few lakes extending into Canadian territory, are drawn. When using the Ncarg4_1 database, Alaska and Hawaii are drawn, but no non-U.S. territory is included. You may add geophysical and/or national areas by name using the *mpOutlineSpecifiers* resource.

`GeophysicalAndUSStates`

MapPlot will draw area outlines of all global geophysical features as well as the states of the U.S. Other national boundaries do not appear unless you specify them by name using the *mpOutlineSpecifiers* resource.

`AllBoundaries`

MapPlot draws area outlines of all national boundaries as well as all U.S. State boundaries. All geophysical features implicitly appear. When the Ncarg4_1 database is in use, specifying an area name whose *mpAreaType* is 2 or 1 (`mpGeophysical` or `mpContinental`) has the effect of suppressing type 3 or higher boundaries within its borders. For instance, setting *mpOutlineSpecifiers* to the value "Africa" would have the effect of suppressing national boundaries within the African continent.

Default: `Geophysical`

mpOutlineSpecifiers

You use the array resource *mpOutlineSpecifiers* to specify particular area outlines **MapPlot** is to draw by name. These areas are added to the basic outline area set specified using the resource *mpOutlineBoundarySets*. Note that the names from different **MapPlot** databases are not necessarily compatible. Therefore, if *mpOutlineSpecifiers* is non-NULL when you set *mpMapDatabaseVersion* to a different value, you are likely to encounter warning messages unless you also set *mpOutlineSpecifiers* using names compatible with the new database. You can retrieve a list of compatible names by getting the value of *mpAreaNames*. Case is not significant for *mpOutlineSpecifier* string values.

When using the Ncarg4_0 database:

In addition to the individual area names, you may include certain pre-defined groups of areas using a number of pre-defined string constants. Moreover, there is a substring matching facility that allows you to specify groups of areas sharing a common substring.

When using the Ncarg4_1 database:

Only the "NullArea" string constant is defined. Due to the hierarchical nature of this database, however, many area groupings occur as named entities in their own right. For instance, you can draw all islands of the Pacific Ocean by including the string "Pacific Islands" as an element of *mpOutlineSpecifiers*. The substring matching facility is not currently supported.

Default: NULL

mpAreaMaskingOn

This boolean resource is the master switch for enabling the area masking facility. If True, **MapPlot** enables area masking, causing areas named in the *mpMaskAreaSpecifiers* array to remain unfilled, consequently allowing previously drawn plot elements to be visible within the areas' outlines. If False, no masking is performed, regardless of the contents of *mpMaskAreaSpecifiers*. For convenience, setting *mpMaskAreaSpecifiers* causes *mpAreaMaskingOn* to be set True, if it is not explicitly set otherwise in the same call.

Default: False

mpMaskAreaSpecifiers

You use the array resource *mpMaskAreaSpecifiers* to specify the names of areas **MapPlot** is to mask. These areas are subtracted from the fill area set specified using the resource *mpFillBoundarySets* and *mpFillAreaSpecifiers*. There is an order of precedence for fill and masking. Explicitly named areas take precedence over area groupings, and small areas take precedence over enclosing larger areas. Otherwise masking takes precedence over filling.

Note that the names from different **MapPlot** databases are not necessarily compatible. Therefore, if *mpMaskAreaSpecifiers* is non-NULL when you set *mpMapDatabaseVersion* to a different value, you are likely to encounter warning messages unless you also set *mpMaskAreaSpecifiers* using names compatible with the new database. You can retrieve a list of compatible names by getting the value of *mpAreaNames*. Case is not significant for *mpMaskAreaSpecifier* string values.

When using the Ncarg4_0 database:

In addition to the individual area names, you may include certain pre-defined groups of areas using a number of pre-defined string constants. Moreover, there is a substring matching facility that allows you to specify groups of areas sharing a common substring.

When using the Ncarg4_1 database:

Only the "NullArea" string constant is defined. Due to the hierarchical nature of this database, however, many area groupings occur as named entities in their own right. For instance, you can draw all islands of the Pacific Ocean by including the string "Pacific Islands" as an element of *mpOutlineSpecifiers*. The substring matching facility is not currently supported.

Default: NULL

mpFillOn

This boolean resource is the master switch for drawing of **MapPlot** area fill. If True, the **MapPlot** enables drawing of map area fill. Otherwise, no map fill will appear, regardless of the setting of any other area-fill-related resources.

Default: False

mpFillDrawOrder

This resource of type NhlTDrawOrder determines when **MapPlot** area fill is drawn relative to other elements of the plot. There are three choices:

`PreDraw`
: Draw area fill before the standard draw phase; the fill underlies all other **MapPlot** elements drawn during the predraw phase and anything drawn during the draw and postdraw phases.

`Draw`
: Draw area fill during the standard draw; the fill overlays anything drawn during the predraw phase but underlies any other **MapPlot** elements drawn during the draw phase and anything drawn during the postdraw phase.

`PostDraw`
: Draw area fill after the standard draw; the fill overlays anything drawn during the predraw and draw phases but underlies any other **MapPlot** elements drawn during the postdraw phase.

Default: `Draw`

mpFillBoundarySets

You set this resource of type NhlTMapBoundarySets to specify the basic boundary set used for fill areas. Given the basic boundaries, you can then add other areas by specifying area names outlines in the *mpFillAreaSpecifiers* string array resource. You can also desiginate areas to be masked (left unfilled) by specifying area names in the *mpMaskAreaSpecifiers* string array resource.

`NoBoundaries`
: **MapPlot** will draw area fill only for areas specified by name in *mpFillAreaSpecifiers* and not specified in *mpMaskSpecifiers*.

`Geophysical`
: **MapPlot** will draw area fill for geophysical features: land, oceans, and inland water bodies. You may specify by name other areas to be filled in the *mpFillAreaSpecifiers* resource.

`National`
: **MapPlot** will individually fill all national boundary areas. Geophysical features are implicitly included. You may specify by name other areas to be filled in the *mpFillAreaSpecifiers* resource.

`USStates`
: **MapPlot** will fill regions comprising the states of the United States. When using the Ncarg4_0 database, Alaska and Hawaii are not included, although a few areas outside the U.S. proper, including the Bahamas and a few lakes extending into Canadian territory, are drawn. When using the Ncarg4_1 database, Alaska and Hawaii are drawn, but no non-U.S. territory is included. You may add geophysical and/or national areas by name using the *mpFillAreaSpecifiers* resource.

`GeophysicalAndUSStates`
: **MapPlot** will draw area fill for all global geophysical features as well as for the states of the United States. Other national boundaries do not appear unless you specify them by name using the *mpFillAreaSpecifiers* resource.

`AllBoundaries`
: **MapPlot** draws area fill for all national boundaries as well as all U.S. states and inland

water. All geophysical features implicitly appear. You can explicitly set the fill attributes for areas whose names are specified in the *mpFillAreaSpecifiers* resource. When the Ncarg4_1 database is in use, specifying an area name whose *mpAreaType* is 1 or 2 (`mpGeophysical` or `mpContinental`) has the effect of suppressing type 3 or higher boundaries within its borders. For instance, setting *mpOutlineSpecifiers* to the value "Africa" would have the effect of suppressing national boundaries within the African continent.

Default: `Geophysical`

mpFillPatternBackground

mpFillPatternBackground specifies the HLU index of the color used for the background in areas filled with any HLU fill pattern besides `SolidFill`. If this resource has the value `Transparent` (-1), no color will be appear behind the fill pattern. The effect will be that the pattern appears transparently on top of anything already drawn.

Default: `Background`

mpMonoFillColor

When set True, all **MapPlot** fill areas (other than those whose color is set explicitly using the *mpSpecifiedFillColors* array resource) are set to a single color, as specified by the value of the scalar resource *cnFillColor*. When False, the elements of the array resource *cnFillColors* control the fill color indexes for **MapPlot** areas based on the area group numbers assigned to each area.

Default: False

mpFillColor

When *cnMonoFillColor* is set True, this resource of type NhlTColorIndex sets a uniform fill color for all **MapPlot** areas, except for those whose fill color is set explicitly using the *mpSpecifiedFillColors* array resource.

Default: `Foreground`

mpFillColors

If *mpMonoFillColor* is False, each element of the array specifies the HLU color index used to fill the **MapPlot** areas belonging a particular area group. The first element of the array specifies the default color for regions within the map projection area that contain neither a filled nor a masked area. The second, third, and fourth elements specify colors for the fixed (geophysical) area groups: the *Ocean* group, the *Land* group, and the *InlandWater* group. For convenience, there are named alias resources for these first four elements of the array: *mpDefaultFillColor*, *mpOceanFillColor*, *mpLandFillColor*, and *mpInlandWaterFillColor*. If you set any of the alias resources at the same time as the *mpFillColors* array resource is set, the named alias resource value overrides. The remaining elements (by default, the fifth through the tenth elements) specify colors for the dynamic groups. If *mpAreaGroupCount* is increased from its minimum (and default) value of 10, more elements become available for additional dynamic group colors.

Default: By default the first 10 elements are assigned the following sequence of HLU color indexes: 16, 10, 8, 10, 26, 22, 11, 23, 13, 19. This choice of colors is intended to look reasonable with the "default" color map.

mpMonoFillPattern

When set True, all **MapPlot** fill areas (other than those whose pattern is set explicitly using the *mpSpecifiedFillPatterns* array resource) are set to a single pattern, as specified by the value of the scalar resource *cnFillPattern*. When False, the elements of the array resource *cnFillPatterns* control the fill pattern indexes for **MapPlot** areas based on the area group numbers assigned to each area.

Default: True

mpFillPattern

When *cnMonoFillPattern* is set True, this resource of type NhlTFillIndex sets a uniform fill pattern for all **MapPlot** areas, except for those whose fill pattern is set explicitly using the *mpSpecifiedFillPatterns* array resource.

Default: `SolidFill`

mpFillPatterns

If *mpMonoFillPattern* is False, each element of the array specifies the HLU fill pattern index used to fill the **MapPlot** areas belonging a particular area group. The first element of the array specifies the default pattern for regions within the map projection area that contain neither a filled nor a masked area. The second, third, and fourth elements specify patterns for the fixed (geophysical) area groups: the *Ocean* group, the *Land* group, and the *InlandWater* group. For convenience, there are named alias resources for these first four elements of the array: *mpDefaultFillPattern*, *mpOceanFillPattern*, *mpLandFillPattern*, and *mpInlandWaterFillPattern*. If you set any of the alias resources at the same time as the *mpFillPatterns* array resource is set, the named alias resource value overrides. The remaining elements (by default, the fifth through the tenth elements) specify patterns for the dynamic groups. If *mpAreaGroupCount* is increased from its minimum (and default) value of 10, more elements become available for additional dynamic area group patterns.

Default: **MapPlot** assigns patterns sequentially, beginning with HLU fill pattern index 1 for the first element of *mpFillPatterns*.

mpMonoFillScale

When set True, all **MapPlot** fill areas (other than those whose fill scale is set explicitly using the *mpSpecifiedFillScales* array resource) use a single fill pattern scale factor, as specified by the value of the scalar resource *cnFillScaleF*. When False, the elements of the array resource *cnFillScales* control the fill scale values for **MapPlot** areas based on the area group numbers assigned to each area.

Default: True

mpFillScaleF

When *cnMonoFillPattern* is set True, this floating point resource sets a uniform fill pattern scale factor for all **MapPlot** areas, except for those whose fill scale factor is set explicitly using the *mpSpecifiedFillScales* array resource.

Default: 1.0

mpFillScales

If *mpMonoFillPattern* is False, each element of the array specifies the scale factor applied to the fill pattern used to fill **MapPlot** areas belonging a particular area group. The first element of the array specifies the default fill pattern scale factor for regions within the map projection area that contain neither a filled nor a masked area. The second, third, and fourth elements specify scale factors for the fixed (geophysical) area groups: the *Ocean* group, the *Land* group, and the *InlandWater* group. For convenience, there are named alias resources for these first four elements of the array: *mpDefaultFillScaleF*, *mpOceanFillScaleF*, *mpLandFillScaleF*, and *mpInlandWaterFillScaleF*. If you set any of the alias resources at the same time as the *mpFillScales* array resource is set, the named alias resource value overrides. The remaining elements (by default, the fifth through the tenth elements) specify patterns for the dynamic groups. If *mpAreaGroupCount* is increased from its minimum (and default) value of 10, more elements become available for additional dynamic area group fill scale values.

Default: 1.0 for all elements

mpFillAreaSpecifiers

You use the array resource *mpFillAreaSpecifiers* to specify explicitly the names of areas **MapPlot** is to fill. Your purpose may be to include an area that would not otherwise be drawn (such as an interior national boundary when *mpFillBoundarySets* has the value `Geophysical`) or it may be simply that you want to control the area's fill attributes explicitly using the associated specified fill attribute arrays. The named areas are added to the basic set of fill areas specified using the resource *mpFillBoundarySets*.

You may explicitly specify the fill attributes of areas named in the *mpFillAreaSpecifiers* array using any or all of the three associated attribute array resources, *mpSpecifiedFillColors*, *mpSpecifiedFillPatterns*, and *mpSpecifiedFillScales*. The elements of each of these arrays correspond one-for-one with the elements of the *mpFillAreaSpecifiers* array. Attributes set using these arrays override all other resources involving fill attributes, including the fill attribute *Mono* flags. Each of these arrays accept a pre-defined special value allowing you to specify individual elements as "unset."

If you also using area masking, note that there is an order of precedence for fill and masking. Explicitly named areas take precedence over area groupings, and small areas take precedence over enclosing larger areas. Otherwise masking takes precedence over filling.

Note that the names from different **MapPlot** databases are not necessarily compatible. Therefore, if *mpFillAreaSpecifiers* is non-NULL when you set *mpMapDatabaseVersion* to a different value, you are likely to encounter warning messages unless you also set *mpFillAreaSpecifiers* using names compatible with the new database. You can retrieve a list of compatible names by getting the value of *mpAreaNames*. Case is not significant for *mpFillAreaSpecifier* string values.

When using the Ncarg4_0 database:

Within the list you may include groups of areas defined by a number of pre-defined string constants. In addition, a substring matching facility allows you to specify groups of areas that share a common substring. When a fill attribute for a specified fill area is not explicitly set, **MapPlot** uses its normal method of setting fill attributes based on area group assignments. The group number is chosen from the *mpFixedAreaGroups* array or from the *mpDynamicAreaGroups* array depending on the setting of the resource *mpSpecifiedFillPriority*. You can override the general priority for an individual area by prepending an exclamation point ('!') to the area's name.

When using the Ncarg4_1 database:

Only the "NullArea" string constant is defined. The substring matching facility is not currently supported. When a fill attribute for a specified fill area is not explicitly set, **MapPlot** uses its normal method of setting fill attributes based on area group assignments. The group number is chosen from the *mpFixedAreaGroups* array if the area's type is 1 (mpGeophysical) or if its type is 2 (mpContinental) and its parent is Water. Otherwise, the group number is chosen from the *mpDynamicAreaGroups* array. The use of the exclamation point ('!') prefix and the *mpSpecifiedFillPriority* resource is not supported.

Default: NULL

mpSpecifiedFillDirectIndexing

This boolean resource determines whether the values contained in the specified fill attribute array resource (*mpSpecifiedFillColors*, *mpSpecifiedFillPatterns*, and *mpSpecifiedFillScales*) resource *mpFillAreaColors* directly specify the attribute value, or whether they specify an area group number used as an index into the regular fill attribute array resources (*mpFillColors*, *mpFillPatterns*, and *mpFillScales*). When *mpSpecifiedFillDirectIndexing* is set True, the values of *mpSpecifiedFillColors* elements represent HLU color indexes, the values of *mpSpecifiedFillPatterns* elements represent HLU pattern indexes, and the values of *mpSpecifiedFillPatterns* elements represent fill pattern scaling factors. Otherwise, the elements of these arrays are converted into integers that represent area group numbers.

Default: True

mpSpecifiedFillPriority

When using the Ncarg4_0 database:

Based on the setting of this resource of type NhlTSpecifiedFillPriority, **MapPlot** decides how to choose a group number for indexing into the regular fill attribute arrays for areas specified in the *mpFillAreaSpecifiers* resource. The group number is only used to determine fill attributes that have not been specified explicitly using the specified fill attribute arrays. The choices are as follows:

`GeophysicalPriority`

Use group numbers from the *mpFixedAreaGroups* array as an index into the fill attribute arrays.

`PoliticalPriority`

Use group numbers from the *mpDynamicAreaGroups* array as an index into the fill attribute arrays.

You can reverse the sense of the general fill priority value for individual named areas in the *mpFillAreaSpecifiers* resource by prefixing the name string with the exclamation point character ('!').

When using the Ncarg4_1 database:

The *mpSpecifiedFillPriority* resource is not supported.

Default: `GeophysicalPriority`

mpSpecifiedFillColors

You may use this array resource to set explicitly the fill color of any **MapPlot** area specified by name in the *mpFillAreaSpecifiers* array resource, overriding the usual method of determining an area's fill color based on its assigned area group numbers. The elements of the *mpSpecifiedFillColors* array correspond one-for-one to the elements of the *mpFillAreaSpecifiers* array.

Depending on the setting of the boolean resource *mpSpecifiedFillDirectIndexing*, you control the level of indirection in the way **MapPlot** interprets the values in *mpSpecifiedFillColors*. If *mpSpecifiedFillDirectIndexing* is True, **MapPlot** treats the values of *mpSpecifiedFillColors* as

HLU color indexes directly determining the fill color of the specified areas. Otherwise, **MapPlot** considers each value as a **MapPlot** group number that temporarily overrides the group numbers normally assigned to the area. In this case, **MapPlot** determines the HLU fill color by using this temporary group number as an index into the *mpFillColors* array.

You may cause any element of the *mpFillAreaSpecifiers* array to retain its normal color by setting the corresponding element of *mpSpecifiedFillColors* to the special value `NullColor` (-1). If *mpSpecifiedFillColors* has fewer elements than *mpFillAreaSpecifiers* **MapPlot** sets all the extra elements of *mpFillAreaSpecifiers* using its normal scheme. If *mpSpecifiedFillColors* has more elements, **MapPlot** ignores them. If any of the elements of *mpSpecifiedFillColors* are invalid, **MapPlot** issues a warning and assigns the area a color using the normal scheme.

Default: <dynamic>

mpSpecifiedFillPatterns

You may use this array resource to set explicitly the fill pattern of any **MapPlot** area specified by name in the *mpFillAreaSpecifiers* array resource, overriding the usual method of determining an area's fill pattern based on its assigned area group numbers. The elements of the *mpSpecifiedFillPatterns* array correspond one-for-one to the elements of the *mpFillAreaSpecifiers* array.

Depending on the setting of the boolean resource *mpSpecifiedFillDirectIndexing*, you control the level of indirection in the way **MapPlot** interprets the values in *mpSpecifiedFillPatterns*. If *mpSpecifiedFillDirectIndexing* is True, **MapPlot** treats the values of *mpSpecifiedFillPatterns* as HLU pattern indexes directly determining the fill pattern of the specified areas. Otherwise, **MapPlot** considers each value as a **MapPlot** group number that temporarily overrides the group numbers normally assigned to the area. In this case, **MapPlot** determines the HLU fill pattern by using this temporary group number as an index into the *mpFillPatterns* array.

You may cause any element of the *mpFillAreaSpecifiers* array to retain its normal pattern by setting the corresponding element of *mpSpecifiedFillPatterns* to the special value `NullFill` (-1). If *mpSpecifiedFillPatterns* has fewer elements than *mpFillAreaSpecifiers*, **MapPlot** sets all the extra elements of *mpFillAreaSpecifiers* using its normal scheme. If *mpSpecifiedFillPatterns* has more elements, **MapPlot** ignores them. If any of the elements of *mpSpecifiedFillPatterns* are invalid, **MapPlot** issues a warning and assigns the area a pattern using the normal scheme.

Default: <dynamic>

mpSpecifiedFillScales

You may use this array resource to explicitly set the fill pattern scale factor of any **MapPlot** area specified by name in the *mpFillAreaSpecifiers* array resource, overriding the usual method of determining an area's fill scale based on its assigned area group numbers. The elements of the *mpSpecifiedFillScales* array correspond one-for-one to the elements of the *mpFillAreaSpecifiers* array.

Depending on the setting of the boolean resource *mpSpecifiedFillDirectIndexing*, you control the level of indirection in the way **MapPlot** interprets the values in *mpSpecifiedFillScales*. If *mpSpecifiedFillDirectIndexing* is True, **MapPlot** treats the values of *mpSpecifiedFillScales* as floating point values directly determining the fill scale of the specified areas. Otherwise, **MapPlot** converts each number into an integer and treats it as a **MapPlot** group number that temporarily overrides the group numbers normally assigned to the area. In this case, **MapPlot** determines the fill scale value by using this temporary group number as an index into the *mpFillScales* array.

You may cause any element of the *FillAreaSpecifiers* array to retain its normal fill scale value by setting the corresponding element of *mpSpecifiedFillScales* to the special value 0.0. If *mpSpecifiedFillScales* has fewer elements than *mpFillAreaSpecifiers*, **MapPlot** sets all the extra elements of *mpFillAreaSpecifiers* using its normal scheme. If *mpSpecifiedFillScales* has more elements, **MapPlot** ignores them. If any of the elements of *mpSpecifiedFillScales* are invalid, **MapPlot** issues a warning and assigns the area a scale using the normal scheme.

Note that a kludge is required if *mpSpecifiedFillDirectIndexing* is set False, so that you are indexing into the *mpFillScales* array using group numbers, and you want to specify that a fill scale be determined by indexing into element 0 of *mpFillScales*. You would need to set the element of *mpSpecifiedFillScales* using a number larger than 0.0 but less than 1.0. If you set the value to 0.0, it would be interpreted as the special value causing **MapPlot** to use its "normal" method of determining the fill scale for the area.

Default: <dynamic>

mpDefaultFillColor

This resource sets the HLU index of the color used to fill all areas within the map projection that, given that *mpFillOn* is True, are neither filled nor masked. *mpDefaultFillColor* is constrained to values greater than `Transparent` (-1). This resource provides an alternate method of setting the value of the first element of *mpFillColors*. If both resources are set in the same call, the value of *mpDefaultFillColor* takes precedence.

Default: 16

mpDefaultFillPattern

This resource sets the HLU index of the fill pattern used to fill all areas within the map projection that, given that *mpFillOn* is True, are neither filled nor masked. If *mpDefaultFillPattern* has the value `HollowFill` (-1), there will be no fill of the ocean area, regardless of the setting of *mpDefaultFillColor*. This resource provides an alternate method of setting the value of the first element of *mpFillPatterns*. If both resources are set in the same call, the value of *mpDefaultFillPattern* takes precedence.

Default: `SolidFill`

mpDefaultFillScaleF

The *mpDefaultFillScaleF* resource controls the scaling of the pattern used to fill all areas within the map projection that, given that *mpFillOn* is True, are neither filled nor masked. It has no effect for solid fill (`SolidFill`, fill index 0). Values greater than 1.0 make the pattern spacing bigger than the default spacing, resulting in a fill that appears less dense. Values less than 1.0 have the opposite effect. Values less than or equal to 0.0 are invalid, generate a warning message, and are reset to the default value, 1.0. This resource provides an alternate method of setting the value of the first element of *mpFillScales*. If both resources are set in the same call, the value of *mpDefaultFillScaleF* takes precedence.

Default: 1.0

mpOceanFillColor

This resource sets the HLU index of the color used to fill areas representing the oceans. *mpOceanFillColor* is constrained to values greater than -1 (`Transparent`). This resource provides an alternate method of setting the value of the second element of *mpFillColors*. If both resources are set in the same call, the value of *mpOceanFillColor* takes precedence.

Default: 10

mpOceanFillPattern

This resource sets the HLU index of the fill pattern used to fill areas representing the oceans. If *mpOceanFillPattern* has the value `HollowFill` (-1), there will be no fill of the ocean area, regardless of the setting of *mpOceanFillColor*. This resource provides an alternate method of setting the value of the second element of *mpFillPatterns*. If both resources are set in the same call, the value of *mpOceanFillPattern* takes precedence.

Default: `SolidFill`

mpOceanFillScaleF

The *mpOceanFillScaleF* resource controls the scaling of the pattern used to fill areas representing the oceans. It has no effect for solid fill (`SolidFill`, fill pattern index 0). Values greater than 1.0 make the pattern spacing bigger than the default spacing, resulting in a fill that appears less dense. Values less than 1.0 have the opposite effect. Values less than or equal to 0.0 are invalid, generate a warning message, and are reset to the default value, 1.0. This resource provides an alternate method of setting the value of the second element of *mpFillScales*. If both resources are set in the same call, the value of *mpOceanFillScaleF* takes precedence.

Default: 1.0

mpLandFillColor

This resource sets the HLU index of the color used to fill areas of land not given a special group color. *mpLandFillColor* is constrained to values greater than -1 (`Transparent`). This resource provides an alternate method of setting the value of the third element of *mpFillColors*. If both resources are set in the same call, the value of *mpLandFillColor* takes precedence.

Default: 8

mpLandFillPattern

This resource sets the HLU index of the fill pattern used to fill areas of land not given a special group color. If *mpLandFillPattern* has the value `HollowFill` (-1), there will be no fill of the land area, regardless of the setting of *mpInlandWaterFillColor*. This resource provides an alternate method of setting the value of the third element of *mpFillPatterns*. If both resources are set in the same call, the value of *mpLandFillPattern* takes precedence.

Default: `SolidFill`

mpLandFillScaleF

The *mpLandFillScaleF* resource controls the scaling of the pattern used to fill areas of land not given a special group color. It has no effect for solid fill (`SolidFill`, fill pattern index 0). Values greater than 1.0 make the pattern spacing bigger than the default spacing, resulting in a fill that appears less dense. Values less than 1.0 have the opposite effect. Values less than or equal to 0.0 are invalid, generate a warning message, and are reset to the default value, 1.0. This resource provides an alternate method of setting the value of the third element of *mpFillScales*. If both resources are set in the same call, the value of *mpLandFillScaleF* takes precedence.

Default: 1.0

mpInlandWaterFillColor

This resource sets the HLU index of the color used to fill areas representing bodies of water except for the oceans. *mpInlandWaterFillColor* is constrained to values greater than -1 (`Transparent`). This resource provides an alternate method of setting the value of the fourth element of *mpFillColors*. If both resources are set in the same call, the value of *mpInlandWaterFillColor* takes precedence.

Default: 10

mpInlandWaterFillPattern

This resource sets the HLU index of the fill pattern used to fill areas representing bodies of water except for the oceans. If *mpInlandWaterFillPattern* has the value `HollowFill` (-1), there will be no fill of the inland water area, regardless of the setting of *mpInlandWaterFillColor*. This resource provides an alternate method of setting the value of the fourth element of *mpFillPatterns*. If both resources are set in the same call, the value of *mpInlandWaterFillPattern* takes precedence.

Default: `SolidFill`

mpInlandWaterFillScaleF

The *mpInlandWaterFillScaleF* resource controls the scaling of the pattern used to fill areas representing bodies of water except for the oceans. It has no effect for solid fill (`SolidFill`, fill pattern index 0). Values greater than 1.0 make the pattern spacing bigger than the default spacing, resulting in a fill that appears less dense. Values less than 1.0 have the opposite effect. Values less than or equal to 0.0 are invalid, generate a warning message, and are reset to the default value, 1.0. This resource provides an alternate method of setting the value of the fourth element of *mpFillScales*. If both resources are set in the same call, the value of *mpInlandWaterFillScaleF* takes precedence.

Default: 1.0

mpGeophysicalLineColor

This resource sets the HLU index of a color used to render interior national boundary outlines. This resource applies to the outlines of all continents and islands, except that when one or more U.S. states are drawn, the portion of North America whose outline is contiguous with the U.S. border is rendered using *USStatesLineColor*.

Default: `Foreground`

mpGeophysicalLineDashPattern

This resource sets the HLU index of a dash pattern used to render interior national boundary outlines. This resource applies to the outlines of all continents and islands, except that when one or more U.S. states are drawn, the portion of North America whose outline is contiguous with the U.S. border is rendered using *USStatesLineDashPattern*.

Default: `SolidLine`

mpGeophysicalLineDashSegLenF

This resource specifies the length of each segment of the dash patterns used to draw geophysical boundary outlines. It is the length in NDC before the dash pattern repeats itself. This resource automatically scales with changes in the size of the viewport of the **MapPlot** object. This resource applies to the outlines of all continents and islands, except that when one or more U.S. states are drawn, the portion of North America whose outline is contiguous with the U.S. border is rendered using *USStatesLineDashSegLenF*. **MapPlot** sets the default value of *mpNationalLineDashSegLenF* dynamically based on the ratio of the actual plot viewport width to a reference viewport width of 0.6 and a reference dash segment length of 0.15.

Default: 0.15 (for a viewport width of 0.6)

mpGeophysicalLineThicknessF

Sets the thickness of the lines used to draw geophysical boundary outlines. The value acts as a multiplier of a (device-dependent) unit thickness. This resource applies to the outlines of all continents and islands, except that when one or more U.S. states are drawn, the portion of North America whose outline is contiguous with the U.S. border is rendered using *USStatesLineThicknessF*.

Default: 1.0

mpUSStateLineColor

This resource sets the HLU index of a color used to render interior U.S. state boundary outlines. This resource affects all exterior boundaries of the U.S. if *any* U.S. state outlines are included the **MapPlot**. Otherwise, U.S. boundaries are rendered like other national boundaries using *mpNationalLineColor* for interior boundaries and *mpGeophysicalLineColor* for lines separating land and ocean.

Default: `Foreground`

mpUSStateLineDashPattern

This resource sets the HLU index of a dash pattern used to render U.S. state boundary outlines. This resources affects all exterior boundaries of the U.S. if *any* U.S. state outlines are included the **MapPlot**. Otherwise, U.S. boundaries are rendered like other national boundaries using *mpNationalLineDashPattern* for interior boundaries and *mpGeophysicalLineDashPattern* for lines separating land and ocean. Note that this resource applies only to lines used to draw interior national boundaries. Lines separating land and oceans are not affected.

Default: `SolidLine`

mpUSStateLineDashSegLenF

This resource specifies the length of each segment of the dash patterns used to draw U.S. state boundary outlines. It is the length in NDC before the dash pattern repeats itself. This resources affects all exterior boundaries of the U.S. if *any* U.S. state outlines are included the **MapPlot**. Otherwise, U.S. boundaries are rendered like other national boundaries using *mpNationalLineDashSegLenF* for interior boundaries and *mpGeophysicalLineDashSegLenF* for lines separating land and ocean. *mpUSStateLineDashSegLenF* automatically scales with changes in the size of the viewport of the **MapPlot** object. **MapPlot** sets the default value of *mpNationalLineDashSegLenF* dynamically based on the ratio of the actual plot viewport width to a reference viewport width of 0.6 and a reference dash segment length of 0.15.

Default: 0.15 (for a viewport width of 0.6)

mpUSStateLineThicknessF

Sets the thickness of the lines used to draw U.S. state boundary outlines. The value acts as a multiplier of a (device-dependent) unit thickness. This resources affects all exterior boundaries of the U.S. if *any* U.S. state outlines are included the **MapPlot**. Otherwise, U.S. boundaries are rendered like other national boundaries using *mpNationalLineThicknessF* for interior boundaries and *mpGeophysicalLineThicknessF* for lines separating land and ocean.

Default: 1.0

mpNationalLineColor

This resource sets the HLU index of a color used to render interior national boundary outlines. Note that this resource applies only to lines used to draw interior national boundaries. Lines separating land and oceans are not affected.

Default: `Foreground`

mpNationalLineDashPattern

This resource sets the HLU index of a dash pattern used to render interior national boundary outlines. Note that this resource applies only to lines used to draw interior national boundaries. Lines separating land and oceans are not affected.

Default: `SolidLine`

mpNationalLineDashSegLenF

This resource specifies the length of each segment of the dash patterns used to draw national boundary outlines. It is the length in NDC before the dash pattern repeats itself. This resource automatically scales with changes in the size of the viewport of the **MapPlot** object. Note that this resource applies only to lines used to draw interior national boundaries. Lines separating land and oceans are not affected. **MapPlot** sets the default value of *mpNationalLineDashSegLenF* dynamically based on the ratio of the actual plot viewport width to a reference viewport width of 0.6 and a reference dash segment length of 0.15.

Default: 0.15 (for a viewport width of 0.6)

mpNationalLineThicknessF

Sets the thickness of the lines used to draw national boundary outlines. The value acts as a multiplier of a (device-dependent) unit thickness. Note that this resource applies only to lines used to draw interior national boundaries. Lines separating land and oceans are not affected by this resource.

Default: 1.0

mpGridAndLimbOn

This boolean resource determines whether the **MapPlot** object draws grid lines representing latitude and longitude as well as (when appropriate) a limb line around the edge of the visible surface of the map projection.

Default: True

mpGridAndLimbDrawOrder

This resource of type NhlTDrawOrder determines when **MapPlot** draws the grid lines representing latitude and longitude relative to other elements of the plot. When a limb line representing the edge of the visible surface of the projection is required, **MapPlot** always renders it at the same time as the grid lines. There are three choices:

`PreDraw`

Draw grid lines before the standard draw phase; the grid lines overlay **MapPlot** fill drawn during the predraw phase but underlie other predraw phase **MapPlot** elements and anything drawn during the draw and postdraw phases.

`Draw`

Draw grid lines during the standard draw; the grid lines overlay anything drawn during the predraw phase and **MapPlot** fill drawn during the draw phase but underlie other draw phase **MapPlot** elements and anything drawn during the postdraw phase.

`PostDraw`
: Draw grid lines after the standard draw; the grid lines overlay anything drawn during the predraw and draw phases and **MapPlot** fill drawn during the postdraw phase but underlie other postdraw phase **MapPlot** elements.

Default: `PostDraw`

mpGridMaskMode

This resource of type NhlTMapGridMaskMode specifies how the **MapPlot** object should mask the grid representing latitudes and longitudes. There are seven choices:

`MaskNone`
: **MapPlot** does not mask the grid. Grid lines appear over the entire visible map projection area.

`MaskOcean`
: **MapPlot** masks the grid over areas of the map occupied by any part of the global ocean.

`MaskNotOcean`
: **MapPlot** masks the grid over areas of the map that are not part of the global ocean, including bodies of water within the interior of continents.

`MaskLand`
: **MapPlot** masks the grid over areas of the map that represent land. Bodies of water within the interior of continents remain unmasked.

`MaskNotLand`
: **MapPlot** masks the grid over all areas of the map that do not represent land, including bodies of water within the interior of continents.

`MaskFillArea`
: **MapPlot** masks the grid over filled or masked areas in the map. The grid is drawn only over 'background' areas within the map projection, that is, those areas of the **MapPlot** that would be filled (assuming fill is enabled) using the color specified by *mpDefaultFillColor*.

`MaskMaskArea`
: **MapPlot** masks the grid only over areas named in the *mpMaskSpecifiers* array.

Default: `MaskNone`

mpGridSpacingF

This resource specifies the spacing, in degrees, between **MapPlot** grid lines representing latitude and longitude. Use this resource if you want the spacing to be equal in latitude and longitude. Setting this resource overrides any values set for the resources *mpGridLatSpacingF* and *mpGridLonSpacingF* and causes them both to be set to the value given *mpGridSpacingF*.

Default: 15.0

mpGridLatSpacingF

This resource specifies the spacing, in degrees, between **MapPlot** grid lines representing latitude. Use this resource if you want to set the latitude spacing independently of the longitude spacing. Note that setting *mpGridSpacingF* at the same time or subsequently overrides the value of this resource.

Default: 15.0

mpGridLonSpacingF

This resource specifies the spacing, in degrees, between **MapPlot** grid lines representing longitude. Use this resource if you want to set the longitude spacing independently of the latitude spacing. Note that setting *mpGridSpacingF* at the same time or subsequently overrides the value of this resource.

Default: 15.0

mpGridMaxLatF

This resource specifies the maximum absolute value of the latitude at which a latitude grid line may appear. For instance, if you set *mpGridMaxLatF* to 70.0, no latitude lines will appear between 70 degrees and 90 degrees or between -70 degrees and -90 degrees. If the latitude spacing were set to 20.0, then, in this case, the highest and lowest latitude lines would be at 60 degrees and -60 degrees.

Default: 90.0

mpGridPolarLonSpacingF

This resource has an effect only when the map projection is such that the North and South poles project to single points. In this case, it controls the way longitude lines are continued in the area between the pole and the latitude line nearest the pole. Basically it allows you to set an alternate spacing for longitude lines near the poles, generally for the purpose of avoiding overcrowding. For instance, if *mpGridPolarLonSpacingF* is set to its default value of 90.0 and the longitudinal spacing is set to 15.0, then only the longitude lines that are multiples of 90.0 continue to the poles. Note that if the longitudinal spacing were set to 20.0, lines would continue only at multiples of 180.0, since there would be no longitude lines to continue at 90.0 and 270.0.

Default: 15.0

mpGridLineColor

This resource sets the HLU index of a color used to draw the grid representing latitudes and longitudes.

Default: `Foreground`

mpGridLineDashPattern

This resource sets the HLU index of a dash pattern used to draw the grid representing latitudes and longitudes.

Default: `SolidLine`

mpGridLineDashSegLenF

This resource specifies the length of each segment of the dash patterns used to draw the grid representing latitudes and longitudes. It is the length in NDC before the dash pattern repeats itself. This resource automatically scales with changes in the size of the viewport of the **MapPlot** object. **MapPlot** sets the default value of *mpGridLineDashSegLenF* dynamically based on the ratio of the actual plot viewport width to a reference viewport width of 0.6 and a reference dash segment length of 0.15.

Default: 0.15 (for a viewport width of 0.6)

mpGridLineThicknessF

This resource determines the thickness of the lines used to draw the grid representing latitudes and longitudes. The value acts as a multiplier of a (device-dependent) unit thickness.

Default: 1.0

mpLimbLineColor

This resource sets the HLU index of the color used for the limb line drawn at the edge of visibility in certain map projections.

Default: `Foreground`

mpLimbLineDashPattern

This resource sets the HLU index of a dash pattern used for the limb line drawn at the edge of visibility in certain map projections.

Default: `SolidLine`

mpLimbLineDashSegLenF

This resource specifies the length of each segment of the dash patterns used for the limb line drawn at the edge of visibility in certain map projections. It is the length in NDC before the dash pattern repeats itself. This resource automatically scales with changes in the size of the viewport of the **MapPlot** object. **MapPlot** sets the default value of *mpLimbLineDashSegLenF* dynamically based on the ratio of the actual plot viewport width to a reference viewport width of 0.6 and a reference dash segment length of 0.15.

Default: 0.15 (for a viewport width of 0.6)

mpLimbLineThicknessF

This resource determines the thickness of the limb line drawn at the edge of visibility in certain map projections. The value acts as a multiplier of a (device-dependent) unit thickness.

Default: 1.0

mpPerimOn

This boolean resource determines whether the **MapPlot** object draws a perimeter line around the edge of the map projection. Depending on the setting of the **MapTransformation** resource *mpEllipticalBoundary*, this perimeter may have a rectangular or elliptical shape.

Default: False

mpPerimDrawOrder

This resource of type NhlTDrawOrder determines when the perimeter line around the edge of the map projection is drawn relative to other elements of the plot. There are three choices:

`PreDraw`

Draw the perimeter before the standard draw phase; the perimeter overlays all other **MapPlot** elements drawn during the predraw phase except **MapPlot** labels but underlies anything drawn during the draw and postdraw phases.

`Draw`

Draw the perimeter during the standard draw; the perimeter overlays anything drawn during the predraw phase and any other **MapPlot** elements (except **MapPlot** labels) drawn during the draw phase but underlies anything drawn during the postdraw phase.

`PostDraw`

Draw the perimeter after the standard draw; the perimeter overlays anything drawn during the predraw and draw phases and all other **MapPlot** elements (except **MapPlot** labels) drawn during the postdraw phase.

Default: `Draw`

mpPerimLineColor

This resource sets the HLU index of the color used to render the perimeter line around the edge of the map projection.

Default: `Foreground`

mpPerimLineDashPattern

This resource sets the HLU index of a dash pattern used to render the perimeter line around the edge of the map projection.

Default: `SolidLine`

mpPerimLineDashSegLenF

This resource specifies the length of each segment of the dash patterns used to draw the perimeter line around the edge of the map projection. It is the length in NDC before the dash pattern repeats itself. This resource automatically scales with changes in the size of the viewport of the **MapPlot** object. **MapPlot** sets the default value of *mpPerimLineDashSegLenF* dynamically based on the ratio of the actual plot viewport width to a reference viewport width of 0.6 and a reference dash segment length of 0.15.

Default: 0.15 (for a viewport width of 0.6)

mpPerimLineThicknessF

This resource determines the thickness of the perimeter line around the edge of the map projection. The value acts as a multiplier of a (device-dependent) unit thickness.

Default: 1.0

mpLabelsOn

This boolean resource determines whether the **MapPlot** object draws labels identifying the equator ("EQ"), Greenwich Meridian ("GM"), International Dateline ("ID"), North Pole ("NP"), and South Pole ("SP").

Default: False

mpLabelDrawOrder

This resource of type NhlTDrawOrder determines when **MapPlot** labels are drawn relative to other elements of the plot. There are three choices:

`PreDraw`
: Draw labels before the standard draw phase; the labels overlay all other **MapPlot** elements drawn during the predraw phase but underlie anything drawn during the draw and postdraw phases.

`Draw`
: Draw labels during the standard draw; the labels overlay anything drawn during the predraw phase and any other **MapPlot** elements drawn during the draw phase but underlie anything drawn during the postdraw phase.

`PostDraw`
: Draw labels after the standard draw; the labels overlay anything drawn during the predraw and draw phases and all other **MapPlot** elements drawn during the postdraw phase.

Default: `PostDraw`

mpLabelFontHeightF

This resource controls the height in NDC of characters used in the **MapPlot** labels. The **MapPlot** text height scales with changes to the viewport width, unless you explicitly set *mpLabelFontHeightF* during the same call.

Default: <dynamic> -- 0.008 for a viewport width of 0.6

mpLabelFontColor

This resource specifies the hlu color index used to draw the characters used for **MapPlot** labels.

Default: True

MapTransformation class resource descriptions

mpProjection

This resource of type NhlTProjection defines the projection used for the map transformation. There are 10 choices:

`Orthographic`
: Transform using an aziumthal orthographic projection.

`Stereographic`
: Transform using an azimuthal stereographic projection.

`LambertEqualArea`
: Transform using the azimuthal Lambert Equal-Area projection.

`Gnomonic`
: Transform using the azimuthal gnomonic projection.

`AzimuthalEquidistant`
: Transform using an azimuthal equidistant projection.

`Satellite`
: Transform using an azimuthal satellite projection.

`Mollweide`
: Transform using a cylindrical Mollweide-type projection.

`Mercator`
: Transform using the cylindrical Mercator projection.

`CylindricalEquidistant`
: Transform using a cylindrical equidistant projection.

`LambertConformal`
: Transform using the conical Lambert Conformal projection.

Default: `CylindricalEquidistant`

mpLeftMapPosF

This read-only resource contains the coordinate of the left edge of the projected area in NDC space.

Default: 0.0

mpRightMapPosF

This read-only resource contains the coordinate of the right edge of the projected area in NDC space.

Default: 1.0

mpBottomMapPosF

This read-only resource contains the coordinate of the bottom edge of the projected area in NDC space.

Default: 0.0

mpTopMapPosF

This read-only resource contains the coordinate of the top edge of the projected area in NDC space.

Default: 1.0

mpCenterLatF

This resource defines the latitude of the center of the map projection coordinate system. It applies for all values of the *mpProjection* resource except for `LambertConformal`. Its value must lie between -90.0 and +90.0, where -90.0 represents the South Pole and +90.0 represents the North Pole.

V4.1 Status Note 2

Default: 0.0

mpCenterLonF

This resource defines the longitude of the center of the map projection coordinate system. It applies for all values of the *mpProjection* resource except for `LambertConformal`. Its value must lie between -180.0 and +180.0, where both extremes represent the longitude of the International Dateline.

V4.1 Status Note 2

Default: 0.0

mpCenterRotF

Unless the origin lies at one of the poles, this resource defines the angle that the map projection coordinate system is rotated counterclockwise from a vector directed north with origin at the projection center. If the projection center is at the North Pole, the vector points in the direction of *mpCenterLonF*+180.0. If the center is the South Pole, the vector points in the direction of *mpCenterLonF*. *mpCenterRotF* applies for all values of the *mpProjection* resource except for `LambertConformal`.

Default: 0.0

mpLimitMode

This resource of type NhlTMapLimitMode specifies how **MapTransformation** determines the extent of the projected globe that is mapped into the viewport area. It has eight possible settings:

`MaximalArea`

The maximum viewable area of the globe allowed by the projection is mapped into the viewport area.

`LatLon`

MapTransformation maps into the viewport the maximum viewable portion of the area given the projection, the projection center, and bounded by latitudes *mpMinLatF* and *mpMaxLatF* and longitudes *mpMinLonF* and *mpMaxLonF*. If you set boundaries such that the resulting area has zero extent, **MapTransformation** issues a warning and resets *mpLimitMode* to the value NhlMAXIMALAREA. If you set *mpMinLatF* to a value greater than *mpMaxLatF* or *mpMinLonF* to a value greater than *mpMaxLatF*, **MapTransformation** issues a warning and exchanges the values.

`Angles`

For most projections, the area mapped into the viewport is based on the positive angular distance from the center of the projection, in each of the four directions determined by the resources *mpRightAngleF*, *mpLeftAngleF*,*mpBottomAngleF*, and *mpTopAngleF*. For the satellite projection, the angles represent angular deviation from the line of sight from the satellite to the projection center. Angular limits are not supported if *mpProjection* has the value NhlLAMBERTCONFORMAL. Note that left, right, bottom, and top are oriented with respect to the viewplane; therefore, if *mpCenterRotF* has a non-zero value, the angles represent varying combinations of latitudinal and longitudinal components.

Depending on the projection, the maximum allowable value for each of the angle resources varies, as follows:

Angle projection limits:

`Orthographic`
- *mpLeftAngleF* and *mpRightAngleF*: 90.0
- *mpBottomAngleF*and *mpTopAngleF*: 90.0

`Stereographic`
- *mpLeftAngleF* and *mpRightAngleF*: 180.0
- *mpBottomAngleF*and *mpTopAngleF*: 180.0

`LambertEqualArea`
- *mpLeftAngleF* and *mpRightAngleF*: 180.0
- *mpBottomAngleF*and *mpTopAngleF*: 180.0

`Gnomonic`
- *mpLeftAngleF* and *mpRightAngleF*: 85.0
- *mpBottomAngleF*and *mpTopAngleF*: 85.0

`AzimuthalEquidistant`
- *mpLeftAngleF* and *mpRightAngleF*: 180.0
- *mpBottomAngleF*and *mpTopAngleF*: 180.0

`Satellite`
- *mpLeftAngleF* and *mpRightAngleF*: 90.0
- *mpBottomAngleF*and *mpTopAngleF*: 90.0

`Mollweide`
- *mpLeftAngleF* and *mpRightAngleF*: 180.0
- *mpBottomAngleF*and *mpTopAngleF*: 180.0

`Mercator`
- *mpLeftAngleF* and *mpRightAngleF*: 180.0
- *mpBottomAngleF*and *mpTopAngleF*: 85.0

`CylindricalEquidistant`
- *mpLeftAngleF* and *mpRightAngleF*: 180.0
- *mpBottomAngleF*and *mpTopAngleF*: 90.0

`LambertConformal`
- *not supported*

If you attempt to set one of the angle resources to a value outside the range, **MapTransformation** issues a warning and coerces the value to the limiting value. If you set angles in such a way that the resulting map extent has zero area, **MapTransformation** issues a warning and resets *mpLimitMode* to NhlMAXIMALAREA.

V4.1 Status Note 3

`NPC`

`NPC` limits the area based on "Normalized Projection Coordinates," which map the maximal area of the projection into a rectangle bounded by 0.0 and 1.0 horizontally and vertically. Use the resources *mpLeftNPCF*, *mpRightNPCF*, *mpBottomNPCF*, and *mpTopNPCF* to set normalized projection coordinate limits. If you set NPC boundaries such that the resulting area has zero extent, **MapTransformation** issues a warning and resets *mpLimitMode* to the value NhlMAXIMALAREA. If you set *mpLeftNPCF* to a value greater than *mpRightNPCF* or *mpBottomNPCF* to a value greater than *mpTopNPCF*, **MapTransformation** issues a warning and exchanges the values. **MapTransformation** keeps the NPC resources updated to their current value no matter which method is used to set the map limits.

`NDC`

`NDC` limits the area based on the current location of the projected map in Normalized Device Coordinates. Use the resources *mpLeftNDCF*, *mpRightNDCF*, *mpBottomNDCF*, and *mpTopNDCF* to set normalized device coordinate limits. If you set NDC boundaries such that the resulting area has zero extent, **MapTransformation** issues a warning and resets *mpLimitMode* to the value NhlMAXIMALAREA. If you set *mpLeftNDCF* to a value greater than *mpRightNDCF* or *mpBottomNDCF* to a value greater than *mpTopNDCF*, **MapTransformation** issues a warning and exchanges the values. Note that setting these resources usually causes the mapping between the projection space and NDC space to change. Changes to the location of the size or shape of the viewport into which the map is projected also modify the mapping into NDC space. Therefore, at the completion of each **SetValues** call, **MapTransformation** internally sets the value of each *mp...NDCF* resource to the new location of the original projected coordinate: that is, to one of the boundaries of the projected area.

`Corners`

MapTransformation maps into the viewport a rectangular area with one corner at the point *mpLeftCornerLatF* and *mpLeftCornerLonF*, and the opposite corner at the point *mpRightCornerLatF* and *mpRightCornerLonF*. The value of *mpLeftCornerLonF* should be less than *mpRightCornerLonF*. If either point is outside the maximal area given the projection and projection center, a warning is returned and the limit mode defaults to `MaximalArea`.

`Points`

MapTransformation maps into the viewport an area defined by a viewspace rectangle containing the four points defined by *mpLeftPointLatF*, *mpLeftPointLonF*, *mpRightPointLatF*, *mpRightPointLonF*, *mpBottomPointLatF*, *mpBottomPointLonF*, *mpTopPointLatF*, and *mpTopPointLonF*. If any point is outside the maximal area given the projection and projection center, a warning is returned and the limit mode defaults to `MaximalArea` mode.

`Window`

MapTransformation maps into the viewport an area bounded by the window coordinates specified by the resources *mpLeftWindowF*, *mpRightWindowF*, *mpBottomWindowF*, and *mpTopWindowF*. The window coordinates are in an intermediate rectangular coordinate system into which the map coordinates are mapped prior to the mapping into NDC coordinates. Its extents vary depending on the value of *mpProjection*. If you set *mpLeftWindowF* to a value greater than *mpRightWindowF*, or *mpBottomWindowF* to a value greater than *mpTopWindowF*, **MapTransformation** issues a warning and exchanges the values. **MapTransformation** keeps the `Window` resources updated to their current value no matter which method is used to set the map limits.

Default: `MaximalArea`

mpMinLatF

When *mpLimitMode* has the value `LatLon`, this resource specifies the minimum latitude bounding the area to be mapped to the viewport. It should not be less than -90.0.

Default: -90.0

mpMaxLatF

When *mpLimitMode* has the value `LatLon`, this resource specifies the maximum latitude bounding the area to be mapped to the viewport. It should not be greater than 90.0.

Default: 90.0

mpMinLonF

When *mpLimitMode* has the value `LatLon`, this resource specifies the minimum longitude bounding the area to be mapped to the viewport. It may be set to any value in the range -540.0 to 540.0 such that *mpMaxLonF* minus *mpMinLonF* is greater than 0.0 and less than or equal to 360.0.

Default: -180.0

mpMaxLonF

When *mpLimitMode* has the value `LatLon`, this resource specifies the maximum longitude bounding the area to be mapped to the viewport. It may be set to any value in the range -540.0 to 540.0 such that *mpMaxLonF* minus *mpMinLonF* is greater than 0.0 and less than or equal to 360.0.

Default: 180.0

mpRelativeCenterLon

When *mpLimitMode* has the value `LatLon` and *mpRelativeCenterLon* is set True, the resource *mpCenterLonF* becomes an offset in degrees from the longitude halfway between *mpMinLonF* and *mpMaxLonF*. That is, if *mpCenterLonF* is 0.0, the half point longitude will be the center of the projection.

Default: False

mpRelativeCenterLat

When *mpLimitMode* has the value `LatLon` and *mpRelativeCenterLat* is set True, the resource *mpCenterLatF* becomes an offset in degrees from the latitude halfway between *mpMinLatF* and *mpMaxLatF*. That is, if *mpCenterLatF* is 0.0, the half point latitude will be the center of the projection.

Default: False

mpLeftAngleF

When *mpLimitMode* has the value `Angles`, this resource specifies the positive angle in degrees left from the center to the edge of the projection, or, if using a satellite projection, the displacement angle left from the satellite line of sight to the projection center. Note that direction of the line over which the angle is measured remains parallel to the top of the viewport, regardless of the rotation angle applied to the projection. *mpLeftAngleF* has a maximum allowable value that varies depending on the projection.

Default: 80.0

mpRightAngleF

When *mpLimitMode* has the value `Angles`, this resource specifies the positive angle in degrees right from the center to the edge of the projection, or, if using a satellite projection, the displacement angle right from the satellite line of sight to the projection center. Note that direction of the line over which the angle is measured remains parallel to the top of the viewport, regardless of the rotation angle applied to the projection. *mpRightAngleF* has a maximum allowable value that varies depending on the projection.

Default: 80.0

mpBottomAngleF

When *mpLimitMode* has the value `Angles`, this resource specifies the positive angle in degrees down from the center to the edge of the projection, or, if using a satellite projection, the displacement angle down from the satellite line of sight to the projection center. Note that direction of the line over which the angle is measured remains parallel to the left edge of the viewport, regardless of the rotation angle applied to the projection. *mpBottomAngleF* has a maximum allowable value that varies depending on the projection.

Default: 80.0

mpTopAngleF

When *mpLimitMode* has the value `Angles`, this resource specifies the positive angle in degrees up from the center to the edge of the projection, or, if using a satellite projection, the displacement angle up from the satellite line of sight to the projection center. Note that direction of the line over which the angle is measured remains parallel to the left edge of the viewport, regardless of the rotation angle applied to the projection. *mpTopAngleF* has a maximum allowable value that varies depending on the projection.

Default: 80.0

mpLeftNPCF

When *mpLimitMode* has the value `NPC`, this resource specifies the left edge of the limiting rectangle in Normalized Projection Coordinates. *mpLeftNPCF* is constrained to values in the range 0.0 through 1.0. If *mpLeftNPCF* is set to a value greater than *mpRightNPCF*, a warning is issued and the values are exchanged. The value of *mpLeftNPCF* is updated whenever the map limits are set using any of the map limit modes.

Default: 0.0

mpRightNPCF

When *mpLimitMode* has the value `NPC`, this resource specifies the right edge of the limiting rectangle in Normalized Projection Coordinates. *mpRightNPCF* is constrained to values in the range 0.0 through 1.0. If *mpRightNPCF* is set to a value less than *mpLeftNPCF*, a warning is issued and the values are exchanged. The value of *mpRightNPCF* is updated whenever the map limits are set using any of the map limit modes.

Default: 1.0

mpBottomNPCF

When *mpLimitMode* has the value NPC, this resource specifies the bottom edge of the limiting rectangle in Normalized Projection Coordinates. *mpBottomNPCF* is constrained to values in the range 0.0 through 1.0. If *mpBottomNPCF* is set to a value greater than *mpTopNPCF*, a warning is issued and the values are exchanged. The value of *mpBottomNPCF* is updated whenever the map limits are set using any of the map limit modes.

Default: 0.0

mpTopNPCF

When *mpLimitMode* has the value NPC, this resource specifies the top edge of the limiting rectangle in Normalized Projection Coordinates. *mpTopNPCF* is constrained to values in the range 0.0 through 1.0. If *mpTopNPCF* is set to a value less than *mpBottomNPCF*, a warning is issued and the values are exchanged. The value of *mpTopNPCF* is updated whenever the map limits are set using any of the map limit modes.

Default: 1.0

mpLeftNDCF

When *mpLimitMode* has the value NDC, this resource specifies the left edge of the limiting rectangle in Normalized Device Coordinates. If *mpLeftNDCF* is set to a value greater than *mpRightNDCF*, a warning is issued and the values are exchanged. Since the relationship between the map projection area and NDC coordinates changes whenever the map limits are changed, as well as when the viewport area is moved or resized, **MapTransformation** resets the value of *mpLeftNDCF* to the NDC value of the left edge of the projected area at the end of any update to the transformation.

Default: <dynamic>

mpRightNDCF

When *mpLimitMode* has the value NDC, this resource specifies the right edge of the limiting rectangle in Normalized Device Coordinates. If *mpRightNDCF* is set to a value less than *mpLeftNDCF*, a warning is issued and the values are exchanged. Since the relationship between the map projection area and NDC coordinates changes whenever the map limits are changed, as well as when the viewport area is moved or resized, **MapTransformation** resets the value of *mpRightNDCF* to the NDC value of the right edge of the projected area at the end of any update to the transformation.

Default: <dynamic>

mpBottomNDCF

When *mpLimitMode* has the value NDC, this resource specifies the bottom edge of the limiting rectangle in Normalized Device Coordinates. If *mpBottomNDCF* is set to a value greater than *mpRightNDCF*, a warning is issued and the values are exchanged. Since the relationship between the map projection area and NDC coordinates changes whenever the map limits are changed, as well as when the viewport area is moved or resized, **MapTransformation** resets the value of *mpBottomNDCF* to the NDC value of the bottom edge of the projected area at the end of any update to the transformation.

Default: <dynamic>

mpTopNDCF

When *mpLimitMode* has the value NDC, this resource specifies the top edge of the limiting rectangle in Normalized Device Coordinates. If *mpTopNDCF* is set to a value less than *mpBottomNDCF*, a warning is issued and the values are exchanged. Since the relationship between the map projection area and NDC coordinates changes whenever the map limits are changed, as well as when the viewport area is moved or resized, **MapTransformation** resets the value of *mpTopNDCF* to the NDC value of the top edge of the projected area at the end of any update to the transformation.

Default: <dynamic>

mpLeftCornerLatF

When *mpLimitMode* has the value `Corners`, this resource specifies the latitude, in degrees, of the left corner point.

Default: 0.0

mpLeftCornerLonF

When *mpLimitMode* has the value `Corners`, this resource specifies the longitude, in degrees, of the left corner point.

Default: 0.0

mpRightCornerLatF

When *mpLimitMode* has the value `Corners`, this resource specifies the latitude, in degrees, of the right corner point.

Default: 0.0

mpRightCornerLonF

When *mpLimitMode* has the value `Corners`, this resource specifies the longitude, in degrees, of the right corner point.

Default: 0.0

mpLeftPointLatF

When *mpLimitMode* has the value `Points`, this resource specifies the latitude, in degrees, of a point along the left edge of the limiting rectangle.

Default: 0.0

mpLeftPointLonF

When *mpLimitMode* has the value `Points`, this resource specifies the longitude, in degrees, of a point along the left edge of the limiting rectangle.

Default: 0.0

mpRightPointLatF

When *mpLimitMode* has the value `Points`, this resource specifies the latitude, in degrees, of a point along the right edge of the limiting rectangle.

Default: 0.0

mpRightPointLonF

When *mpLimitMode* has the value `Points`, this resource specifies the longitude, in degrees, of a point along the right edge of the limiting rectangle.

Default: 0.0

mpBottomPointLatF

When *mpLimitMode* has the value `Points`, this resource specifies the latitude, in degrees, of a point along the bottom edge of the limiting rectangle.

Default: 0.0

mpBottomPointLonF

When *mpLimitMode* has the value `Points`, this resource specifies the longitude, in degrees, of a point along the bottom edge of the limiting rectangle.

Default: 0.0

mpTopPointLatF

When *mpLimitMode* has the value `Points`, this resource specifies the latitude, in degrees, of a point along the top edge of the limiting rectangle.

Default: 0.0

mpTopPointLonF

When *mpLimitMode* has the value `Points`, this resource specifies the longitude, in degrees, of a point along the top edge of the limiting rectangle.

Default: 0.0

mpLeftWindowF

When *mpLimitMode* has the value `Window`, this resource specifies the value, in window coordinates, of the left edge of the projection.

Default: 0.0

mpRightWindowF

When *mpLimitMode* has the value `Window`, this resource specifies the value, in window coordinates, of the right edge of the projection.

Default: 0.0

mpBottomWindowF

When *mpLimitMode* has the value `Window`, this resource specifies the value, in window coordinates, of the bottom edge of the projection.

Default: 0.0

mpTopWindowF

When *mpLimitMode* has the value `Window`, this resource specifies the value, in window coordinates, of the top edge of the projection.

Default: 0.0

mpLambertParallel1F

When *mpProjection* is set to the value `LambertConformal`, this resource specifies one of the two standard parallels (lines of latitude in degrees) used to define the conic projection. A simpler conic projection is defined when *mpLambertParallel1F* and *mpLambertParallel2F* are equal. Normally, *mpLambertParallel1F* and *mpLambertParallel2F* should not be more than 90.0 degrees apart.

V4.1 Status Note 2

Default: .001

mpLambertParallel2F

When *mpProjection* is set to the value `LambertConformal`, this resource specifies one of the two standard parallels (lines of latitude in degrees) used to define the conic projection. A simpler conic projection is defined when *mpLambertParallel1F* and *mpLambertParallel2F* are equal. Normally, *mpLambertParallel1F* and *mpLambertParallel2F* should not be more than 90.0 degrees apart.

V4.1 Status Note 2

Default: 89.999

mpLambertMeridianF

When *mpProjection* is set to the value `LambertConformal`, this resource specifies the central meridian (line of longitude in degrees) of the projection. This resource has nearly the same meaning that *mpCenterLonF* has for the other projections.

V4.1 Status Note 2

Default: 0.0

mpSatelliteDistF

When *mpProjection* is set to `Satellite`, this resource specifies the distance in multiples of the earth's radius of the satellite at the view origin. If the value is 1.0 or less, the satellite projection is replaced by its limiting case: an orthographic projection.

Default: 1.0

mpSatelliteAngle1F

When *mpProjection* is set to `Satellite` and *mpSatelliteDistF* has a value greater than 1.0, this resource specifies the angle in degrees between a line to the center of the earth and the line of sight (to which the projection plane is perpendicular).

Default: 0.0

mpSatelliteAngle2F

This resource only has an effect when *mpProjection* is set to `Satellite`, *mpSatelliteDistF* has a value greater than 1.0, and *mpSatelliteAngle1F* has a non-zero value. Imagine a compass (the geometrical instrument) with its joint at the position of the satellite, its first leg extending to the center of the earth, and the second pointing in the direction of the line of sight. The second leg is longer than the first, so that it meets a plane through the center of the earth normal to the first leg. The angle in degrees between the two legs is *SatelliteAngle1F*. When *SatelliteAngle2F* has the value 0.0, a line drawn from the first leg of the compass to the second will appear horizontal and directed toward the right. Increasing values of *SatelliteAngle2F* correspond to counterclockwise rotations of the compass, (and hence, the line of sight) from this base position.

Default: 0.0

mpEllipticalBoundary

When the boolean resource *mpEllipticalBoundary* is set True, the map projection area is limited to an ellipse inscribed within the normal rectangular perimeter of the viewport.

Default: False

mpGreatCircleLinesOn

This resource selects between two methods for defining the transformation of lines or edges between defined points of the data space graphics primitive functions, NhlDataPolygon and NhlDataPolyline. If *mpGreatCircleLinesOn* is False, the space between the defined points is treated as if the latitude/longitude coordinate system were cartesian. This method of performing the transformation results in all straight lines between the defined points when *mpProjection* is set to `CylindricalEquidistant`. Another characteristic of this method is that, for any projection, a line between two points at the same latitude is coincident with the line of latitude itself.

If *mpGreatCircleLinesOn* is set True, lines between defined points follow a great circle route, the shortest distance on the surface of the globe between the two points. In this case, lines appear curved on a cylindrical equidistant projection, except for the special cases of lines between points of equal longitude or of lines drawn near the equator. For any projection, lines between points of equal latitude diverge farther and farther from the line of latitude as the distance from the equator increases.

Default: False

NG4.1 Home, Index, Examples, Glossary, Feedback, Ref Contents, Ref WhereAmI?

PlotManager class resource descriptions

pmOverlaySequenceIds

This read-only resource can be used to get an array of object ids comprised of the transforms in the overlay sequence. The elements of the array will be ordered according to the overlay sequence, starting with the base plot.

Default: NULL

pmAnnoViews

This array resource contains the ids of the **View** objects added as annotations by the user to the plot object. If you set this resource, the contents of the existing array are replaced. However, if you add a single annotation using the **NhlAddAnnotation** function, the id of the view is appended to the end of the existing array. When a view becomes an annotation, the **PlotManager** creates an **AnnoManager** object you can use to control the location and/or the size of the object relative to the base plot's viewport or data coordinate space. Once a view has a controlling **AnnoManager**, it can no longer be drawn directly. Instead it is drawn automatically whenever you draw the primary base plot of which it is a plot member.

When you remove a single annotation using the **NhlRemoveAnnotation** function, the remaining **View** ids retain their order within the array, but are moved to fill in the space that had been occupied by the id of the view removed. The result is an array containing one fewer element. If you destroy a **View** object currently used as an annotation, the effect on the *pmAnnoViews* resource is the same as if you had removed it using **NhlRemoveAnnotation**.

Default: NULL

pmAnnoManagers

Each element of this read-only array resource contains the id of the **AnnoManager** object created to control the **View** annotation identified by the corresponding element of the *pmAnnoViews* array. Whenever *pmAnnoViews* is modified, *pmAnnoManagers* is updated to match.

Default: NULL

pmTitleDisplayMode

This resource of type NhlTAnnotationDisplayMode determines whether the plot object displays a **Title** object. It has four settings:

`NoCreate`

If *pmTitleDisplayMode* has the value `NoCreate` when the plot object is created, the **PlotManager** does not create a **Title** object. The plot object will never be able to display a **Title** belonging to itself during its lifetime, and attempts to change the value of *pmTitleDisplayMode* later will fail with an error message. However, as a base plot it still would be able display a **Title** object belonging to an added overlay plot.

`Never`

If *pmTitleDisplayMode* has the value `Never` when the plot object is created, the **PlotManager** does create a **Title** object, but it is not drawn.

`Always`

If *pmTitleDisplayMode* has the value `Always` when the plot object is created, the **PlotManager** creates a **Title** object that will be drawn, assuming it meets all conditions of displayability set by the **Title** object itself.

`Conditional`

If *pmTitleDisplayMode* has the value `Conditional` when the plot object is created, the **PlotManager** creates a **Title** that *may* be drawn, assuming it meets all conditions of displayability set by the **Title** object itself. If the plot object becomes an overlay, the **Title** will be displayed only if no other **Title** objects belonging to other transforms in the overlay sequence have already been displayed.

This resource may be intercepted or disabled by:

- **ContourPlot**
- **StreamlinePlot**
- **VectorPlot**
- **XyPlot**

Default: `NoCreate`

pmTitleZone

This resource specifies the annotation zone used to determine the location of the **Title** object. Note that the **PlotManager** object's treatment of the **Title** object does not completely follow the rules of the PlotManager Location Control Model.

Default: 4

pmTickMarkDisplayMode

This resource of type NhlTAnnotationDisplayMode determines whether the plot object displays a **TickMark** object. It has four settings:

`NoCreate`

If *pmTickMarkDisplayMode* has the value `NoCreate` when the plot object is created, the **PlotManager** does not create a **TickMark** object. The plot object will never be able to display a **TickMark** belonging to itself during its lifetime, and attempts to change the value of *pmTickMarkDisplayMode* later will fail with an error message. However, as a base plot it still would be able display a **TickMark** object belonging to an added overlay plot.

`Never`

If *pmTickMarkDisplayMode* has the value `Never` when the plot object is created, the **PlotManager** does create a **TickMark** object, but it is not drawn.

`Always`

If *pmTickMarkDisplayMode* has the value `Always` when the plot object is created, the **PlotManager** creates a **TickMark** object that will be drawn, assuming it meets all conditions of displayability set by the **TickMark** object itself.

`Conditional`

If *pmTickMarkDisplayMode* has the value `Conditional` when the plot object is created, the **PlotManager** creates a **TickMark** that *may* be drawn, assuming it meets all conditions of displayability set by the **TickMark** object itself. If the plot object becomes an overlay, the **TickMark** will be displayed only if no other **TickMark** objects belonging to other plots in the overlay sequence have already been displayed.

This resource may be intercepted or disabled by:

- **ContourPlot**
- **MapPlot**
- **StreamlinePlot**
- **VectorPlot**
- **XyPlot**

Default: `NoCreate`

pmTickMarkZone

This resource specifies the annotation zone used to determine the location of the **TickMark** object. Since the **TickMark** object is by its nature confined to a bounding box very near the **PlotManager** plot viewport, its zone must be set to 2 or less. Attempts to set the value of *pmTickMarkZone* to a value greater than 2 result in an error message; the resource will be reset to the default value. Note that the **PlotManager** object's treatment of the **TickMark** object does not completely follow the rules of the PlotManager Location Control Model.

Default: 2

pmLabelBarDisplayMode

This resource of type NhlTAnnotationDisplayMode determines whether the plot object displays a **LabelBar** object. It has four settings:

`NoCreate`

If *pmLabelBarDisplayMode* has the value `NoCreate` when the plot object is created, the **PlotManager** does not create a **LabelBar** object. The plot object will never be able to display a **LabelBar** belonging to itself during its lifetime, and attempts to change the value of *pmLabelBarDisplayMode* later will fail with an error message. However, as a base plot it still would be able display a **LabelBar** object belonging to an added overlay plot.

`Never`

If *pmLabelBarDisplayMode* has the value `Never` when the plot object is created, the **PlotManager** does create a **LabelBar** object, but it is not drawn.

`Always`

If *pmLabelBarDisplayMode* has the value `Always` when the plot object is created, the **PlotManager** creates a **LabelBar** object that will be drawn, assuming it meets all conditions of displayability set by the **LabelBar** object itself.

`Conditional`

If *pmLabelBarDisplayMode* has the value `Conditional` when the plot object is created, the **PlotManager** creates a **LabelBar** that *may* be drawn, assuming it meets all conditions of displayability set by the **LabelBar** object itself. If the plot object becomes an overlay, the **LabelBar** will be displayed only if no other **LabelBar** objects belonging to other plots in the overlay sequence have already been displayed.

This resource may be intercepted or disabled by:

- **ContourPlot**
- **StreamlinePlot**
- **VectorPlot**
- **XyPlot**

Default: `NoCreate`

pmLabelBarWidthF

This resource specifies the desired width of the **LabelBar** object in NDC units. Note that the actual width of the **LabelBar** may vary somewhat from the desired width, depending on the setting of certain resources belonging to the **LabelBar**. If not set explicitly, the value of this resource adjusts dynamically in proportion to changes of the viewport width.

Default: 0.15 (for a viewport width of 0.6)

pmLabelBarHeightF

This resource specifies the desired height of the **LabelBar** object in NDC units. Note that the actual height of the **LabelBar** may vary somewhat from the desired height, depending on the setting of certain resources belonging to the **LabelBar**. If not set explicitly, the value of this resource adjusts dynamically in proportion to changes of the viewport height.

Default: 0.6 (for a viewport height of 0.6)

pmLabelBarZone

This resource specifies the annotation zone used to determine the location of the **LabelBar** object. The **PlotManager** manages the location of **LabelBar** object according to the rules of the PlotManager Location Control Model.

Default: 6

pmLabelBarSide

This resource of type NhlTPosition determines where to place the **LabelBar** object in relation to the sides of the plot object's viewport. The PlotManager Location Control Model requires this resource to allow control of the **LabelBar** object in a manner consistent with other annotations. Its value determines a coordinate system origin and the direction of the positional resources *pmLabelBarParallelPosF* and *pmLabelBarOrthogonalPosF*. It also constrains the value of the **LabelBar** object resource *lbJustification* appropriately. There are four settings that behave as follows, unless *pmLabelBarZone* is set to one of the special zones (0 or 1):

`Top`

The **PlotManager** locates the **LabelBar** annotation relative to a line paralleling the top viewport boundary. *pmLabelBarOrthogonalPosF* increases in the direction of increasing NDC Y-Axis values and is constrained to positive values. *pmLabelBarParallelPosF* increases in the direction of increasing NDC X-Axis values. *lbJustification* is constrained to `BottomRight`, `BottomCenter`, or `BottomLeft`.

`Bottom`

The **PlotManager** locates the **LabelBar** annotation relative to a line paralleling the bottom viewport boundary. *pmLabelBarOrthogonalPosF* increases in the direction of decreasing NDC Y-Axis values and is constrained to positive values. *pmLabelBarParallelPosF* increases in the direction of increasing NDC X-Axis values. *lbJustification* is constrained to NhlTOPRIGHT, NhlTOPCENTER, or NhlTOPLEFT.

`Right`

The **PlotManager** locates the **LabelBar** annotation relative to a line paralleling the right viewport boundary. *pmLabelBarOrthogonalPosF* increases in the direction of increasing NDC X-Axis values and is constrained to positive values. *pmLabelBarParallelPosF* increases in the direction of increasing NDC Y-Axis values. *lbJustification* is constrained to `TopLeft`, `CenterLeft`, or `BottomLeft`.

`Left`

The **PlotManager** locates the **LabelBar** annotation relative to a line paralleling the left viewport boundary. *pmLabelBarOrthogonalPosF* increases in the direction of decreasing NDC X-Axis values and is constrained to positive values. *pmLabelBarParallelPosF* increases in the direction of increasing NDC Y-Axis values. *lbJustification* is constrained to `TopRight`, `CenterRight`, or `BottomRight`.

If **pmLabelBarZone** is set to 0, the **PlotManager** locates the **LabelBar** relative to the viewport center. If **pmLabelBarZone** is 1, the direction of the *pmLabelBarOrthogonalPosF* is opposite to the specification given above. Also if the **pmLabelBarZone** is either 0 or 1, *lbJustification* is not constrained, and *pmLabelBarOrthogonalPosF* may take on negative values.

Default: `Bottom`

pmLabelBarParallelPosF

pmLabelBarParallelPosF specifies the coordinate of the base location of the **LabelBar** object annotation parallel to the current *pmLabelBarSide* and directed toward increasing NDC values. The PlotManager Location Control Model requires this resource to allow control of the **LabelBar** object annotation in a manner consistent with other annotations.

Default: 0.5

pmLabelBarOrthogonalPosF

pmLabelBarOrthogonalPosF sets the coordinate of the base location of the **LabelBar** object annotation orthogonal to the current *pmLabelBarSide*. Unless the value of*pmLabelBarZone* is 1, it is directed away from the center of the viewport. The PlotManager Location Control Model requires this resource to allow control of the **LabelBar** object annotation in a manner consistent with other annotations.

Default: 0.02

pmLegendDisplayMode

This resource of type NhlTAnnotationDisplayMode determines whether the plot object displays a **Legend** object. It has four settings:

`NoCreate`

If *pmLegendDisplayMode* has the value `NoCreate` when the plot object is created, the

PlotManager does not create a **Legend** object. The plot object will never be able to display a **Legend** belonging to itself during its lifetime, and attempts to change the value of *pmLegendDisplayMode* later will fail with an error message. However, as a base plot it still would be able display a **Legend** object belonging to an added overlay plot.

`Never`

If *pmLegendDisplayMode* has the value `Never` when the plot object is created, the **PlotManager** does create a **Legend** object, but it is not drawn.

`Always`

If *pmLegendDisplayMode* has the value `Always` when the plot object is created, the **PlotManager** creates a **Legend** object that will be drawn, assuming it meets all conditions of displayability set by the **Legend** object itself.

`Conditional`

If *pmLegendDisplayMode* has the value `Conditional` when the plot object is created, the **PlotManager** creates a **Legend** that *may* be drawn, assuming it meets all conditions of displayability set by the **Legend** object itself. If the plot object becomes an overlay, the **Legend** will be displayed only if no other **Legend** objects belonging to other plots in the overlay sequence have already been displayed.

This resource may be intercepted or disabled by:

- **ContourPlot**
- **StreamlinePlot**
- **VectorPlot**
- **XyPlot**

Default: `NoCreate`

pmLegendWidthF

This resource specifies the desired width of the **Legend** object in NDC units. Note that the actual width of the **Legend** may vary somewhat from the desired width, depending on the setting of certain resources belonging to the **Legend**. If not set explicitly, the value of this resource adjusts dynamically in proportion to changes of the viewport width.

Default: 0.55 (for a viewport width of 0.6)

pmLegendHeightF

This resource specifies the desired height of the **Legend** object in NDC units. Note that the actual height of the **Legend** may vary somewhat from the desired height, depending on the setting of certain resources belonging to the **Legend**. If not set explicitly, the value of this resource adjusts dynamically in proportion to changes of the viewport height.

Default: 0.18 (for a viewport height of 0.6)

pmLegendZone

This resource specifies the annotation zone used to determine the location of the **Legend** object. The **PlotManager** manages the location of **Legend** object according to the rules of the PlotManager Location Control Model.

Default: 7

pmLegendSide

This resource of type NhlTPosition determines where to place the **Legend** object in relation to the sides of the plot object's viewport. The PlotManager Location Control Model requires this resource to allow control of the **Legend** object in a manner consistent with other annotations. Its value determines a coordinate system origin and the direction of the positional resources, *pmLegendParallelPosF* and *pmLegendOrthogonalPosF*. It also constrains the value of the **Legend** object resource *lgJustification* appropriately. There are four settings that behave as follows, unless *pmLegendZone* is set to one of the special zones (0 or 1):

`Top`
: The **PlotManager** locates the **Legend** annotation relative to a line paralleling the top viewport boundary. *pmLegendOrthogonalPosF* increases in the direction of increasing NDC Y-Axis values and is constrained to positive values. *pmLegendParallelPosF* increases in the direction of increasing NDC X-Axis values. *lgJustification* is constrained to `BottomRight`, `BottomCenter`, or `BottomLeft`.

`Bottom`
: The **PlotManager** locates the **Legend** annotation relative to a line paralleling the bottom viewport boundary. *pmLegendOrthogonalPosF* increases in the direction of decreasing NDC Y-Axis values and is constrained to positive values. *pmLegendParallelPosF* increases in the direction of increasing NDC X-Axis values. *lgJustification* is constrained to `TopRight`, `TopCenter`, or `TopLeft`.

`Right`
: The **PlotManager** locates the **Legend** annotation relative to a line paralleling the right viewport boundary. *pmLegendOrthogonalPosF* increases in the direction of increasing NDC X-Axis values and is constrained to positive values. *pmLegendParallelPosF* increases in the direction of increasing NDC Y-Axis values. *lgJustification* is constrained to `TopLeft`, `CenterLeft`, or `BottomLeft`.

`Left`
: The **PlotManager** locates the **Legend** annotation relative to a line paralleling the left viewport boundary. *pmLegendOrthogonalPosF* increases in the direction of decreasing NDC X-Axis values and is constrained to positive values. *pmLegendParallelPosF* increases in the direction of increasing NDC Y-Axis values. *lgJustification* is constrained to `TopRight`, `CenterRight`, or `BottomRight`.

If **pmLegendZone** is set to 0, The **PlotManager** locates the **Legend** relative to the viewport center. If **pmLegendZone** is 1, the direction of the *pmLegendOrthogonalPosF* is opposite to the specification given above. Also if the **pmLegendZone** is either 0 or 1, *lgJustification* is not constrained, and *pmLegendOrthogonalPosF* may take on negative values.

Default: `Bottom`

pmLegendParallelPosF

pmLegendParallelPosF specifies the coordinate of the base location of the **Legend** object annotation parallel to the current *pmLegendSide* and directed toward increasing NDC values. The PlotManager Location Control Model requires this resource to allow control of the **Legend** object annotation in a manner consistent with other annotations.

Default: 0.5

pmLegendOrthogonalPosF

pmLegendOrthogonalPosF sets the coordinate of the base location of the **Legend** object annotation orthogonal to the current *pmLegendSide*. Unless the value of*pmLegendZone* is 1, it is directed away from the center of the viewport. The PlotManager Location Control Model requires this resource to allow control of the **Legend** object annotation in a manner consistent with other annotations.

Default: 0.02

ScalarField class resource descriptions

sfDataArray

This resource specifies a two-dimensional array containing the data comprising a scalar field. There is no default for this resource, and it must be specified in order to create a **ScalarField** object successfully. The faster varying dimension (second dimension in C or NCL, first dimension in Fortran) is assumed to align with the X or horizontal axis of a rectangular coordinate system. However, you may exchange the dimensions of the array by appropriately setting the resource, *sfExchangeDimensions*. Although the **ScalarField** object always converts the data into type NhlTFloat before passing it along to **ScalarField** receiver objects, it accepts data arrays containing data of any of the following types:

- NhlTByte
- NhlTCharacter
- NhlTShort
- NhlTInteger
- NhlTLong
- NhlTFloat
- NhlTDouble

Default: NULL

sfXArray

The values of the *sfXArray* resource sequentially represent the X-Axis coordinate of each successive element along the faster-varying dimension of the data array. If *sfXArray* is set to a non-NULL value, **ScalarField** checks to ensure first that it has the same number of elements as the X-axis dimension of the data array and second that it is monotonically increasing or descreasing. If the input array fails either of these tests, *sfXArray* is reset to its default value of NULL, and the data array is treated as if each element were equally spaced along the X Axis between the values of *sfXCStartV* and *sfXCEndV*. If the input array is valid, **ScalarField** replaces the current value of *sfXCStartV* with the value of the first element of *sfXArray*, and *sfXCEndV* with the value of the last element. You can specify *sfXArray* using a single-dimensioned array of any of the types allowed for *sfDataArray*.

V4.1 Status Note 1

Default: NULL

sfYArray

The values of the *sfYArray* resource sequentially represent the Y-Axis coordinate of each successive element along the slower-varying dimension of the data array. If *sfYArray* is set to a non-NULL value, **ScalarField** checks to ensure first that it has the same number of elements as the Y-axis dimension of the data array and second that it is monotonically increasing or descreasing. If the input array fails either of these tests, *sfYArray* is reset to its default value of

NULL, and the data array is treated as if each element were equally spaced along the Y Axis between the values of *sfYCStartV* and *sfYCEndV*. If the input array is valid, **ScalarField** replaces the current value of *sfYCStartV* with the value of the first element of *sfYArray*, and *sfYCEndV* with the value of the last element. You can specify *sfYArray* using a single-dimensioned array of any of the types allowed for *sfDataArray*.

V4.1 Status Note 1

Default: NULL

sfCopyData

The boolean resource *sfCopyData* allows you to control whether the **ScalarField** object always copies the elements of the *sfDataArray* into a private memory area. If you set *sfCopyData* False, **ScalarField** will not copy the *sfXArray* or the *sfYArray* data and will try to avoid making a copy of the *sfDataArray* data. You are then responsible for ensuring that the data memory locations for these arrays are preserved intact for the lifetime of the **ScalarField**. In return, you save some dynamic memory space. Note that a number of circumstances require **ScalarField** to copy the data regardless. These include supplying data of any type other than NhlTFloat, requesting that the data dimensions be exchanged, or setting a stride value greater than 1 in either the X or the Y dimension. Also, if you specify a data array subset and one or both of the coordinate array resources (*sfXArray* and/or *sfYArray*) is non-NULL, copies are made of the subsetted part of the coordinate array.

Default: True

sfExchangeDimensions

When this boolean resource is set True, **ScalarField** exchanges the dimensions of the data array. Rows become columns and columns become rows; the fast varying dimension becomes the slow varying dimension and vice-versa. Whenever *sfExchangeDimensions* is True, **ScalarField** must copy the data array, regardless of the setting of the *sfCopyData* resource.

Default: False

sfMissingValueV

If you set *sfMissingValueV* to a non-NULL value, **ScalarField** treats occurrences of its value in the data array as representing an undefined value. If left unset, **ScalarField** assumes that all values in the data array are valid. You may set *sfMissingValueV* using any of the data types **ScalarField** accepts for the *sfDataArray* resource.

Default: NULL

sfDataMinV

By default the **ScalarField** object sets *sfDataMinV* to the smallest value in the data array. However, if you set the value of this resource, **ScalarField** does not process the array to find the minimum value. Instead it assumes that the set value is, in fact, the minimum value. You may set *sfDataMinV* using any of the data types **ScalarField** accepts for the *sfDataArray* resource.

Default: <dynamic>

sfDataMaxV

By default the **ScalarField** object sets *sfDataMaxV* to the largest value in the data array. However, if you set the value of this resource, **ScalarField** does not process the array to find the maximum value. Instead it assumes that the set value is, in fact, the maximum value. You may set *sfDataMaxV* using any of the data types **ScalarField** accepts for the *sfDataArray* resource.

Default: <dynamic>

sfXCStartV

This resource specifies the location along the X Axis of the first element of the faster-varying dimension of the data. If the array resource *sfXArray* is set and found to be valid, **ScalarField** overwrites the current value of *sfXCStartV*, replacing it with the value of the first element of *sfXArray*. Otherwise, you can set *sfXCStartV* to any value (presumably meaningful in the context of the your data). If *sfXCStartV* is greater than *sfXCEndV*, the direction of the data locations along the X Axis is opposite the direction of the positive X Axis. If *sfXArray* is NULL or invalid and *sfXCStartV* is not specified, **ScalarField** sets *sfXCStartV* to the value 0. You may set *sfXCStartV* using any of the data types **ScalarField** accepts for the *sfDataArray* resource.

Default: <dynamic>

sfXCEndV

This resource specifies the location along the X Axis of the last element of the faster-varying dimension of the data. If the array resource *sfXArray* is set and found to be valid, **ScalarField** overwrites the current value of *sfXCEndV*, replacing it with the value of the last element of *sfXArray*. Otherwise, you can set *sfXCEndV* to any value (presumably meaningful in the context of the your data). If *sfXCStartV* is greater than *sfXCEndV*, the direction of the data locations along the X Axis is opposite the direction of the positive X Axis. If *sfXArray* is NULL or invalid and *sfXCEndV* is not specified, **ScalarField** sets *sfXCEndV* to the size of the faster-varying data dimension minus one. You may set *sfXCEndV* using any of the data types **ScalarField** accepts for the *sfDataArray* resource.

Default: <dynamic>

sfYCStartV

This resource specifies the location along the Y Axis of the first element of the slower-varying dimension of the data. If the array resource *sfYArray* is set and found to be valid, **ScalarField** overwrites the current value of *sfYCStartV*, replacing it with the value of the first element of *sfYArray*. Otherwise, you can set *sfYCStartV* to any value (presumably meaningful in the context of the your data). If *sfYCStartV* is greater than *sfYCEndV*, the direction of the data locations along the Y Axis is opposite the direction of the positive Y Axis. If *sfYArray* is NULL or invalid and *sfYCStartV* is not specified, **ScalarField** sets *sfYCStartV* to the value 0. You may set *sfYCStartV* using any of the data types **ScalarField** accepts for the *sfDataArray* resource.

Default: <dynamic>

sfYCEndV

This resource specifies the location along the Y Axis of the last element along the slower-varying dimension of the data. If the array resource *sfYArray* is set and found to be valid, **ScalarField** overwrites the current value of *sfYCStartV*, replacing it with the value of the last element of *sfYArray*. Otherwise, you can set *sfYCStartV* to any value (presumably meaningful in the context of the your data). If *sfYCStartV* is greater than *sfYCEndV*, the direction of the data locations along the Y Axis is opposite the direction of the positive Y Axis. If *sfYArray* is NULL or invalid and *sfYCEndV* is not specified, **ScalarField** sets *sfYCEndV* to the size of the slower-varying data dimension minus one. You may set *sfYCEndV* using any of the data types **ScalarField** accepts for the *sfDataArray* resource.

Default: <dynamic>

sfXCStartSubsetV

sfXCStartSubsetV specifies the starting location along the X Axis of a rectangular sub-array of the data array. It should be set to a value within the data coordinate extent as established by the values of *sfXCStartV* and *sfXCEndV*. If it is outside the coordinate extent, **ScalarField** issues a warning and sets it to the value of *sfXCStartV*. If the values of *sfXCStartSubsetV* and *sfXCEndSubsetV* are oppositely directed from the direction established by *sfXCStartV* and *sfXCEndV*, **ScalarField** issues a warning and exchanges their values. When you do not explicitly set a value for *sfXCStartSubsetV*:

- at initialization or when the X-Axis dimension size changes, if *sfXCStartIndex* is not set, *sfXCStartSubsetV* takes the value of *sfXCStartV*;
- else if *sfXCStartIndex* is set, or there is a change to *sfXCStartV*, *sfXCEndV* or *sfXArray*, *sfXCStartSubsetV* takes the data coordinate value indexed by the value of *sfXCStartIndex*;
- else *sfXCStartSubsetV* retains its current value.

Since the data array grid consists of data located at discrete points, the location specified by *sfXCStartSubsetV* might well fall between two data points. In this case, **ScalarField** selects a beginning index for the array subset such that the requested location is included within the subset. Thus the subset array, as delivered to the receiving object, may represent a slightly larger area of the data coordinate space than requested. Get the value of the read-only resource *sfXCActualStartF* in order to find the exact start of the subset along the X Axis. You may set *sfXCSubsetStartV* using any of the data types that **ScalarField** accepts for the *sfDataArray* resource.

Default: <dynamic>

sfXCEndSubsetV

sfXCEndSubsetV specifies the ending location along the X Axis of a rectangular sub-array of the data array. It should be set to a value within the data coordinate extent as established by the values of *sfXCStartV* and *sfXCEndV*. If it is outside the coordinate extent, **ScalarField** issues a warning and sets it to the value of *sfXCEndV*. If the values of *sfXCStartSubsetV* and *sfXCEndSubsetV* are oppositely directed from the direction established by *sfXCStartV* and *sfXCEndV*, **ScalarField** issues a warning and exchanges their values. When you do not explicitly set a value for *sfXCEndSubsetV*:

- at initialization or when the X-Axis dimension size changes, if *sfXCEndIndex* is not set, *sfXCEndSubsetV* takes the value of *sfXCEndV*;
- else if *sfXCEndIndex* is set, or there is a change to *sfXCStartV*, *sfXCEndV* or *sfXArray*, *sfXCEndSubsetV* takes the data coordinate value indexed by the value of *sfXCEndIndex*;
- else *sfXCEndSubsetV* retains its current value.

Since the data array grid consists of data located at discrete points, the location specified by *sfXCEndSubsetV* might well fall between two data points. In this case, **ScalarField** selects an ending index for the array subset such that the requested location is included within the subset. Thus the subset array, as delivered to the receiving object, may represent a slightly larger area of the data coordinate space than requested. Get the value of the read-only resource *sfXCActualEndF* in order to find the exact end of the subset along the X Axis. You may set *sfXCSubsetEndV* using any of the data types that **ScalarField** accepts for the *sfDataArray* resource.

Default: <dynamic>

sfYCStartSubsetV

sfYCStartSubsetV specifies the starting location along the Y Axis of a rectangular sub-array of the data array. It should be set to a value within the data coordinate extent as established by the values of *sfYCStartV* and *sfYCEndV*. If it is outside the coordinate extent, **ScalarField** issues a warning and sets it to the value of *sfYCStartV*. If the values of *sfYCStartSubsetV* and *sfYCEndSubsetV* are oppositely directed from the direction established by *sfYCStartV* and *sfYCEndV*, **ScalarField** issues a warning and exchanges their values. When you do not explicitly set a value for *sfYCStartSubsetV*:

- at initialization or when the Y-Axis dimension size changes, if *sfYCStartIndex* is also set, *sfYCStartSubsetV* takes the value of *sfYCStartV*;
- else if *sfYCStartIndex* is set, or there is a change to *sfYCStartV*, *sfYCEndV* or *sfYArray*, *sfYCStartSubsetV* takes the data coordinate value indexed by the value of *sfYCStartIndex*;
- else *sfYCStartSubsetV* retains its current value.

Since the data array grid consists of data located at discrete points, the location specified by *sfYCStartSubsetV* might well fall between two data points. In this case, **ScalarField** selects a beginning index for the array subset such that the requested location is included within the subset. Thus the subset array, as delivered to the receiving object, may represent a slightly larger area of the data coordinate space than requested. Get the value of the read-only resource *sfYCActualStartF* in order to find the exact start of the subset along the Y Axis. You may set *sfYCSubsetStartV* using any of the data types that **ScalarField** accepts for the *sfDataArray* resource.

Default: <dynamic>

sfYCEndSubsetV

sfYCEndSubsetV specifies the ending location along the Y Axis of a rectangular sub-array of the data array. It should be set to a value within the data coordinate extent as established by the values of *sfYCStartV* and *sfYCEndV*. If it is outside the coordinate extent, **ScalarField** issues a warning and sets it to the value of *sfYCEndV*. If the values of *sfYCStartSubsetV* and *sfYCEndSubsetV* are oppositely directed from the direction established by *sfYCStartV* and *sfYCEndV*, **ScalarField** issues a warning and exchanges their values. When you do not explicitly set a value for *sfYCEndSubsetV*:

- at initialization or when the Y-Axis dimension size changes, if *sfYCEndIndex* is not set, *sfYCEndSubsetV* takes the value of *sfYCEndV*;
- else if *sfYCEndIndex* is set, or there is a change to *sfYCStartV*, *sfYCEndV* or *sfYArray*, *sfYCEndSubsetV* takes the data coordinate value indexed by the value of *sfYCEndIndex*;
- else *sfYCEndSubsetV* retains its current value.

Since the data array grid consists of data located at discrete points, the location specified by *sfYCEndSubsetV* might well fall between two data points. In this case, **ScalarField** selects an ending index for the array subset such that the requested location is included within the subset. Thus the subset array, as delivered to the receiving object, may represent a slightly larger area of the data coordinate space than requested. Get the value of the read-only resource *sfYCActualEndF* in order to find the exact end of the subset along the Y Axis. You may set *sfYCSubsetEndV* using any of the data types that **ScalarField** accepts for the *sfDataArray* resource.

Default: <dynamic>

sfXCStartIndex

This resource specifies the smaller of two indexes along the X-Axis dimension bounding a rectangular sub-array of the data array. Explicitly setting *sfXCStartSubsetV* overrides any value set for *sfXCStartIndex* and causes it to be set to the array index value associated with the closest data point such that the requested data coordinate location is included within the subset. Otherwise, when you set *sfXCStartIndex*, the data coordinate value indexed by its value is placed into *sfXCStartSubsetV*. *sfXCStartIndex* retains its value when the data array changes, unless the X-Axis dimension size also changes. In this case, assuming neither *sfXCStartSubsetV* nor *sfXCStartIndex* is explicitly set at the same time, *sfXCStartIndex* is set to 0.

Default: <dynamic>

sfXCEndIndex

This resource specifies the larger of two indexes along the X-Axis dimension bounding a rectangular sub-array of the data array. Explicitly setting *sfXCEndSubsetV* overrides any value set for *sfXCEndIndex* and causes it to be set to the array index value associated with the closest data point such that the requested data coordinate location is included within the subset. Otherwise, when you set *sfXCEndIndex*, the data coordinate value indexed by its value is placed into *sfXCEndSubsetV*. *sfXCEndIndex* retains its value when the data array changes, unless the X-Axis dimension size also changes. In this case, assuming neither *sfXCEndSubsetV* nor *sfXCEndIndex* is explicitly set at the same time, *sfXCEndIndex* is set to the size of the X-Axis dimension minus one.

Default: <dynamic>

sfYCStartIndex

This resource specifies the smaller of two indexes along the Y-Axis dimension bounding a rectangular sub-array of the data array. Explicitly setting *sfYCStartSubsetV* overrides any value set for *sfYCStartIndex*, and causes it to be set to the array index value associated with the closest data point such that the requested data coordinate location is included within the subset. Otherwise, when you set *sfYCStartIndex*, the data coordinate value indexed by its value is placed into *sfYCStartSubsetV*. *sfYCStartIndex* retains its value when the data array changes, unless the Y-Axis dimension size also changes. In this case, assuming neither *sfYCStartSubsetV* nor *sfYCStartIndex* is explicitly set at the same time, *sfYCStartIndex* is set to 0.

Default: <dynamic>

sfYCEndIndex

This resource specifies the larger of two indexes along the Y-Axis dimension bounding a rectangular sub-array of the data array. Explicitly setting *sfYCEndSubsetV* overrides any value set for *sfYCEndIndex*, and causes it to be set to the array index value associated with the closest data point such that the requested data coordinate location is included within the subset. Otherwise, when you set *sfYCEndIndex*, the data coordinate value indexed by its value is placed into *sfYCEndSubsetV*. *sfYCEndIndex* retains its value when the data array changes, unless the Y-Axis dimension size also changes. In this case, assuming neither *sfYCEndSubsetV* nor *sfYCEndIndex* is explicitly set at the same time, *sfYCEndIndex* is set to the size of the Y-Axis dimension minus one.

Default: <dynamic>

sfXCActualStartF

When a subset of the data is specified by location in data coordinate space using *sfXCStartSubsetV*, this location might well fall between two data points. In this case, **ScalarField** selects a beginning index for the array subset such that the requested location is included within the subset. Thus the subset array, as delivered to the receiving object, may represent a slightly larger area of the data coordinate space than requested. The read-only resource *sfXCActualStartF* always contains the actual starting location of the data subset along the X Axis.

Default: <dynamic>

sfXCActualEndF

When a subset of the data is specified by location in data coordinate space using *sfXCEndSubsetV*, this location might well fall between two data points. In this case, **ScalarField** selects an ending index for the array subset such that the requested location is included within the subset. Thus the subset array, as delivered to the receiving object, may represent a slightly larger area of the data coordinate space than requested. The read-only resource *sfXCActualEndF* always contains the actual ending location of the data subset along the X Axis.

Default: <dynamic>

sfYCActualStartF

When a subset of the data is specified by location in data coordinate space using *sfYCStartSubsetV*, this location might well fall between two data points. In this case, **ScalarField** selects a beginning index for the array subset such that the requested location is included within the subset. Thus the subset array, as delivered to the receiving object, may represent a slightly larger area of the data coordinate space than requested. The read-only resource *sfYCActualStartF* always contains the actual starting location of the data subset along the Y Axis.

Default: <dynamic>

sfYCActualEndF

When a subset of the data is specified by location in data coordinate space using *sfYCEndSubsetV*, this location might well fall between two data points. In this case, **ScalarField** selects an ending index for the array subset such that the requested location is included within the subset. Thus the subset array, as delivered to the receiving object, may represent a slightly larger area of the data coordinate space than requested. The read-only resource *sfYCActualEndF* always contains the actual ending location of the data subset along the Y Axis.

Default: <dynamic>

sfXCStride

When *sfXCStride* has a value greater than one, the **ScalarField** object creates a new array containing only columns indexed along the X dimension by successive multiples (starting with 0) of the *sfXCStride* value added to the starting index. The starting index is based on the current state of the applicable subsetting resources; it is 0 if no array subsetting is in effect. The resulting array is passed to the receiving plot object.

Note that if ending index minus starting index is not an exact multiple of the *sfXCStride* value, then the data coordinate range may be reduced slightly.

Default: 1

sfYCStride

When *sfYCStride* has a value greater than one, the **ScalarField** object creates a new array containing only rows indexed along the Y dimension by successive multiples (starting with 0) of the *sfYCStride* value added to the starting index. The starting index is based on the current state of the applicable subsetting resources; it is 0 if no array subsetting is in effect. The resulting array is passed to the receiving plot object.

Note that if ending index minus starting index is not an exact multiple of the *sfYCStride* value, then the data coordinate range may be reduced slightly.

Default: 1

StreamlinePlot class resource descriptions

stVectorFieldData

Specifies the id of a **VectorField** data object. There is no default for this resource; it is the only resource that must be set in order for the **StreamlinePlot** object to draw a plot. You *may* create a **StreamlinePlot** object without setting the *stVectorFieldData* resource, and auxiliary annotations such as tick marks and titles may appear as the result of a draw, but the **StreamlinePlot** itself will not show up. The **VectorField** object can provide either regularly spaced or irregular rectangular gridded data to the **StreamlinePlot** object, and the VectorField object provides a number of resources for controlling the ingestion of the raw data.

Default: <none>

stStreamlineDrawOrder

This resource of type NhlTDrawOrder determines when the streamlines are drawn relative to the drawing of other elements of a composite plot. There are three choices:

`PreDraw`
: Draw the streamlines before the standard draw phase; the arrows will be overlaid by any subsequently drawn elements.

`Draw`
: Draw the streamlines during the standard draw phase; the arrows will overlay any elements drawn during the predraw phase, but will underlie elements drawn during the postdraw phase.

`PostDraw`
: Draw the streamlines after the standard draw; the arrows will overlay any elements drawn during the predraw and draw phases.

Default: `Draw`

stMapDirection

This resource controls whether the streamline direction is mapped into the same coordinate space as the location of the vector data grid point, or whether it is rendered in a locally uniform Cartesian coordinate space. This resource has an effect whenever a non-uniform transformation is in effect. These include most of the **MapTransformation** transformations and **IrregularTransformation** transformations. Also included are logarithmic transformations provided by the **LogLinTransformation** and even linear transformations when the X and Y unit size is different.

Default: True

stStepSizeF

This resource specifies, in NDC, the basic step size used to build the streamlines. Where the flow direction changes rapidly, the step size is scaled down from the basic step size in order to retain accuracy. Smaller step sizes produce a more accurate plot at the cost of a longer drawing time. At initialization or any time *stStepSizeF* is given a value less than or equal to 0.0, **StreamlinePlot** calculates a value for *stStepSizeF* based on the NDC size of an individual grid cell assuming a linear transformation from data to NDC space. Otherwise, when *stStepSizeF* is not set explicitly, **StreamlinePlot** makes proportional adjustments to its value in response to changes in the assumed NDC size of the grid cell (resulting from changes to the viewport size and/or the number of elements in the data grid).

Default: <dynamic>

stMinLineLengthF

THIS RESOURCE IS NOT YET IMPLEMENTED.

Default:

stMinLineSpacingF

This resource specifies the minimum distance in NDC that a streamline in progress is allowed to approach existing streamlines before being terminated. In general, giving this value a larger number increases the distance between streamlines, and has a tendency to create more, but shorter streamlines. The spacing is only checked at intervals, so streamlines can sometimes approach closer than the specified distance. The *stCrossoverCheckCount* resource determines the frequency with with the check is performed. At initialization or any time *stMinLineSpacingF* is given a value less than or equal to 0.0, **StreamlinePlot** calculates a value for *stMinLineSpacingF* based on the NDC size of an individual grid cell assuming a linear transformation from data to NDC space. Otherwise, when *stMinLineSpacingF* is not set explicitly, **StreamlinePlot** makes proportional adjustments to its value in response to changes in the assumed NDC size of the grid cell (resulting from changes to the viewport size and/or the number of elements in the data grid).

Default: <dynamic>

stMinStepFactorF

This resource specifies a multiplying factor for the *stStepSizeF* resource that determines the minimum amount the streamline must have grown each time the stream progress is checked in order not to be terminated. The stream progress is checked each *stLengthCheckCount* iterations through the stream-building loop. Points of convergence or divergence typically cause stream growth to diminish and the streamline eventually to be terminated.

Default: 2.0

stLengthCheckCount

This resource specifies the number of iterations through the streamline-building loop between each check of the streamline growth. If the distance between the current position of the streamline and the position saved at the time of the previous check is less than a minimum amount, defined as *stMinStepFactorF* times *stStepSizeF*, then the current streamline is terminated and a new one begun if possible.

Default: 35

stCrossoverCheckCount

This resource specifies the number of iterations through the streamline-building loop between checks for streamline crossover, that is, one streamline growing closer than *stMinLineSpacingF* to previously created streamlines. A negative value causes Streamlines to check for crossover only when a new data grid box is entered. At each crossover check, the current streamline position is compared with a sampling of previous streamline positions retained in an internal circular list.

This list is currently fixed to a length of 750. Since up to this number of comparisons are performed at each crossover check, the frequency with which these checks are performed can have a noticeable impact on the drawing time.

Default: -1

stLineStartStride

StreamlinePlot can start only one streamline within each data grid box. This resource allows you to define a sparser streamline plot by allowing only certain grid boxes to be eligible for starting a streamline. If *stLineStartStride* is set to 3, for instance, only every third grid box (along each axis direction) will be eligible for starting a streamline. A streamline will actually be started within an eligible grid box only if no previously drawn streamline passes through the grid box.

Default: 2

stArrowStartStride

StreamlinePlot draws directional arrows based on the data grid location. No more than one directional arrow can be drawn within each grid box. This resource allows you to draw fewer directional arrows by skipping grid boxes. If *stArrowStartStride* is set to 3, for instance, only every third grid box (along each axis direction) will be eligible for a directional arrow. Of course, the arrow will appear only if a streamline is actually drawn within the area of the grid box.

Default: 2

stArrowLengthF

This resource specifies, in units of NDC, the length of the lines used to draw the directional arrows that appear periodically along the length of the streamlines. At initialization or any time *stArrowLengthF* is given a value less than or equal to 0.0, **StreamlinePlot** calculates a value for *stArrowLengthF* based on the NDC size of an individual grid cell assuming a linear transformation from data to NDC space. Otherwise, when *stArrowLengthF* is not set explicitly, **StreamlinePlot** makes proportional adjustments to its value in response to changes in the assumed NDC size of the grid cell (resulting from changes to the viewport size and/or the number of elements in the data grid).

Default: <dynamic>

stMinArrowSpacingF

This resource specifies an NDC minimum distance for adjacent directional arrowheads along a single streamline. If the data grid is transformed in such a way that adjacent grid cells become very close in NDC space, as for instance in many map projections near the poles, you can use this resource to help reduce the otherwise cluttered appearance of these regions of the plot. If *stMinArrowSpacingF* is less than or equal to 0.0, then no arrowheads are eliminated. When *stMinArrowSpacingF* is not set explcitly, **StreamlinePlot** adjusts its value in proportion to changes to *vpWidthF*.

Default: 0.0

stLineThicknessF

This resource sets the thickness of the lines used to draw the streamlines and also the directional arrows. The value acts as a multiplier of a (device-dependent) unit thickness.

Default: 1.0

stLineColor

This resource of type NhlTColorIndex sets the color index used to draw the streamlines.

Default: `Foreground`

stNoDataLabelOn

This boolean resource, when set True, causes a label to appear when **StreamlinePlot** is drawn without any data having been provided. Except for the label string, all attributes of this label, including its position, are set using resources belonging to the zero field label. When set False, no such label appears.

Default: True

stNoDataLabelString

This resource contains the string that appears in the 'No Data' label if you draw a **StreamlinePlot** object without providing any data. No substitution substrings are allowed in this label, since all the substitutions depend on data being available. Except for the boolean switch that turns it on and off, all attributes of this label, including its position, are set using resources belonging to the zero field label.

Default: "NO STREAMLINE DATA"

stZeroFLabelOn

The **StreamlinePlot** object draws a zero field label annotation only when *stZeroFLabelOn* is set True and the **ScalarField** data are determined to consist entirely of zero magnitude vectors within the limits of the available precision.

Default: True

stZeroFLabelString

Specifies the string to use when drawing a zero field label. The string may contain function codes.

Default: "ZERO FIELD"

stZeroFLabelTextDirection

This resource of type NhlTTextDirection specifies the direction of the text in the zero field label. There are two choices:

`Down`

Each character is placed below the previous character in the text string.

`Across`

Each character is placed to the right of the previous character in the text string.

These descriptions apply before rotation due to *stZeroFieldAngleF*.

Default: `Across`

stZeroFLabelFontHeightF

This resource controls the height in NDC of characters used in the text of the zero field label. The character width scales proportionally, unless you also modify the aspect ratio using the *stZeroFLabelFontAspectF* resource. The zero label text height scales with changes to the viewport width, unless you explicitly set *stZeroFLabelFontHeightF* during the same call.

Default: <dynamic>

stZeroFLabelFont

This resource of type NhlTFont specifies the font used to render the zero field label.

Default: 0

stZeroFLabelFontColor

This resource specifies the HLU color index used to render zero field label text.

Default: True

stZeroFLabelFontAspectF

This resource determines the shape of the zero field label characters. Values increasing from 1.0 result in characters that appear thinner. Values decreasing from 1.0 make the characters appear wider. Values less than or equal to 0.0 result in a WARNING message and a restoration of the default value.

Default: 1.3125

stZeroFLabelFontThicknessF

Sets the thickness of the line used to draw zero field label text. The value acts as a multiplier of a (device-dependent) unit thickness. This resource is ignored when the *stZeroFLabelFont* specifies a filled font (font indexes 21-22, 25-26, 29-30, and 33-37).

Default: 1.0

stZeroFLabelFontQuality

This resource of type NhlTFontQuality determines the quality of the font used to draw the **StreamlinePlot** zero field label.

Default: `High`

stZeroFLabelConstantSpacingF

Normally when *stZeroFFontQuality* is set to `High`, the **StreamlinePlot** object writes zero field label text with proportional spacing. Setting the *stZeroFLabelConstantSpacingF* to a value greater than 0.0 overrides this behavior. Instead, the distance from the start of one character to the next is computed by multiplying a single standard character width by the value of *stZeroFConstantSpacingF*. When *stZeroFConstantSpacingF* has a value between 0.0 and 1.0, characters will overlap. A value of 1.0 implies, on average, no space between characters, while values increasing from 1.0 cause the space between characters to grow. This parameter is ignored when *stZeroFLabelFontQuality* is not `High`. Values less than 0.0 result in an error and are replaced with the default value.

Default: 0.0

stZeroFLabelAngleF

This resource specifies the angle, in degrees, of the zero field label text and its surrounding box.

Default: 0.0

stZeroFLabelFuncCode

This resource of type NhlTCharacter sets the function code character that the low level utilities will use when parsing the zero field label string.

Default: **:**

stZeroFLabelBackgroundColor

This resource sets the background color used to fill the box surrounding the zero field label. If you do not want the box to be filled at all, set *stZeroFLabelBackgroundColor* to `Transparent` (-1).

Default: `Background`

stZeroFLabelPerimOn

stZeroFLabelPerimOn is a boolean resource that determines whether **StreamlinePlot** will draw an outline around the perimeter of the box surrounding the zero field label. If set False, no outline will be drawn.

Default: True

stZeroFLabelPerimSpaceF

stZeroFLabelPerimSpaceF determines the spacing or margin between the text of the zero field label and the edge of the zero field label box as a fraction of the current label text height.

Default: 0.33

stZeroFLabelPerimColor

This resource sets the HLU color index used to draw the perimeter of the zero field label box.

Default: `Foreground`

stZeroFLabelPerimThicknessF

This resource determines the thickness of the perimeter line around the zero field label box. The value acts as a multiplier of a (device-dependent) unit thickness.

Default: 1.0

stZeroFLabelZone

StreamlinePlot implements the zero field label as an embedded annotation. *stZeroFLabelZone* specifies the **PlotManager** zone assigned to the zero field annotation. The PlotManager Location Control Model requires this resource to allow control of the zero field label consistent with other annotations. If *stZeroFLabelZone* is set to 0, the positional origin is the center of the plot viewport; otherwise it is on or outside one of the sides of the viewport.

If you create a **StreamlinePlot** object without an active **PlotManager** by setting *tfPlotManagerOn* False, then **StreamlinePlot** manages the zero field annotation by itself. In this case, the *stZeroFLabelZone* resource is not as meaningful.

Default: 0

stZeroFLabelSide

This resource of type NhlTPosition determines where to place the zero field annotation in relation to the sides of the plot object. The PlotManager Location Control Model requires this resource to allow control of the zero field label consistent with other annotations. Its value determines a coordinate system origin and the direction of the positional resources, *stZeroFLabelParallelPosF* and *stZeroFLabelOrthogonalPosF*. It also constrains the value of the *stZeroFLabelJust* appropriately. There are four settings, which behave as follows, unless *stZeroFLabelZone* is set to one of the special zones (0 or 1):

`Top`

The **PlotManager** locates the zero field label annotation relative to a line paralleling the top viewport boundary. *stZeroFLabelOrthogonalPosF* increases in the direction of increasing NDC Y-Axis values and is constrained to positive values. *stZeroFLabelParallelPosF* increases in the direction of increasing NDC X-Axis values. *stZeroFLabelJust* is constrained to `BottomRight`, `BottomCenter`, or `BottomLeft`.

`Bottom`

The **PlotManager** locates the zero field label annotation relative to a line paralleling the bottom viewport boundary. *stZeroFLabelOrthogonalPosF* increases in the direction of decreasing NDC Y-Axis values and is constrained to positive values. *stZeroFLabelParallelPosF* increases in the direction of increasing NDC X-Axis values. *stZeroFLabelJust* is constrained to `TopRight`, `TopCenter`, or `TopLeft`.

`Right`

The **PlotManager** locates the zero field label annotation relative to a line paralleling the right viewport boundary. *stZeroFLabelOrthogonalPosF* increases in the direction of increasing NDC X-Axis values and is constrained to positive values. *stZeroFLabelParallelPosF* increases in the direction of increasing NDC Y-Axis values. *stZeroFLabelJust* is constrained to `TopLeft`, `CenterLeft`, or `BottomLeft`.

`Left`

The **PlotManager** locates the zero field label annotation relative to a line paralleling the left viewport boundary. *stZeroFLabelOrthogonalPosF* increases in the direction of decreasing NDC X-Axis values and is constrained to positive values. *stZeroFLabelParallelPosF* increases in the direction of increasing NDC Y-Axis values. *stZeroFLabelJust* is constrained to `TopRight`, `CenterRight`, or `BottomRight`.

If **stZeroFLabelZone** is set to 0, The **PlotManager** locates the zero field label annotation relative to the viewport center. If **stZeroFLabelZone** is 1, the direction of the *stZeroFLabelOrthogonalPosF* is opposite to the specification given above. Also if the **stZeroFLabelZone** is either 0 or 1, *stZeroFLabelJust* is not constrained, and *stZeroFLabelOrthogonalPosF* may take on negative values.

Default: `Bottom`

stZeroFLabelParallelPosF

stZeroFLabelParallelPosF specifies the coordinate of the base location of the zero field label annotation parallel to the current *stZeroFLabelSide* and directed toward increasing NDC values. The PlotManager Location Control Model requires this resource to allow control of the zero field label consistent with other annotations.

Default: 0.0

stZeroFLabelOrthogonalPosF

stZeroFLabelOrthogonalPosF sets the coordinate of the base location of the zero field label annotation orthogonal to the current *stZeroFLabelSide* and directed away from the center of the viewport. The PlotManager Location Control Model requires this resource to allow control of the zero field label consistent with other annotations.

Default: 0.0

stZeroFLabelJust

This resource of type NhlTJustification, after constraint to an appropriate value based on *stZeroFLabelSide*, sets the justification of the zero field label annotation with respect to its base location. The PlotManager Location Control Model requires this resource to allow control of the zero field label consistent with other annotations.

Default: `CenterCenter`

Transform class resource descriptions

tfDoNDCOverlay

Ordinarily, the data space of a **transform** added as an overlay is transformed into the data space of the base plot. If *tfDoNDCOverlay* is set True, however, this transformation does not take place. Instead, the **transform** is aligned in NDC space with the base plot. Its viewport is sized and positioned to coincide with the base plot's viewport. As with normal overlays, the base plot still assumes responsibility for drawing the **transform** and managing its annotations.

Default: False

tfPlotManagerOn

This boolean resource specifies whether a **transform** should instantiate a **PlotManager** composite class member. It is a create-only resource; if a **plotmanager** is not instantiated when the **transform** is created, it cannot be instantiated later. If a **plotmanager** is activated, the **transform** becomes a plot object, allowing it to act as a base plot for annotations and overlays. Otherwise, it is called a simple **transform**. The primary reason to create a simple **transform** would be to conserve memory and improve performance.

Default: True

Title class resource descriptions

tiDeltaF

Sets the default offset for all of the titles. The absolute value of this resource multiplied by the appropriate *ti...FontHeightF* resource value gives the amount, in NDC, that the titles will be offset perpendicularly from the viewport boundary in the direction opposite the viewport center. After the effect of this offset is calculated, if any part of a title is left overlapping the viewport, it is adjusted farther outward such that no overlap remains. If *tiMainOffsetYF*, *tiXAxisOffsetYF*, or *tiYAxisOffsetXF* are non-zero, they specify an additional offset applied unconditionally from the adjusted position in the same or the opposite direction.

Default: 1.5

tiUseMainAttributes

When this boolean resource is set True, the **Title** object copies the value of Main title resources that set most of the text attributes to the corresponding X-Axis and Y-Axis title resources. The resources copied include *tiMainFont*, *tiMainFontHeightF*, *tiMainFontAspectF*, *tiMainConstantSpacingF*, *tiMainFontThicknessF*, *tiMainFuncCode*, *tiMainFontColor*, *tiMainFont*, *tiMainJust*, and *tiMainFontQuality*.

Default: False

---

tiMainOn

Setting this boolean resource True causes the Main title to appear. Setting it False turns the Main title off. By default, *tiMainOn* will be set True if and only if you explicitly set *tiMainString*.

Default: <dynamic>

tiMainString

Sets the string to use as the Main title. The string may contain Text Function Codes that allow you to control typographical attributes such as subscripting and superscripting, change fonts within a string, embed newlines, etc.

Default: "Main"

tiMainSide

This resource of type NhlTPosition specifies the side of the viewport relative to which the justification point of the Main title is located. There are two choices:

`Top`
: Locate the justification point of the Main title relative to the top edge of the viewport.

`Bottom`
: Locate the justification point of the Main title relative to the bottom edge of the viewport.

If you set *tiMainPosition* to `Left`, `Center`, or `Right` **Title** issues a warning message and reverts to its default value.

Default: `Top`

tiMainPosition

This resource of type NhlTPosition specifies the base horizontal location of the justification point of the Main title with respect to the viewport boundaries. There are three choices:

`Left`
: Locate the base position of the justification point of the Main title along the line of the left edge of the viewport.

`Center`
: Locate the base position of the justification point of the Main title along the line of the horizontal center of the viewport.

`Right`
: Locate the base position of justification point of the Main title along the line of the right edge of the viewport.

From any of these base positions, you can displace the Main title left or right by setting the resource *tiMainOffsetXF*. If you set *tiMainPosition* to `Bottom` or `Top` **Title** issues a warning message and reverts to the default value, `Center`.

Default: `Center`

tiMainOffsetXF

Sets an offset in NDC coordinates by which the Main title will be displaced in a direction parallel to the X Axis from its base location. This value does *not* scale in response to changes in the viewport size.

Default: 0.0

tiMainOffsetYF

Sets a positive or negative offset in NDC coordinates by which the Main title will be displaced in a direction parallel to the Y Axis from its base location. This offset is applied after any offset due to *tiDeltaF* and the adjustment performed to prevent the Main title from intruding into the viewport (or into the X-Axis title). Therefore, this resource allows you to displace the title any distance away from the viewport or into its interior. The positive direction coincides with that of the NDC Y Axis. Its value does *not* scale in response to changes in the viewport size.

Default: 0.0

tiMainJust

This resource of type NhlTJustification sets the Justification point of the Main title.

Default: `CenterCenter`

tiMainFont

This resource of type NhlTFont sets the font index for the Main title.

Default: 0

tiMainFontColor

Sets the font color index for the Main title.

Default: 1

tiMainFontHeightF

Sets the font height in NDC coordinates of the Main title. This value scales proportionally to changes in the width of the viewport.

This resource may be intercepted or disabled by:

- **PlotManager**

Default: 0.025

tiMainFontAspectF

This resource sets the aspect ratio, defined as the height divided by the width of a reference character box within which all characters of the font will fit. Its value determines the width of characters in the Main title relative to *tiMainFontHeightF*. If given the value 1.0, wide characters (such as 'M'or 'W') will be approximately square in shape. Values increasing from 1.0 result in thinner characters, while values decreasing from 1.0 result in wider characters.

Default: 1.3125

tiMainConstantSpacingF

If this resource has the value 0.0, the characters in the Main title will be proportionally spaced. If it has a value greater than 0.0, the distance from the beginning of one character to the beginning of the next character will be constant. The value of *tiMainConstantSpacingF* sets this constant distance as a multiple of the width of a reference character box within which all characters of the font will fit. If *tiMainConstantSpacingF* has the value 1.0, the constant distance will cause there to be almost no space between two wide characters (such as 'M' or 'W') placed next to each other. As *tiMainConstantSpacingF* decreases from 1.0, characters will begin to overlap. As it increases from 1.0, characters become further separated from each other. Values less than 0.0 result in an error and are replaced with the default value.

Default: 0.0

tiMainFontThicknessF

Sets the thickness of the lines used to draw the characters of the Main title. The value acts as a multiplier of a (device-dependent) unit thickness. This resource is ignored when the *tiMainFont* specifies a filled font (font indexes 21-22, 25-26, 29-30, and 33-37).

Default: 1.0

tiMainFuncCode

Specifies the character used to delimit Text Function Codes that may be embedded in the Main title.

Default: **:**

tiMainFontQuality

Sets the font quality for the Main title.

Default: `High`

tiMainAngleF

Angle of counterclockwise rotation of the Main title.

Default: 0.00

tiMainDirection

This resource of type NhlTTextDirection specifies the direction of the text used for the Main title. There are two choices:

`Down`
: Each character is placed below the previous character in the text string.

`Across`
: Each character is placed to the right of the previous character in the text string.

These descriptions apply before any rotation due to *tiMainAngleF* is applied to the Main title.

Default: `Across`

---

tiXAxisOn

Setting this boolean resource True causes the X-Axis title to appear. Setting it False turns the X-Axis title off. By default, *tiXAxisOn* will be set True if and only if you explicitly set *tiXAxisString*.

Default: <dynamic>

tiXAxisString

Sets the string to use as the X-Axis title. The string may contain Text Function Codes that allow you to control typographical attributes such as subscripting and superscripting, change fonts within a string, embed newlines, etc.

Default: "XAxis"

tiXAxisSide

This resource of type NhlTPosition specifies the side of the viewport relative to which the justification point of the X-Axis title is located. There are two choices:

`Top`
: Locate the justification point of the X-Axis title relative to the top edge of the viewport.

`Bottom`
: Locate the justification point of the X-Axis title relative to the bottom edge of the viewport.

If you set *tiXAxisPosition* to `Left`, `Center`, or `Right` **Title** issues a warning message and reverts to the default value, `Bottom`.

Default: `Bottom`

tiXAxisPosition

This resource of type NhlTPosition specifies the base horizontal location of the justification point of the X-Axis title with respect to the viewport boundaries. There are three choices:

`Left`
: Locate the base position of the justification point of the X-Axis title along the line of the left edge of the viewport.

`Center`
: Locate the base position of the justification point of the X-Axis title along the line of the horizontal center of the viewport.

`Right`
: Locate the base position of the justification point of the X-Axis title along the line of the right edge of the viewport.

From any of these base positions, you can displace the X-Axis title left or right by setting the resource *tiXAxisOffsetXF*. If you set *tiXAxisPosition* to `Bottom` or `Top`, **Title** issues a warning message and reverts to the default value, `Center`.

Default: `Center`

tiXAxisOffsetXF

Sets an offset in NDC coordinates by which the X-Axis title will be displaced in a direction parallel to the X Axis from its base location. This value does *not* scale in response to changes in the viewport size.

Default: 0.0

tiXAxisOffsetYF

Sets a positive or negative offset in NDC coordinates by which the X-Axis title will be displaced in a direction parallel to the Y Axis from its base location. This offset is applied after any offset due to *tiDeltaF* and the adjustment performed to prevent the X-Axis title from intruding into the viewport. Therefore, this resource allows you to displace the title any distance away from the viewport or into its interior. The positive direction coincides with that of the NDC Y Axis. Its value does *not* scale in response to changes in the viewport size.

V4.1 Status Note 1

Default: 0.0

tiXAxisJust

This resource of type NhlTJustification sets the Justification point of the X-Axis title.

Default: `CenterCenter`

tiXAxisFont

This resource of type NhlTFont sets the font index for the X-Axis title.

Default: 0

tiXAxisFontColor

Sets the font color for the X-Axis title.

Default: 1

tiXAxisFontHeightF

Sets the font height in NDC coordinates of the X-Axis title. This value scales proportionally to changes in the width of the viewport.

This resource may be intercepted or disabled by:

- **PlotManager**

Default: 0.025

tiXAxisFontAspectF

This resource sets the aspect ratio, defined as the height divided by the width of a reference character box within which all characters of the font will fit. Its value determines the width of characters in the X-Axis title relative to *tiXAxisFontHeightF*. If given the value 1.0, wide characters (such as 'M' or 'W') will be approximately square in shape. Values increasing from 1.0 result in thinner characters, while values decreasing from 1.0 result in wider characters.

Default: 1.3125

tiXAxisConstantSpacingF

If this resource has the value 0.0, the characters in the X-Axis title will be proportionally spaced. If it has a value greater than 0.0, the distance from the beginning of one character to the beginning of the next character will be constant. The value of *tiXAxisConstantSpacingF* sets this constant distance as a multiple of the width of a reference character box within which all characters of the font will fit. If *tiXAxisConstantSpacingF* has the value 1.0, the constant distance will cause there to be almost no space between two wide characters (such as 'M' or 'W') placed next to each other. As *tiXAxisConstantSpacingF* decreases from 1.0, characters will begin to overlap. As it increases from 1.0, characters become further separated from each other. Values less than 0.0 result in an error and are replaced with the default value.

Default: 0.0

tiXAxisFontThicknessF

Sets the thickness of the lines used to draw the characters of the X-Axis title. The value acts as a multiplier of a (device-dependent) unit thickness. This resource is ignored when the *tiXAxisFont* specifies a filled font (font indexes 21-22, 25-26, 29-30, and 33-37).

Default: 1.0

tiXAxisFuncCode

Specifies the character used to delimit Text Function Codes that may be embedded in the X-Axis title.

Default: **:**

tiXAxisFontQuality

Sets the font quality for the X-Axis title.

Default: `High`

tiXAxisAngleF

Sets the counterclockwise rotation angle for the X-Axis title.

Default: 0.0

tiXAxisDirection

This resource of type NhlTTextDirection specifies the direction of the text used for the X-Axis title. There are two choices:

`Down`

Each character is placed below the previous character in the text string.

`Across`

Each character is placed to the right of the previous character in the text string.

These descriptions apply before any rotation due to *tiXAxisAngleF* is applied to the X-Axis title.

Default: `Across`

---

tiYAxisOn

Setting this boolean resource True causes the Y-Axis title to appear. Setting it False turns the Y-Axis title off. By default, *tiYAxisOn* will be set True if and only if you explicitly set *tiYAxisString*.

Default: <dynamic>

tiYAxisString

Sets the string to use as the Y-Axis title. The string may contain Text Function Codes that allow you to control typographical attributes such as subscripting and superscripting, change fonts within a string, embed newlines, etc.

Default: "YAxis"

tiYAxisSide

This resource of type NhlTPosition specifies the side of the viewport relative to which the justification point of the Y-Axis title is located. There are two choices:

`Left`

Locate the justification point of the Y-Axis title relative to the left edge of the viewport.

`Right`

Locate the justification point of the Y-Axis title relative to the right edge of the viewport.

If you set *tiYAxisPosition* to `Bottom`, `Center`, or `Top` **Title** issues a warning message and reverts to the default value, `Left`.

Default: `Left`

tiYAxisPosition

This resource of type NhlTPosition specifies the base vertical location of the justification point of the Y-Axis title with respect to the viewport boundaries. There are three choices:

`Bottom`

Locate the base position of the justification point of the Y-Axis title along the line of the bottom edge of the viewport.

`Center`

Locate the base position of the justification point of the Y-Axis title along the line of the horizontal center of the viewport.

`Top`

Locate the base position of the justification point of the Y-Axis title along the line of the top edge of the viewport.

From any of these base positions, you can displace the Y-Axis title up or down by setting the resource *tiYAxisOffsetYF*. If you set *tiYAxisPosition* to `Right` or `Left` **Title** issues a warning message and reverts to the default value, `Center`.

Default: `Center`

tiYAxisOffsetYF

Sets an offset in NDC coordinates by which the Y-Axis title will be displaced in a direction parallel to the Y Axis from its base location. This value does *not* scale in response to changes in the viewport size.

Default: 0.0

tiYAxisOffsetXF

Sets a positive or negative offset in NDC coordinates by which the Y-Axis title will be displaced in a direction parallel to the X Axis from its base location. This offset is applied after any offset due to *tiDeltaF* and the adjustment performed to prevent the Y-Axis title from intruding into the viewport. Therefore, this resource allows you to displace the title any distance away from the viewport or into its interior. The positive direction coincides with that of the NDC X Axis. Its value does *not* scale in response to changes in the viewport size.

V4.1 Status Note 1

Default: 0.0

tiYAxisJust

This resource of type NhlTJustification sets the Justification point of the Y-Axis title.

Default: `CenterCenter`

tiYAxisFont

This resource of type NhlTFont sets the font index for the Y-Axis title.

Default: 0

tiYAxisFontColor

Sets the font color for the Y-Axis title.

Default: 1

tiYAxisFontHeightF

Sets the font height in NDC coordinates to use for the Y-Axis title. This values scales proportionally to changes in the height of the viewport.

This resource may be intercepted or disabled by:

- **PlotManager**

Default: 0.025

tiYAxisFontAspectF

This resource sets the aspect ratio, defined as the height divided by the width of a reference character box within which all characters of the font will fit. Its value determines the width of characters in the Y-Axis title relative to *tiYAxisFontHeightF*. If given the value 1.0, wide characters (such as ’M’ or ’W’) will be approximately square in shape. Values increasing from 1.0

result in thinner characters, while values decreasing from 1.0 result in wider characters.

Default: 1.3125

tiYAxisConstantSpacingF

If this resource has the value 0.0, the characters in the Y-Axis title will be proportionally spaced. If it has a value greater than 0.0, the distance from the beginning of one character to the beginning of the next character will be constant. The value of *tiYAxisConstantSpacingF* sets this constant distance as a multiple of the width of a reference character box within which all characters of the font will fit. If *tiYAxisConstantSpacingF* has the value 1.0, the constant distance will cause there to be almost no space between two wide characters (such as ’M’ or ’W’) placed next to each other. As *tiYAxisConstantSpacingF* decreases from 1.0, characters will begin to overlap. As it increases from 1.0, characters become further separated from each other. Values less than 0.0 result in an error and are replaced with the default value.

Default: 0.0

tiYAxisFontThicknessF

Sets the thickness of the lines used to draw the characters of the Y-Axis title. The value acts as a multiplier of a (device-dependent) unit thickness. This resource is ignored when the *tiYAxisFont* specifies a filled font (font indexes 21-22, 25-26, 29-30, and 33-37).

Default: 1.0

tiYAxisFuncCode

Specifies the character used to delimit Text Function Codes that may be embedded in the Y-Axis title.

Default: **:**

tiYAxisFontQuality

Sets the font quality for the Y-Axis title.

Default: `High`

tiYAxisAngleF

Sets the counterclockwise rotation angle for the Y-Axis title.

Default: 90.0

tiYAxisDirection

This resource of type NhlTTextDirection specifies the direction of the text used for the Y-Axis title. There are two choices:

`Down`

Each character is placed below the previous character in the text string.

`Across`

Each character is placed to the right of the previous character in the text string.

These descriptions apply before any rotation due to *tiYAxisAngleF* is applied to the Y-Axis title.

Default: `Across`

TickMark class resource descriptions

tmSciNoteCutoff

This resource sets the value used for all **TickMark** label formats that have the dynamic attribute set on for the exponent switch length (as specified by the Floating Point Format Specification scheme). By default all the label format strings set the exponent switch length dynamic attribute on, causing the value of *tmSciNoteCutoff* to specify the number of digits used to control switching between exponential and floating point format when generic formatting is in effect. If you explicitly set the exponent switch length conversion field (signaled by the '?' character) in the format string to anything other than the dynamic value ('*'), the value in the format string overrides the value set for *tmSciNoteCutoff*.

Default: 6

tmBorderThicknessF

Sets the linewidth scale factor for the tick mark border. The linewidth is a multiple of the parent workstation device's minimum linewidth.

Default: 2.0

tmBorderLineColor

Set the color of the tick mark border.

Default: 1

tmXUseBottom

Causes the top tick marks to use the resources for the bottom tick marks. The resources affected are:

> *tmXTStyle*, *tmXTMode*, *tmXTMinorPerMajor*, *tmXTNoMinor*, *tmXTDataLeftF*, *tmXTDataRightF*, *tmXTTickStartF*, *tmXTTickEndF*, *tmXTMaxTicks*, *tmXTTickSpacing*, *tmXTValues*, *tmXTLabels*, *tmXTMinorValues*, *tmXTMajorThicknessF*, *tmXTMajorLineColor*, *tmXTMajorLengthF*, *tmXTMajorOutwardLengthF*, *tmXTMinorThicknessF*, *tmXTMinorLineColor*, *tmXTMinorLengthF*, *tmXTMinorOutwardLength*, *tmXTLabelFuncCode*, *tmXTConstantSpacingF*, *tmXTLabelFont*, *tmXTLabelFontHeightF*, *tmXTLabelFontColor*, *tmXTLabelFontAspectF*, *tmXTLabelFontThicknessF*, *tmXTLabelFontQuality*, *tmXTLabelAngleF*, *tmXTLabelDirection*, *tmXTLabelDeltaF*, *tmXTIrregularPoints*, *tmXTIrrTensionF*, *tmXTLabelStride*, *tmXTAutoPrecision*, *tmXTPrecision* and *tmXTFormat*.

All other Top **TickMark** resources are unaffected.

Default: True

tmXMajorGrid

Setting True turns on the X Axis major grid; False turns it off. Grid lines will extend the bottom tick marks to the top border. If the bottom ticks are turned off, the top tick marks will be extended to the bottom.

Default: False

tmXMinorGrid

Setting True turns on the X Axis minor grid; False turns it off. Grid lines will extend the bottom tick marks to the top border. If the bottom ticks are turned off, the top tick marks will be extended to the bottom.

Default: False

tmXMajorGridThicknessF

Sets the linewidth scale factor for the thickness of the major X Axis grid lines. Value is a multiple of the device's minimum linewidth.

Default: 2.0

tmXMajorGridLineColor

Sets color index for major X Axis grid lines.

Default: 1

tmXMajorGridLineDashPattern

Sets dash pattern index for the major X Axis grid lines.

Default: 0

tmXMinorGridThicknessF

Sets the linewidth scale factor for the minor grid. Value is a multiple of the device's minimum linewidth.

Default: 1.0

tmXMinorGridLineColor

Sets color index for minor X Axis grid lines.

Default: 1

tmXMinorGridLineDashPattern

Sets dash pattern index for minor X Axis grid lines.

Default: 0

tmXBOn

Setting True turns on bottom tick marks; False turns them off.

Default: True

tmXTOn

Setting True turns on top tick marks; False turns them off.

Default: True

tmXBLabelsOn

Setting True turns on tick mark labels if *tmXBOn* is set; False turns them off.

Default: True

tmXTLabelsOn

Setting True turns on tick mark labels if *tmXTOn* is set; False turns them off.

Default: False

tmXBBorderOn

Setting True turns bottom border on; False turns it off. Does not turn off bottom ticks.

Default: True

tmXTBorderOn

Setting True turns top border on; False turns it off. Does not turn off top ticks.

Default: True

tmXBMode

This enumerated resource of type NhlTTickMarkMode determines the method for specifying the spacing of the ticks and the contents of the labels along the bottom X Axis. It has three possible settings:

`Automatic`:

Based on the values of *tmXBDataLeftF* and *tmXBDataRightF*, **TickMark** chooses a 'nice' spacing between the major tick marks, such that the number of major ticks does not exceed the value of *tmXBMaxTicks*. Labels are generated based on the current settings of *tmSciNoteCutoff*, *tmXBAutoPrecision*, *tmXBPrecision*, and *tmXBFormat*.

`Manual`:

Starting with the value *tmXBTickStartF*, **TickMark** sets major tick marks at intervals separated by a distance of *tmXBTickSpacingF* until *tmXBTickEndF* is exceeded. Of these tick marks, only those that fall between *tmXBDataLeftF* and *tmXBDataRightF* may appear along the bottom X Axis. If *tmXBTickStartF* and *tmXBTickEndF* are not set, **TickMark** issues an informational message and sets *tmXBTickStartF* to the minimum of *tmXBDataLeftF* and *tmXBDataRightF*, and it sets *tmXBTickEndF* to the maximum of *tmXBDataLeftF* and *tmXBDataRightF*. If *tmXBTickSpacingF* is not set, **TickMark** issues a warning message and *tmXBMode* reverts to `Automatic` mode. Labels are generated based on the current settings of *tmXBTickSpacingF*, *tmSciNoteCutoff*, *tmXBAutoPrecision*, *tmXBPrecision*, and *tmXBFormat*.

`Explicit`:

TickMark uses the array resource *tmXBValues* to determine the coordinate positions of the tick marks. Of these tick marks, only those that fall between *tmXBDataLeftF* and *tmXBDataRightF* may appear along the bottom X Axis. If *tmXBValues* is not set, **TickMark** issues a warning message and *tmXBMode* reverts to `Automatic` mode. The corresponding element of the string array resource *tmXBLabels* determines the label used for each tick specified by *tmXBValues*. If *tmXBLabels* is not set, **TickMark** issues a warning and outputs no labels for the bottom X Axis. You may optionally also set the array resource *tmXBMinorValues* to specify the locations of minor ticks along the axis.

Default: `Automatic`

tmXTMode

This enumerated resource of type NhlTTickMarkMode determines the method for specifying the spacing of the ticks and the contents of the labels along the top X Axis. It has three possible settings:

`Automatic`:

Based on the values of *tmXTDataLeftF* and *tmXTDataRightF*, **TickMark** chooses a 'nice' spacing between the major tick marks, such that the number of major ticks does not exceed the value of *tmXTMaxTicks*. Labels are generated based on the current settings of *tmSciNoteCutoff*, *tmXTAutoPrecision*, *tmXTPrecision*, and *tmXTFormat*.

`Manual`:

Starting with the value *tmXTTickStartF*, **TickMark** sets major tick marks at intervals separated by a distance of *tmXTTickSpacingF* until *tmXTTickEndF* is exceeded. Of these tick marks, only those that fall between *tmXTDataLeftF* and *tmXTDataRightF* may appear along the top X Axis. If *tmXTTickStartF* and *tmXTTickEndF* are not set, **TickMark** issues an informational message and sets *tmXTTickStartF* to the minimum of *tmXTDataLeftF* and *tmXTDataRightF*, and it sets *tmXTTickEndF* to the maximum of *tmXTDataLeftF* and *tmXTDataRightF*. If *tmXTTickSpacingF* is not set, **TickMark** issues a warning message and *tmXTMode* reverts to `Automatic` mode. Labels are generated based on the current settings of *tmXTTickSpacingF*, *tmSciNoteCutoff*, *tmXTAutoPrecision*, *tmXTPrecision*, and *tmXTFormat*.

`Explicit`:

TickMark uses the array resource *tmXTValues* to determine the coordinate positions of the tick marks. Of these tick marks, only those that fall between *tmXTDataLeftF* and *tmXTDataRightF* may appear along the top X Axis. If *tmXTValues* is not set, **TickMark** issues a warning message and *tmXTMode* reverts to `Automatic` mode. The corresponding element of the string array resource *tmXTLabels* determines the label used for each tick specified by *tmXTValues*. If *tmXTLabels* is not set, **TickMark** issues a warning and outputs no labels for the top X Axis. You may optionally also set the array resource *tmXTMinorValues* to specify the locations of minor ticks along the axis.

Default: `Automatic`

tmXBStyle

This enumerated resource of type NhlTTickMarkStyle sets the style of the **TickMark** bottom X Axis. There are 5 styles:

`Linear`:

A linear coordinate system is set up along the X Axis with the left boundary set to the value of *tmXBDataLeftF* and the right boundary set to the value of *tmXBDataRightF*. If *tmXBDataLeftF*and *tmXBDataRightF* are found to be equal when compared with a precision of 7 significant digits, a warning is issued and the bottom X-Axis tick marks are turned off.

`Log`:

A logarithmic coordinate system is set up along the X Axis with the left boundary set to the value of *tmXBDataLeftF* and the right boundary set to the value of *tmXBDataRightF*. If either *tmXBDataLeftF* or *tmXBDataRightF* is less than or equal to 0.0, a warning is issued and the bottom X-Axis tick marks are turned off.

`Irregular`:

An irregular coordinate system is set up along the X Axis based on the coordinate samples contained in the array resource *tmXBIrregularPoints*. If *tmXBIrregularPoints* is found to be invalid, a warning is issued and *tmXBStyle* reverts to `Linear`. If the data bounds *tmXBDataLeftF* or *tmXBDataRightF* are only partially within the range defined by the maximum and minimum elements of *tmXBIrregularPoints*, a warning is issued and, as appropriate, either *tmXBDataLeftF* or *tmXBDataRightF* is reset to the value of the maximum or minimum element of the *tmXBIrregularPoints*. If the range between *tmXBDataLeftF* and *tmXBDataRightF* is entirely outside the range of the *tmXBIrregularPoints* array, a fatal error is currently issued.

`Time`:

Not yet implemented.

`Geographic`:

Not yet implemented.

This resource may be intercepted or disabled by:

- **PlotManager**

Default: `Linear`

tmXBIrrTensionF

Specifies the value of the tension parameter applied to the spline approximation used to set up an irregular transformation based on the values in the *tmXBIrregularPoints* array. This resource is only used if tmXBStyle is `Irregular`. Small values (less than ~1.0) imply a relaxed approximation large values (greater than ~5.0) imply a tight approximation.

This resource may be intercepted or disabled by:

- **PlotManager**

V4.1 Status Note 2

Default: 2.0

tmXTStyle

This enumerated resource of type NhlTTickMarkStyle sets the style of the **TickMark** top X Axis. There are 5 styles:

`Linear`:

A linear coordinate system is set up along the X Axis with the left boundary set to the value of *tmXTDataLeftF* and the right boundary set to the value of *tmXTDataRightF*. If *tmXTDataLeftF*and *tmXTDataRightF* are found to be equal when compared with a precision of 7 significant digits, a warning is issued and the top X-Axis tick marks are turned off.

`Log`:

A logarithmic coordinate system is set up along the X Axis with the left boundary set to the value of *tmXTDataLeftF* and the right boundary set to the value of *tmXTDataRightF*. If either *tmXTDataLeftF* or *tmXTDataRightF* is less than or equal to 0.0, a warning is issued and the top X-Axis tick marks are turned off.

`Irregular`:

An irregular coordinate system is set up along the X Axis based on the coordinate samples contained in the array resource *tmXTIrregularPoints*. If *tmXTIrregularPoints* is found to be invalid, a warning is issued and *tmXTStyle* reverts to `Linear`. If the data bounds *tmXTDataLeftF* or *tmXTDataRightF* are only partially within the range defined by the maximum and minimum elements of *tmXTIrregularPoints*, a warning is issued and, as appropriate, either *tmXTDataLeftF* or *tmXTDataRightF* is reset to the value of the maximum or minimum element of the *tmXTIrregularPoints*. If the range between *tmXTDataLeftF* and *tmXTDataRightF* is entirely outside the range of the *tmXTIrregularPoints* array, a fatal error is currently issued.

`Time`:

Not yet implemented.

`Geographic`:

Not yet implemented.

This resource may be intercepted or disabled by:

- **PlotManager**

Default: `Linear`

tmXTIrrTensionF

Specifies the value of the tension parameter applied to the spline approximation used to set up an irregular transformation based on the values in the *tmXTIrregularPoints* array. This resource is only used if tmXTStyle is `Irregular`. Small values (less than ~1.0) imply a relaxed approximation large values (greater than ~5.0) imply a tight approximation.

This resource may be intercepted or disabled by:

- **PlotManager**

V4.1 Status Note 2

Default: 2.0

tmXBAutoPrecision

When True, this boolean resource causes **TickMark** to set the precision (number of significant digits) for the bottom tick mark labels based on the range of values used for the labels. In general, the value of *tmXBPrecision* is ignored. However, if *tmXBMode* is set to `Manual`, the precision is set such that all significant digits necessary to express the value of the *tmXBTickSpacingF* resource are used in determining the precision. In this case, the value of *tmXBPrecision* sets a cap on the number of significant digits used to express the value of *tmXBTickSpacingF*.

When set False, the value of *tmXBPrecision* controls the precision. However, if the significant digit conversion field (signaled by the '.' character) is set explicitly in the *tmXBFormat* string (according to the rules of the Floating Point Format Specification scheme), then the resources *tmXBAutoPrecision* and *tmXBPrecision* are both ignored, and the precision is controlled completely by the *tmXBFormat* specification.

Default: True

tmXBPrecision

If *tmXBAutoPrecision* is set False and the significant digit field in the *tmXBFormat* string (as specified by the Floating Point Format Specification scheme) has the dynamic attribute set on, this resource controls the precision used for the bottom tick mark labels. Assuming the default setting of the *tmXBFormat* resource, the value set for *tmXBPrecision* will be the number of digits used for the bottom tick mark label representing the value with the maximum absolute value. The other labels will have the same number of digits to the right of the decimal point, but may have fewer digits to the left of the decimal point. All labels will be rounded to the number of decimal places represented.

Default: 4

tmXBFormat

This string resource specifies, according to the HLU Floating Point Format Specification scheme, the format of the numbers used to label the major ticks along the bottom X Axis. It applies only when *tmXBMode* is set to `Automatic` or `Manual`. Unless set explicitly, the significant digits conversion field is set dynamically based on the values of the *tmXBAutoPrecision* and *tmXBPrecision* resources. Likewise, the exponent switch length conversion field is set dynamically based on the value of the *tmSciNoteCutoff* resource. If the leftmost significant digit conversion field has the dynamic attribute set, the assumed leftmost significant digit is set dynamically for each label based on the largest absolute value used to generate the bottom labels. Combined with the zero format flag, this has the effect of causing all the labels to contain the same number of places to the right of the decimal point. The default format string also sets the "at-sign" ('@') format flag to force fractional values to include an initial '0' character preceding the decimal point. In addition, the exponent conversion field includes, by default, the 's' flag to cause exponents to be written using the superscript notational form.

If *tmXBStyle* is set to `Log`, exponential notation is forced on, regardless of the character used for the conversion specifier. Also the zero flag is turned off to ensure that the mantissas (which always evaluate to 1.0 because of the spacing enforced for `Log` style labels) are not turned on because of the need to fill out the significant digits on the right of the decimal point.

Default: "0@*+^sg"

tmXTAutoPrecision

When True, this boolean resource causes **TickMark** to set the precision (number of significant digits) for the top tick mark labels based on the range of values used for the labels. In general, the value of *tmXTPrecision* is ignored. However, if *tmXTMode* is set to `Manual`, the precision is set such that all significant digits necessary to express the value of the *tmXTTickSpacingF* resource are used in determining the precision. In this case, the value of *tmXTPrecision* sets a cap on the number of significant digits used to express the value of *tmXTTickSpacingF*.

When set False, the value of *tmXTPrecision* controls the precision. However, if the significant digit conversion field (signaled by the '.' character) is set explicitly in the *tmXTFormat* string (according to the rules of the Floating Point Format Specification scheme),then the resources *tmXTAutoPrecision* and *tmXTPrecision* are both ignored, and the precision is controlled completely by the *tmXTFormat* specification.

Default: True

tmXTPrecision

If *tmXTAutoPrecision* is set False and the significant digit field in the *tmXTFormat* string (as specified by the Floating Point Format Specification scheme) has the dynamic attribute set on, this resource controls the precision used for the top tick mark labels. Assuming the default setting of the *tmXTFormat* resource, the value set for *tmXTPrecision* will be the number of digits used for the top tick mark label representing the value with the maximum absolute value. The other labels will have the same number of digits to the right of the decimal point, but may have fewer digits to the left of the decimal point. All labels will be rounded to the number of decimal places represented.

Default: 4

tmXTFormat

This string resource specifies, according to the HLU Floating Point Format Specification scheme, the format of the numbers used to label the major ticks along the top X Axis. It applies only when *tmXTMode* is set to `Automatic` or `Manual`. Unless set explicitly, the significant digits conversion field is set dynamically based on the values of the *tmXTAutoPrecision* and *tmXTPrecision* resources. Likewise, the exponent switch length conversion field is set dynamically based on the value of the *tmSciNoteCutoff* resource. If the leftmost significant digit conversion field has the dynamic attribute set, the assumed leftmost significant digit is set dynamically for each label based on the largest absolute value used to generate the top labels. Combined with the zero format flag, this has the effect of causing all the labels to contain the same number of places to the right of the decimal point. The default format string also sets the "at-sign" ('@') format flag to force fractional values to include an initial '0' character preceding the decimal point. In addition, the exponent conversion field includes, by default, the 's' flag to cause exponents to be written using the superscript notational form.

If *tmXTStyle* is set to `Log`, exponential notation is forced on, regardless of the character used for the conversion specifier. Also the zero flag is turned off to ensure that the mantissas (which always evaluate to 1.0 because of the spacing enforced for `Log` style labels) are not turned on because of the need to fill out the significant digits on the right of the decimal point.

Default: "0@*+^sg"

tmXBMinorPerMajor

Sets the number of minor tick marks per major tick mark for the bottom X Axis. When using `Log` style tick marks, only the values 1, 4, and 8 are accepted as valid. If the `Explicit` mode is in effect, **TickMark** ignores this resource: you must instead specify the spacing of each minor tick individually using *tmXBMinorValues*. If you do not set this resource and the mode is `Automatic`, its value is determined dynamically depending on the major tick spacing, as follows: if the major spacing is 1 or 5 times some power of 10, *tmXBMinorPerMajor* is set to 4; if the spacing is 2 or 4 times a power of 10, it is set to 3; if the spacing is 3 times a power of 10, it is set to 2. Otherwise, if the mode is `Manual`, *tmXBMinorPerMajor* defaults to the value 3.

Default: <dynamic>

tmXTMinorPerMajor

Sets the number of minor tick marks per major tick mark for the top X Axis. When using `Log` style tick marks, only the values 1, 4, and 8 are accepted as valid. If the `Explicit` mode is in effect, **TickMark** ignores this resource: you must instead specify the spacing of each minor tick individually using *tmXTMinorValues*. If you do not set this resource and the mode is `Automatic`, its value is determined dynamically depending on the major tick spacing, as follows: if the major spacing is 1 or 5 times some power of 10, *tmXTMinorPerMajor* is set to 4; if the spacing is 2 or 4 times a power of 10, it is set to 3; if the spacing is 3 times a power of 10, it is set to 2. Otherwise, if the mode is `Manual`, *tmXTMinorPerMajor* defaults to the value 3.

Default: <dynamic>

tmXBMinorOn

Setting this resource True turns on minor tick marks; False turns them off.

Default: True

tmXTMinorOn

Setting this resource True turns on minor tick marks; False turns them off.

Default: True

tmXBLabelStride

Used to skip tick mark labels. The value represents how many tick marks to skip between tick mark labels when drawing the tick mark labels. Will cause the tick mark object to draw first, then skip.

Default: 0

tmXTLabelStride

Used to skip tick mark labels. The value represents how many tick marks to skip between tick mark labels when drawing the tick mark labels. Will cause the tick mark object to draw first, then skip.

Default: 0

tmXBDataLeftF

Sets the data value that maps to the left side of the bottom tick marks.

This resource may be intercepted or disabled by:

- **PlotManager**

Default: 0.0

tmXBDataRightF

Sets the data value that maps to the right side of the bottom tick marks.

This resource may be intercepted or disabled by:

- **PlotManager**

Default: 0.0

tmXBTickStartF

Sets the beginning of the tick marks. Must be less than *tmXBTickEndF*. If *tmXBTickStartF* is greater than the minimum data extent, no tick marks will be drawn between the minimum data extent and *tmXBTickStartF*.

Default: 0.0

tmXBTickEndF

Sets the ending of the tick marks. Must be greater than *tmXBTickStartF*. If *tmXBTickEndF* is less than the maximum data extent, no tick marks will be drawn between the maximum data extent and *tmXBTickEndF*.

Default: 0.0

tmXBMaxTicks

Used for `Automatic` mode ticks only. Specifies the maximum allowable tick marks for the bottom X Axis.

Default: 7

tmXBTickSpacingF

Used for `Manual` mode ticks only. Specifies the spacing between major tick marks. If `Log` style is being used, this value must be the number of decades (i.e. 1 means ticks will be placed and labels at 10**1, 10**2, etc.)

Default: 0.0

tmXBIrregularPoints

Used for `Irregular` style ticks only. Specifies an array of data locations spaced at equal intervals along the bottom X Axis. These points represent a discrete approximation to the desired coordinate system. These points must be strictly monotonically increasing or decreasing. Therefore, periodic coordinate systems can not be approximated using this resource.

This resource may be intercepted or disabled by:

- **PlotManager**

Default: NULL

tmXBValues

Used only for `Explicit` mode tick marks. This resource is an array of data locations at which tick marks will be drawn. The values must increase or decrease monotonically. If you do not set *tmXBValues* when in `Explicit` mode, **TickMark** issues a warning and *tmXBMode* reverts to `Automatic` mode.

V4.1 Status Note 1

Default: NULL

tmXBLabels

Used only for `Explicit` mode tick marks. This resource is an array of strings that correspond to the values in *tmXBValues*. Each element of this resource is placed under the corresponding tick location in *tmXBValues*. If a string array element is set to NULL, no label will appear at the corresponding tick mark location. If you do not set *tmXBLabels* when in `Explicit` mode, **TickMark** issues a warning and outputs no labels for the bottom X Axis.

Default: NULL

tmXBMinorValues

Used only for `Explicit` mode tick marks. This resource is an array of data locations at which the minor tick marks will be drawn. If not set, minor tick marks will be absent. If you do not set *tmXBMinorValues* when in `Explicit` mode, minor tick marks will be absent from the bottom X Axis.

Default: NULL

tmXBMajorThicknessF

Sets linewidth scale factor for bottom major tick marks. Value is a multiple of the minimum linewidth on the current device.

Default: 2.0

tmXBMajorLineColor

Sets the line color index for the major tick marks.

Default: 1

tmXBMajorLengthF

Sets the length of the X-Axis bottom major tick marks in NDC coordinates. This length scales with changes to the viewport height, unless you explicitly set *tmXBMajorLengthF* at the same time.

Default: <dynamic> -- initially 0.02 for a viewport height of 0.6

tmXBMajorOutwardLengthF

Length of the major X-Axis bottom tick marks that will be drawn on the outside of the tick mark border. Value is in NDC coordinates. This value scales proportionally to changes in height.

Default: 0.0

tmXBMinorThicknessF

Sets linewidth scale factor for bottom minor tick marks. Value represents a multiple of the current device's minimum linewidth.

Default: 1.0

tmXBMinorLineColor

Sets color index for bottom minor tick marks.

Default: 1

tmXBMinorLengthF

Sets the length in NDC coordinates for X bottom tick marks. This length scales with changes to the viewport height, unless you explicitly set *tmXBMinorLengthF* at the same time.

Default: <dynamic> -- initially 0.01 for a viewport height of 0.6

tmXBMinorOutwardLengthF

Length of minor X-Axis bottom tick marks that will be drawn on the outside of the tick mark border. Value is in NDC coordinates. This value scales proportionally to changes in height.

Default: 0.0

tmXBLabelFuncCode

This resource of type NhlTCharacter sets the function code character for the X-Axis bottom labels.

Default: **:**

tmXBLabelFont

Sets the font index for the bottom X-Axis labels.

Default: 0

tmXBLabelFontHeightF

Sets the font height in NDC coordinates for the bottom X Axis labels. This value scales proportionally to changes in the width of the viewport.

Default: 0.02

tmXBLabelFontColor

Sets the color of the bottom X-Axis labels.

Default: 1

tmXBLabelFontAspectF

Sets the ratio of height/width for the bottom X-Axis labels.

Default: 1.3125

tmXBLabelFontThicknessF

Sets the thickness of the lines used to draw the characters of the bottom X-Axis label. The value acts as a multiplier of a (device-dependent) unit thickness. This resource is ignored when the *tmXBLabelFont* specifies a filled font (font indexes 21-22, 25-26, 29-30, and 33-37).

Default: 1.0

tmXBLabelFontQuality

This resource of type NhlTFontQuality determines the quality of the font used to draw the bottom X-Axis label.

Default: `High`

tmXBLabelConstantSpacingF

If this resource has the value 0.0, the characters in the bottom X-Axis label will be proportionally spaced. If it has a value greater than 0.0, the distance from the beginning of one character to the beginning of the next character will be constant. The value of *tmXBLabelConstantSpacingF* sets this constant distance as a multiple of the width of a reference character box within which all characters of the font will fit. If *tmXBLabelConstantSpacingF* has the value 1.0, the constant distance will cause there to be almost no space between two wide characters (such as 'M' or 'W') placed next to each other. As *tmXBLabelConstantSpacingF* decreases from 1.0, characters will begin to overlap. As it increases from 1.0, characters become further separated from each other. Values less than 0.0 result in an error and are replaced with the default value.

Default: 0.0

tmXBLabelJust

Sets the justification point for all bottom X-Axis labels.

Default: `TopCenter`

tmXBLabelAngleF

Sets the counterclockwise rotation of the X-Axis bottom labels.

Default: 0.0

tmXBLabelDirection

Sets the drawing direction of the X-Axis bottom labels.

Default: `Across`

tmXBLabelDeltaF

Sets an offset from the default location for the bottom tick mark labels. This value multiplied by the *tmXBFontHeightF* represents the amount, in NDC coordinates, that the bottom tick marks will be displaced from their default location in the Y direction. A positive value moves labels in the positive direction in NDC coordinates, and a negative value moves labels in the negative direction.

Default: 0.0

tmXTDataLeftF

Sets the data value that maps to the left side of the top tick marks.

This resource may be intercepted or disabled by:

- **PlotManager**

Default: 0.0

tmXTDataRightF

Sets the data value that maps to the right side of the top tick marks.

This resource may be intercepted or disabled by:

- **PlotManager**

Default: 0.0

tmXTTickStartF

Sets the beginning of the tick marks. Must be less than *tmXTTickEndF*. If *tmXTTickStartF* is greater than the minimum data extent, no tick marks will be drawn between the minimum data extent and *tmXTTickStartF*.

Default: 0.0

tmXTTickEndF

Sets the ending of the tick marks. Must be greater than *tmXTTickStartF*. If *tmXTTickEndF* is less than the maximum data extent, no tick marks will be drawn between the maximum data extent and *tmXTTickEndF*.

Default: 0.0

tmXTMaxTicks

Used for `Automatic` mode ticks only. Specifies the maximum allowable tick marks for the top X Axis.

Default: 7

tmXTTickSpacingF

Used for `Manual` mode ticks only. Specifies the spacing between major tick marks. If `Log` style is being used, this value must be the number of decades (i.e. 1 means ticks will be placed and labels at 10**1, 10**2, etc.)

Default: 0.0

tmXTIrregularPoints

Used for `Irregular` style ticks only. Specifies an array of data locations spaced at equal intervals along the top X Axis. These points represent a discrete approximation to the desired coordinate system. These points must be strictly monotonically increasing or decreasing. Therefore, periodic coordinate systems can not be approximated using this resource.

This resource may be intercepted or disabled by:

- **PlotManager**

Default: NULL

tmXTValues

Used only for `Explicit` mode tick marks. This resource is an array of data locations at which tick marks will be drawn. The values must increase or decrease monotonically. If you do not set *tmXTValues* when in `Explicit` mode, **TickMark** issues a warning and *tmXTMode* reverts to `Automatic` mode.

V4.1 Status Note 1

Default: NULL

tmXTLabels

Used only for `Explicit` mode tick marks. This resource is an array of strings that correspond to the values in *tmXTValues*. Each element of this resource is placed under the corresponding tick location in *tmXTValues*. If a string array element is set to NULL, no label will appear at the corresponding tick mark location. If you do not set *tmXTLabels* when in `Explicit` mode (assuming *tmXUseBottom* is False), **TickMark** issues a warning and outputs no labels for the top X Axis.

Default: NULL

tmXTMinorValues

Used only for `Explicit` mode tick marks. This resource is an array of data locations at which the minor tick marks will be drawn. If not set, minor tick marks will be absent. If you do not set *tmXTMinorValues* when in `Explicit` mode (assuming *tmXUseBottom* is False), minor tick marks will be absent from the top X Axis.

Default: NULL

tmXTMajorThicknessF

Sets linewidth scale factor for top major tick marks. Value is a multiple of the minimum linewidth on the current device.

Default: 2.0

tmXTMajorLineColor

Sets the line color index for the top major tick marks.

Default: 1

tmXTMajorLengthF

Sets the length of the X-Axis top major tick marks in NDC coordinates. This length scales with changes to the viewport height, unless you explicitly set *tmXTMajorLengthF* (or *tmXBMajorLengthF* if *tmXUseBottom* is True) at the same time.

Default: <dynamic> -- initially 0.02 for a viewport height of 0.6

tmXTMajorOutwardLengthF

Length of the major X-Axis top tick marks that will be drawn on the outside of the tick mark border. Value is in NDC coordinates. This value scales proportionally to changes in height.

Default: 0.0

tmXTMinorThicknessF

Sets linewidth scale factor for top minor tick marks. Value represents a multiple of the current device's minimum linewidth.

Default: 1.0

tmXTMinorLineColor

Sets color index for top minor tick marks.

Default: 1

tmXTMinorLengthF

Sets the length in NDC coordinates for X top tick marks. This length scales with changes to the viewport height, unless you explicitly set *tmXTMinorLengthF* (or *tmXBMinorLengthF* if *tmXUseBottom* is True) at the same time.

Default: <dynamic> -- initially 0.01 for a viewport height of 0.6

tmXTMinorOutwardLengthF

Length of minor X-Axis top tick marks that will be drawn on the outside of the tick mark border.

Value is in NDC coordinates. This value scales proportionally to changes in height.

Default: 0.0

tmXTLabelFuncCode

This resource of type NhlTCharacter sets the function code character for the X-Axis top labels.

Default: **:**

tmXTLabelFont

Sets the font index for the bottom X-Axis labels.

Default: 0

tmXTLabelFontHeightF

Sets the font height in NDC coordinates for the top X Axis labels. This value changes proportionally to changes in the width of the viewport.

Default: 0.02

tmXTLabelFontColor

Sets the color of the top X-Axis labels.

Default: 1

tmXTLabelFontAspectF

Sets the ratio of height/width for the top X-Axis labels.

Default: 1.3125

tmXTLabelFontThicknessF

Sets the thickness of the lines used to draw the characters of the top X-Axis label. The value acts as a multiplier of a (device-dependent) unit thickness. This resource is ignored when the *tmXTLabelFont* specifies a filled font (font indexes 21-22, 25-26, 29-30, and 33-37).

Default: 1.0

tmXTLabelFontQuality

This resource of type NhlTFontQuality determines the quality of the font used to draw the top X-Axis label.

Default: `High`

tmXTLabelConstantSpacingF

If this resource has the value 0.0, the characters in the top X-Axis label will be proportionally spaced. If it has a value greater than 0.0, the distance from the beginning of one character to the beginning of the next character will be constant. The value of *tmXTLabelConstantSpacingF* sets this constant distance as a multiple of the width of a reference character box within which all characters of the font will fit. If *tmXTLabelConstantSpacingF* has the value 1.0, the constant distance will cause there to be almost no space between two wide characters (such as 'M' or 'W') placed next to each other. As *tmXTLabelConstantSpacingF* decreases from 1.0, characters will begin to overlap. As it increases from 1.0, characters become further separated from each other. Values less than 0.0 result in an error and are replaced with the default value.

Default: 0.0

tmXTLabelJust

Sets the justification point for all top X-Axis labels.

Default: `BottomCenter`

tmXTLabelAngleF

Sets the counterclockwise rotation of the X-Axis top labels.

Default: 0.0

tmXTLabelDirection

Sets the drawing direction of the X-Axis top labels.

Default: `Across`

tmXTLabelDeltaF

Sets an offset from the default location for the top tick mark labels. This value multiplied by the *tmXTFontHeightF* represents the amount, in NDC coordinates, that the top tick marks will be displaced from their default location in the Y direction. A positive value moves labels in the positive direction in NDC coordinates, and a negative value moves labels in the negative direction.

Default: 0.0

tmYUseLeft

Causes the right tick marks to use the resources for the left tick marks. The resources affected are

> *tmYRStyle*, *tmYRMode*, *tmYRMinorPerMajor*, *tmYRNoMinor*, *tmYRDataBottomF*, *tmYRDataTopF*, *tmYRTickStartF*, *tmYRTickEndF*, *tmYRMaxTicks*, *tmYRTickSpacing*, *tmYRValues*, *tmYRLabels*, *tmYRMinorValues*, *tmYRMajorThicknessF*, *tmYRMajorLineColor*, *tmYRMajorLengthF*, *tmYRMajorOutwardLengthF*, *tmYRMinorThicknessF*, *tmYRMinorLineColor*, *tmYRMinorLengthF*, *tmYRMinorOutwardLength*, *tmYRLabelFuncCode*, *tmYRConstantSpacingF*, *tmYRLabelFont*, *tmYRLabelFontHeightF*, *tmYRLabelFontColor*, *tmYRLabelFontAspectF*, *tmYRLabelFontThicknessF*, *tmYRLabelFontQuality*, *tmYRLabelAngleF*, *tmYRLabelDirection*, *tmYRLabelDeltaF*, *tmYRIrregularPoints*, *tmYRIrrTensionF*, *tmYRLabelStride*, *tmYRAutoPrecision*, *tmYRPrecision*, and *tmYRFormat*.

All other Right **TickMark** resources are unaffected.

Default: True

tmYLOn

Setting True turns on the left tick marks, and False turns them off.

Default: True

tmYROn

Setting True turns on the right tick marks, and False turns them off.

Default: True

tmYLLabelsOn

Setting True turns left labels on if *tmYLOn* is set. False turns them off.

Default: True

tmYRLabelsOn

Setting True turns right labels on if *tmYROn* is set. False turns them off.

Default: False

tmYLBorderOn

Setting True turns left border on and False turns it off. Tick marks will still be drawn unless *tmYLOn* is False.

Default: True

tmYRBorderOn

Setting True turns right border on and False turns it off. Tick marks will still be drawn unless *tmYROn* is False.

Default: True

tmYLMode

This enumerated resource of type NhlTTickMarkMode determines the method for specifying the spacing of the ticks and the contents of the labels along the left Y Axis. It has three possible settings:

`Automatic`:

Based on the values of *tmYLDataBottomF* and *tmYLDataTopF*, **TickMark** chooses a 'nice' spacing between the major tick marks, such that the number of major ticks does not exceed the value of *tmYLMaxTicks*. Labels are generated based on the current settings of *tmSciNoteCutoff*, *tmYLAutoPrecision*, *tmYLPrecision*, and *tmYLFormat*.

`Manual`:

Starting with the value *tmYLTickStartF*, **TickMark** sets major tick marks at intervals separated by a distance of *tmYLTickSpacingF* until *tmYLTickEndF* is exceeded. Of these tick marks, only those that fall between *tmYLDataBottomF* and *tmYLDataTopF* may appear along the left Y Axis. If *tmYLTickStartF* and *tmYLTickEndF* are not set **TickMark** issues an informational message and sets *tmYLTickStartF* to the minimum of *tmYLDataBottomF* and *tmYLDataTopF*, and it sets *tmYLTickEndF* to the maximum of *tmYLDataBottomF* and *tmYLDataTopF*. If *tmYLTickSpacingF* is not set, **TickMark** issues a warning message and *tmYLMode* reverts to `Automatic` mode. Labels are generated based on the current settings of *tmYLTickSpacingF*, *tmSciNoteCutoff*, *tmYLAutoPrecision*, *tmYLPrecision*, and *tmYLFormat*.

`Explicit`:

TickMark uses the array resource *tmYLValues* to determine the coordinate positions of the tick marks. Of these tick marks, only those that fall between *tmYLDataBottomF* and *tmYLDataTopF* may appear along the left Y Axis. If *tmYLValues* is not set, **TickMark** issues a warning message and *tmYLMode* reverts to `Automatic` mode. The corresponding element of the string array resource *tmYLLabels* determines the label used for each tick specified by *tmYLValues*. If *tmYLLabels* is not set, **TickMark** issues a warning and outputs no labels for the left Y Axis. You may optionally also set the array resource *tmYLMinorValues* to specify the locations of minor ticks along the axis.

Default: `Automatic`

tmYRMode

This enumerated resource of type NhlTTickMarkMode determines the method for specifying the spacing of the ticks and the contents of the labels along the right Y Axis. It has three possible settings:

`Automatic`:

Based on the values of *tmYRDataBottomF* and *tmYRDataTopF*, **TickMark** chooses a 'nice' spacing between the major tick marks, such that the number of major ticks does not exceed the value of *tmYRMaxTicks*. Labels are generated based on the current settings of *tmSciNoteCutoff*, *tmYRAutoPrecision*, *tmYRPrecision*, and *tmYRFormat*.

`Manual`:

Starting with the value *tmYRTickStartF*, **TickMark** sets major tick marks at intervals separated by a distance of *tmYRTickSpacingF* until *tmYRTickEndF* is exceeded. Of these tick marks, only those that fall between *tmYRDataBottomF* and *tmYRDataTopF* may appear along the right Y Axis. If *tmYRTickStartF* and *tmYRTickEndF* are not set, **TickMark** issues an informational message and sets *tmYRTickStartF* to the minimum of *tmYRDataBottomF* and *tmYRDataTopF*, and it sets *tmYRTickEndF* to the maximum of *tmYRDataBottomF* and *tmYRDataTopF*. If *tmYRTickSpacingF* is not set, **TickMark** issues a warning message and *tmYRMode* reverts to `Automatic` mode. Labels are generated based on the current settings of *tmYRTickSpacingF*, *tmSciNoteCutoff*, *tmYRAutoPrecision*, *tmYRPrecision*, and *tmYRFormat*.

`Explicit`:

TickMark uses the array resource *tmYRValues* to determine the coordinate positions of the tick marks. Of these tick marks, only those that fall between *tmYRDataBottomF* and *tmYRDataTopF* may appear along the right Y Axis. If *tmYRValues* is not set, **TickMark** issues a warning message and *tmYRMode* reverts to `Automatic` mode. The corresponding element of the string array resource *tmYRLabels* determines the label used for each tick specified by *tmYRValues*. If *tmYRLabels* is not set, **TickMark** issues a warning and outputs no labels for the right Y Axis. You may optionally also set the array resource *tmYRMinorValues* to specify the locations of minor ticks along the axis.

Default: `Automatic`

tmYLStyle

This enumerated resource of type NhlTTickMarkStyle sets the style of the **TickMark** left Y Axis. There are 5 styles:

`Linear`:

A linear coordinate system is set up along the Y Axis with the bottom boundary set to the value of *tmYLDataBottomF* and the top boundary set to the value of *tmYLDataTopF*. If *tmYLDataBottomF*and *tmYLDataTopF* are found to be equal when compared with a precision of 7 significant digits, a warning is issued and the left Y-Axis tick marks are turned off.

`Log`:

A logarithmic coordinate system is set up along the Y Axis with the bottom boundary set to the value of *tmYLDataBottomF* and the top boundary set to the value of *tmYLDataTopF*. If either *tmYLDataBottomF* or *tmYLDataTopF* is less than or equal to 0.0, a warning is issued and the left Y-Axis tick marks are turned off.

`Irregular`:

An irregular coordinate system is set up along the Y Axis based on the coordinate samples contained in the array resource *tmYLIrregularPoints*. If *tmYLIrregularPoints* is found to be invalid, a warning is issued and *tmYLStyle* reverts to `Linear`. If the data bounds *tmYLDataBottomF* or *tmYLDataTopF* are only partially within the range defined by the maximum and minimum elements of *tmYLIrregularPoints*, a warning is issued and, as appropriate, either *tmYLDataBottomF* or *tmYLDataTopF* is reset to the value of the maximum or minimum element of the *tmYLIrregularPoints*. If the range between *tmYLDataBottomF* and *tmYLDataTopF* is entirely outside the range of the *tmYLIrregularPoints* array, a fatal error is currently issued.

`Time`:

Not yet implemented.

`Geographic`:

Not yet implemented.

This resource may be intercepted or disabled by:

- **PlotManager**

Default: `Linear`

tmYLIrrTensionF

Specifies the value of the tension parameter applied to the spline approximation used to set up an irregular transformation based on the values in *tmYLIrregularPoints* array. This resource is only used if tmYLStyle is `Irregular`. Small values (less than ~1.0) imply a relaxed approximation; larg values (greater than ~5.0) imply a tight approximation.

This resource may be intercepted or disabled by:

- **PlotManager**

V4.1 Status Note 2

Default: 2.0

tmYRStyle

This enumerated resource of type NhlTTickMarkStyle sets the style of the **TickMark** right Y Axis. There are 5 styles:

`Linear`:

A linear coordinate system is set up along the Y Axis with the bottom boundary set to the value of *tmYRDataBottomF* and the top boundary set to the value of *tmYRDataTopF*. If *tmYRDataBottomF*and *tmYRDataTopF* are found to be equal when compared with a precision of 7 significant digits, a warning is issued and the right Y-Axis tick marks are turned off.

`Log`:

A logarithmic coordinate system is set up along the Y Axis with the bottom boundary set to the value of *tmYRDataBottomF* and the top boundary set to the value of *tmYRDataTopF*. If either *tmYRDataBottomF* or *tmYRDataTopF* is less than or equal to 0.0, a warning is issued and the left Y-Axis tick marks are turned off.

`Irregular`:

An irregular coordinate system is set up along the Y Axis based on the coordinate samples contained in the array resource *tmYRIrregularPoints*. If *tmYRIrregularPoints* is found to be invalid, a warning is issued and *tmYRStyle* reverts to `Linear`. If the data bounds *tmYRDataBottomF* or *tmYRDataTopF* are only partially within the range defined by the maximum and minimum elements of *tmYRIrregularPoints*, a warning is issued and, as appropriate, either *tmYRDataBottomF* or *tmYRDataTopF* is reset to the value of the maximum or minimum element of the *tmYRIrregularPoints*. If the range between *tmYRDataBottomF* and *tmYRDataTopF* is entirely outside the range of the *tmYRIrregularPoints* array, a fatal error is currently issued.

`Time`:

Not yet implemented.

`Geographic`:

Not yet implemented.

This resource may be intercepted or disabled by:

- **PlotManager**

Default: `Linear`

tmYRIrrTensionF

Specifies the value of the tension parameter applied to the spline approximation used to set up an irregular transformation based on the values in *tmYLIrregularPoints* array. This resource is only used if tmYLStyle is `Irregular`. Small values (less than ~1.0) imply a relaxed approximation; larg values (greater than ~5.0) imply a tight approximation.

This resource may be intercepted or disabled by:

- **PlotManager**

V4.1 Status Note 2

Default: 2.0

tmYLAutoPrecision

When True, this boolean resource causes **TickMark** to set the precision (number of significant digits) for the left tick mark labels based on the range of values used for the labels. In general, the value of *tmYLPrecision* is ignored. However, if *tmYLMode* is set to `Manual`, the precision is set such that all significant digits necessary to express the value of the *tmYLTickSpacingF* resource are used in determining the precision. In this case, the value of *tmYLPrecision* sets a cap on the number of significant digits used to express the value of *tmYLTickSpacingF*.

When set False, the value of *tmYLPrecision* controls the precision. However, if the significant digit conversion field (signaled by the '.' character) is set explicitly in the *tmYLFormat* string (according to the rules of the Floating Point Format Specification scheme),then the resources *tmYLAutoPrecision* and *tmYLPrecision* are both ignored, and the precision is controlled completely by the *tmYLFormat* specification.

Default: True

tmYLPrecision

If *tmYLAutoPrecision* is set False and the significant digit field in the *tmYLFormat* string (as specified by the Floating Point Format Specification scheme) has the dynamic attribute set on, this resource controls the precision used for the left tick mark labels. Assuming the default setting of the *tmYLFormat* resource, the value set for *tmYLPrecision* will be the number of digits used for the left tick mark label representing the value with the maximum absolute value. The other labels will have the same number of digits to the right of the decimal point, but may have fewer digits to the left of the decimal point. All labels will be rounded to the number of decimal places represented.

Default: 4

tmYLFormat

This string resource specifies, according to the HLU Floating Point Format Specification scheme, the format of the numbers used to label the major ticks along the left Y Axis. It applies only when *tmYLMode* is set to `Automatic` or `Manual`. Unless set explicitly, the significant digits conversion field is set dynamically based on the values of the *tmYLAutoPrecision* and *tmYLPrecision* resources. Likewise, the exponent switch length conversion field is set dynamically based on the value of the *tmSciNoteCutoff* resource. If the leftmost significant digit conversion field has the dynamic attribute set, the assumed leftmost significant digit is set dynamically for each label based on the largest absolute value used to generate the left labels. Combined with the zero format flag, this has the effect of causing all the labels to contain the same number of places to the right of the decimal point. The default format string also sets the "at-sign" ('@') format flag to force fractional values to include an initial '0' character preceding the decimal point. In addition, the exponent conversion field includes, by default, the 's' flag to cause exponents to be written using the superscript notational form.

If *tmYLStyle* is set to `Log`, exponential notation is forced on, regardless of the character used for the conversion specifier. Also the zero flag is turned off to ensure that the mantissas (which always evaluate to 1.0 because of the spacing enforced for `Log` style labels) are not turned on because of the need to fill out the significant digits on the right of the decimal point.

Default: "0@*+^sg"

tmYRAutoPrecision

When True, this boolean resource causes **TickMark** to set the precision (number of significant digits) for the right tick mark labels based on the range of values used for the labels. In general, the value of *tmYRPrecision* is ignored. However, if *tmYRMode* is set to `Manual`, the precision is set such that all significant digits necessary to express the value of the *tmYRTickSpacingF* resource are used in determining the precision. In this case, the value of *tmYRPrecision* sets a cap on the number of significant digits used to express the value of *tmYRTickSpacingF*.

When set False, the value of *tmYRPrecision* controls the precision. However, if the significant digit conversion field (signaled by the '.' character) is set explicitly in the *tmYRFormat* string (according to the rules of the Floating Point Format Specification scheme),then the resources *tmYRAutoPrecision* and *tmYRPrecision* are both ignored, and the precision is controlled completely by the *tmYRFormat* specification.

Default: True

tmYRPrecision

If *tmYRAutoPrecision* is set False and the significant digit field in the *tmYRFormat* string (as specified by the Floating Point Format Specification scheme) has the dynamic attribute set on, this resource controls the precision used for the right tick mark labels. Assuming the default setting of the *tmYRFormat* resource, the value set for *tmYRPrecision* will be the number of digits used for the right tick mark label representing the value with the maximum absolute value. The other labels will have the same number of digits to the right of the decimal point, but may have fewer digits to the left of the decimal point. All labels will be rounded to the number of decimal places represented.

Default: 4

tmYRFormat

This string resource specifies, according to the HLU Floating Point Format Specification scheme, the format of the numbers used to label the major ticks along the right Y Axis. It applies only when *tmYRMode* is set to `Automatic` or `Manual`. Unless set explicitly, the significant digits conversion field is set dynamically based on the values of the *tmYRAutoPrecision* and *tmYRPrecision* resources. Likewise, the exponent switch length conversion field is set dynamically based on the value of the *tmSciNoteCutoff* resource. If the leftmost significant digit conversion field has the dynamic attribute set, the assumed leftmost significant digit is set dynamically for each label based on the largest absolute value used to generate the right labels. Combined with the zero format flag, this has the effect of causing all the labels to contain the same number of places to the right of the decimal point. The default format string also sets the "at-sign" ('@') format flag to force fractional values to include an initial '0' character preceding the decimal point. In addition, the exponent conversion field includes, by default, the 's' flag to cause exponents to be written using the superscript notational form.

If *tmYRStyle* is set to `Log`, exponential notation is forced on, regardless of the character used for the conversion specifier. Also the zero flag is turned off to ensure that the mantissas (which always evaluate to 1.0 because of the spacing enforced for `Log` style labels) are not turned on because of the need to fill out the significant digits on the right of the decimal point.

Default: "0@*+^sg"

tmYMajorGrid

Setting True turns on the Y Axis grid. Grid lines will extend from the left tick marks to the right border. If the left ticks are turned off, the right tick marks will be extended to the left border.

Default: False

tmYMinorGrid

Setting True turns on the Y Axis grid. Grid lines will extend from the left tick marks to the right border. If the left ticks are turned off, the right tick marks will be extended to the left border.

Default: False

tmYMajorGridThicknessF

Sets the linewidth scale factor for the Y Axis major grid lines.

Default: 2.0

tmYMajorGridLineColor

Sets the color index for the Y Axis major grid lines.

Default: 1

tmYMajorGridLineDashPattern

Sets the dash pattern for the Y Axis major grid lines.

Default: 0

tmYMinorGridThicknessF

Sets the linewidth scale factor for the Y Axis minor grid lines.

Default: 1.0

tmYMinorGridLineColor

Sets the color index for the Y Axis minor grid lines.

Default: 1

tmYMinorGridLineDashPattern

Sets the dash pattern index for the Y Axis minor grid lines.

Default: 0

tmYLMinorPerMajor

Sets the number of minor tick marks per major tick mark for the left Y Axis. When using `Log` style tick marks, only the values 1, 4, and 8 are accepted as valid. If the `Explicit` mode is in effect, **TickMark** ignores this resource: you must instead specify the spacing of each minor tick individually using *tmYLMinorValues*. If you do not set this resource and the mode is `Automatic`, its value is determined dynamically depending on the major tick spacing, as follows: if the major spacing is 1 or 5 times some power of 10, *tmYLMinorPerMajor* is set to 4; if the spacing is 2 or 4 times a power of 10, it is set to 3; if the spacing is 3 times a power of 10, it is set to 2. Otherwise, if the mode is `Manual`, *tmYLMinorPerMajor* defaults to the value 3.

Default: <dynamic>

tmYRMinorPerMajor

Sets the number of minor tick marks per major tick mark for the right Y Axis. When using `Log` style tick marks, only the values 1, 4, and 8 are accepted as valid. If the `Explicit` mode is in effect, **TickMark** ignores this resource: you must instead specify the spacing of each minor tick individually using *tmYRMinorValues*. If you do not set this resource and the mode is `Automatic`, its value is determined dynamically depending on the major tick spacing, as follows: if the major spacing is 1 or 5 times some power of 10, *tmYRMinorPerMajor* is set to 4; if the spacing is 2 or 4 times a power of 10, it is set to 3; if the spacing is 3 times a power of 10, it is set to 2. Otherwise, if the mode is `Manual`, *tmYRMinorPerMajor* defaults to the value 3.

Default: <dynamic>

tmYLMinorOn

Turns on minor tick marks when set True; False turns them off.

Default: True

tmYRMinorOn

Turns on minor tick marks when set True; False turns them off.

Default: True

tmYLLabelStride

Used to skip tick mark labels. The value represents how many tick marks to skip between tick mark labels when drawing the tick mark labels. Will cause the tick mark object to draw first, then skip.

Default: 0

tmYRLabelStride

Used to skip tick mark labels. The value represents how many tick marks to skip between tick mark labels when drawing the tick mark labels. Will cause the tick mark object to draw first, then skip.

Default: 0

tmYLDataTopF

The data value that will be mapped to the top left corner of the Y Axis.

This resource may be intercepted or disabled by:

- **PlotManager**

Default: 0.0

tmYLDataBottomF

The data value that will be mapped to the bottom left corner of the Y axis.

This resource may be intercepted or disabled by:

- **PlotManager**

Default: 0.0

tmYLTickStartF

Sets the beginning of the tick marks. Must be less than *tmYLTickEndF*. If *tmYLTickStartF* is greater than the minimum data extent, no tick marks will be drawn between the minimum data extent and *tmYLTickStartF*.

Default: 0.0

tmYLTickEndF

Sets the ending of the tick marks. Must be greater than *tmYLTickStartF*. If *tmYLTickEndF* is less than the maximum data extent, no tick marks will be drawn between the maximum data extent and *tmYLTickEndF*.

Default: 0.0

tmYLMaxTicks

Used with `Automatic` mode. Sets the maximum number of tick marks that will be chosen.

Default: 7

tmYLTickSpacingF

Used for `Manual` mode ticks only. Specifies the spacing between major tick marks. If `Log` style is being used, this value must be the number of decades (i.e. 1 means ticks will be placed and labels at 10**1, 10**2, etc.)

Default: 0.0

tmYLIrregularPoints

Used for `Irregular` style ticks only. Specifies an array of data locations spaced at equal intervals along the left Y Axis. These points represent a discrete approximation to the desired coordinate system. These points must be strictly monotonically increasing or decreasing. Therefore, periodic coordinate systems can not be approximated using this resource. approximated using this resource.

This resource may be intercepted or disabled by:

- **PlotManager**

Default: NULL

tmYLValues

Used only for `Explicit` mode tick marks. This resource is an array of data locations at which tick marks will be drawn. The values must increase or decrease monotonically. If you do not set *tmYLValues* when in `Explicit` mode, **TickMark** issues a warning and *tmYLMode* reverts to

`Automatic` mode.

V4.1 Status Note 1

Default: NULL

tmYLLabels

Used only for `Explicit` mode tick marks. This resource is an array of strings to be used as tick mark labels for the locations in *tmYLValues*. If a string array element is set to NULL, no label will appear at the corresponding tick mark location. If you do not set *tmYLLabels* when in `Explicit` mode, **TickMark** issues a warning and outputs no labels for the left Y Axis.

Default: NULL

tmYLMinorValues

Used only for `Explicit` mode tick marks. This resource is an array of data locations at which the minor tick marks will be drawn. If you do not set *tmYLMinorValues* when in `Explicit` mode, minor tick marks will be absent from the left Y Axis.

Default: NULL

tmYLMajorThicknessF

Sets the linewidth scale factor to be used for the Y-Axis left major tick marks.

Default: 2.0

tmYLMajorLineColor

Sets the line color index to be used for drawing the Y-Axis left major tick marks.

Default: 1

tmYLMajorLengthF

Sets the length of the Y-Axis left major tick marks in NDC coordinates. This length scales with changes to the viewport width, unless you explicitly set *tmYLMajorLengthF* at the same time.

Default: <dynamic> -- initially 0.02 for a viewport width of 0.6

tmYLMajorOutwardLengthF

Sets the length of the major tick marks that will be drawn outside of the viewport. This value scales proportionally to changes in width.

Default: 0.0

tmYLMinorThicknessF

Sets the linewidth scale factor for the Y Axis minor tick marks.

Default: 1.0

tmYLMinorLineColor

Sets the color index for the Y Axis minor tick marks.

Default: 1

tmYLMinorLengthF

Sets minor tick mark length for the Y Axis minor tick marks. This length scales with changes to the viewport width, unless you explicitly set *tmYLMinorLengthF* at the same time.

Default: <dynamic> -- initially 0.01 for a viewport height of 0.6.

tmYLMinorOutwardLengthF

Sets the length of the Y-Axis left minor tick marks that will be drawn outside of the viewport. This value scales proportionally to changes in width.

Default: 0.0

tmYLLabelFuncCode

This resource of type NhlTCharacter sets the function code character for the Y-Axis left labels.

Default: **:**

tmYLLabelFont

Sets the font index to be used for the Y-Axis left tick mark labels.

Default: 0

tmYLLabelFontHeightF

Sets the height of the Y-Axis left labels in NDC coordinates. This value scales proportionally to changes in the height of the viewport.

Default: 0.02

tmYLLabelFontColor

Sets the color index for the left Y-Axis tick mark labels.

Default: 1

tmYLLabelFontAspectF

Sets the aspect ratio of height/width for each character of the Y-Axis left tick mark labels.

Default: 1.3125

tmYLLabelFontThicknessF

Sets the thickness of the lines used to draw the characters of the left Y-Axis label. The value acts as a multiplier of a (device-dependent) unit thickness. This resource is ignored when the *tmYLLabelFont* specifies a filled font (font indexes 21-22, 25-26, 29-30, and 33-37).

Default: 1.0

tmYLLabelFontQuality

This resource of type NhlTFontQuality determines the quality of the font used to draw the left Y-Axis label.

Default: `High`

tmYLLabelConstantSpacingF

If this resource has the value 0.0, the characters in the left Y-Axis label will be proportionally spaced. If it has a value greater than 0.0, the distance from the beginning of one character to the beginning of the next character will be constant. The value of *tmYLLabelConstantSpacingF* sets this constant distance as a multiple of the width of a reference character box within which all characters of the font will fit. If *tmYLLabelConstantSpacingF* has the value 1.0, the constant distance will cause there to be almost no space between two wide characters (such as 'M' or 'W') placed next to each other. As *tmYLLabelConstantSpacingF* decreases from 1.0, characters will begin to overlap. As it increases from 1.0, characters become further separated from each other. Values less than 0.0 result in an error and are replaced with the default value.

Default: 0.0

tmYLLabelJust

Sets font justification point for Y-Axis left tick mark labels.

Default: `CenterRight`

tmYLLabelAngleF

Sets the counterclockwise rotation for all Y-Axis tick mark labels.

Default: 0.0

tmYLLabelDirection

Sets the drawing direction for all Y-Axis tick mark labels.

Default: `Across`

tmYLLabelDeltaF

Sets an offset from the default location for the left tick mark labels. This value multiplied by the *tmYLFontHeightF* represents the amount, in NDC coordinates, that the left tick marks will be displaced from their default location in the X direction. A positive value moves labels in the positive direction in NDC coordinates, and a negative value moves labels in the negative direction.

Default: 0.0

tmYRDataTopF

The data value that will be mapped to the top right corner of the Y Axis.

This resource may be intercepted or disabled by:

- **PlotManager**

Default: 0.0

tmYRDataBottomF

The data value that will be mapped to the bottom right corner of the Y Axis.

This resource may be intercepted or disabled by:

- **PlotManager**

Default: 0.0

tmYRTickStartF

Sets the beginning of the tick marks. Must be less than *tmYRTickEndF*. If *tmYRTickStartF* is greater than the minimum data extent, no tick marks will be drawn between the minimum data extent and *tmYRTickStartF*.

Default: 0.0

tmYRTickEndF

Sets the ending of the tick marks. Must be greater than *tmYRTickStartF*. If *tmYRTickEndF* is less than the maximum data extent, no tick marks will be drawn between the maximum data extent and *tmYRTickEndF*.

Default: 0.0

tmYRMaxTicks

Used with `Automatic` mode. Sets the maximum number of tick marks that will be drawn.

Default: 7

tmYRTickSpacingF

Used for `Manual` mode ticks only. Specifies the spacing between major tick marks. If `Log` style is being used, this value must be the number of decades (i.e. 1 means ticks will be placed and labels at 10**1, 10**2, etc.)

Default: 0.0

tmYRIrregularPoints

Used for `Irregular` style ticks only. Specifies an array of data locations spaced at equal intervals along the right Y Axis. These points represent a discrete approximation to the desired coordinate system. These points must be strictly monotonically increasing or decreasing. Therefore, periodic coordinate systems can not be approximated using this resource.

This resource may be intercepted or disabled by:

- **PlotManager**

Default: NULL

tmYRValues

Used only for `Explicit` mode tick marks. This resource is an array of data locations at which tick marks will be drawn. The values must increase or decrease monotonically. If you do not set *tmYRValues* when in `Explicit` mode, **TickMark** issues a warning and *tmYRMode* reverts to `Automatic` mode.

V4.1 Status Note 1

Default: NULL

tmYRLabels

Used only for `Explicit` mode tick marks. This resource is an array of strings to be used as tick mark labels for the locations in *tmYRValues*. If a string array element is set to NULL, no label will appear at the corresponding tick mark location. If you do not set *tmYRLabels* when in `Explicit` mode (assuming *tmYUseLeft* is False), **TickMark** issues a warning and outputs no labels for the right Y Axis.

Default: NULL

tmYRMinorValues

Used only for `Explicit` mode tick marks. This resource is an array of data locations at which the minor tick marks will be drawn. If you do not set *tmYRMinorValues* when in `Explicit` mode (assuming *tmYUseLeft* is False), minor tick marks will be absent from the right Y Axis.

Default: NULL

tmYRMajorThicknessF

Sets the linewidth scale factor for the Y-Axis right major tick marks.

Default: 2.0

tmYRMajorLineColor

Sets the color index for the Y-Axis right major tick marks.

Default: 1

tmYRMajorLengthF

Sets the length of the Y-Axis right major tick marks in NDC coordinates. This length scales with changes to the viewport width, unless you explicitly set *tmYRMajorLengthF* (or *tmYLMajorLengthF* if *tmYUseLeft* is True) at the same time.

Default: <dynamic> -- initially 0.02 for a viewport width of 0.6

tmYRMajorOutwardLengthF

Sets the length of the Y-Axis right major tick marks that will be drawn outside of the current viewport. This value scales proportionally to changes in width.

Default: 0.0

tmYRMinorThicknessF

Sets the length of the Y Axis minor tick marks in NDC coordinates. This value scales proportionally to changes in width.

Default: 1.0

tmYRMinorLineColor

Sets the line color index for the right Y Axis minor tick marks.

Default: 1

tmYRMinorLengthF

Sets the length in NDC coordinates of the Y Axis minor tick marks. This length scales with changes to the viewport width, unless you explicitly set *tmYRMinorLengthF* (or *tmYLMinorLengthF* if *tmYUseLeft* is True) at the same time.

Default: <dynamic> -- initially 0.01 for a viewport height of 0.6.

tmYRMinorOutwardLengthF

Sets the length of the right Y Axis minor tick marks that will be drawn outside of the viewport. This value scales proportionally to changes in width.

Default: 0.0

tmYRLabelFuncCode

This resource of type NhlTCharacter sets the function code character for the Y-Axis right labels.

Default: **:**

tmYRLabelFont

Sets the font index for the right Y-Axis tick mark labels.

Default: 0

tmYRLabelFontHeightF

Sets the font height of the Y-Axis right labels in NDC coordinates. This value scales proportionally to changes in viewport height.

Default: 0.02

tmYRLabelFontColor

Sets the color index for right Y-Axis labels.

Default: 1

tmYRLabelFontAspectF

Sets the ratio of height/width for the right Y-Axis label characters.

Default: 1.3125

tmYRLabelFontThicknessF

Sets the thickness of the lines used to draw the characters of the right Y-Axis label. The value acts as a multiplier of a (device-dependent) unit thickness. This resource is ignored when the *tmYRLabelFont* specifies a filled font (font indexes 21-22, 25-26, 29-30, and 33-37).

Default: 1.0

tmYRLabelFontQuality

This resource of type NhlTFontQuality determines the quality of the font used to draw the right Y-Axis label.

Default: `High`

tmYRLabelConstantSpacingF

If this resource has the value 0.0, the characters in the right Y-Axis label will be proportionally spaced. If it has a value greater than 0.0, the distance from the beginning of one character to the beginning of the next character will be constant. The value of *tmYRLabelConstantSpacingF* sets this constant distance as a multiple of the width of a reference character box within which all characters of the font will fit. If *tmYRLabelConstantSpacingF* has the value 1.0, the constant distance will cause there to be almost no space between two wide characters (such as 'M' or 'W') placed next to each other. As *tmYRLabelConstantSpacingF* decreases from 1.0, characters will begin to overlap. As it increases from 1.0, characters become further separated from each other. Values less than 0.0 result in an error and are replaced with the default value.

Default: 0.0

tmYRLabelJust

Sets the justification point for the right Y-Axis labels.

Default: `CenterLeft`

tmYRLabelAngleF

Sets the counterclockwise rotation of all of the right Y-Axis labels.

Default: 0.0

tmYRLabelDirection

Sets the drawing direction for the right Y-Axis labels.

Default: `Across`

tmYRLabelDeltaF

Sets an offset from the default location for the right tick mark labels. This value multiplied by the *tmYRFontHeightF* represents the amount, in NDC coordinates, that the right tick marks will be displaced from their default location in the X direction. A positive value moves labels in the positive direction in NDC coordinates, and a negative value moves labels in the negative direction.

Default: 0.0

IrregularPlot class resource descriptions

The IrregularPlot class has no local resources.

IrregularTransformation class resource descriptions

trXAxisType

This resource of type NhlTAxisType sets the type of transformation used for the X Axis. It has three possible settings:

`IrregularAxis`

Based on the values of the *trXCoordPoints* resource (and possibly the *trXInterPoints* resource), an irregular coordinate system is defined along the X Axis. If *trXCoordPoints* is unset or found to be invalid, a warning will be issued and *trXAxisType* will be reset to its default value: `LinearAxis`.

`LogAxis`

A logarithmic coordinate system is established for the X Axis with boundaries defined by the values of *trXMinF* and *trXMaxF*. If *trXMinF* is less than or equal to 0.0, a warning will be issued and *trXAxisType* will be reset to its default value: `LinearAxis`.

`LinearAxis`

A linear coordinate system is established for the X Axis with boundaries defined by the values of *trXMinF* and *trXMaxF*.

This resource may be intercepted or disabled by:

- **ContourPlot**
- **StreamlinePlot**
- **VectorPlot**
- **XyPlot**

Default: `LinearAxis`

trXCoordPoints

If *trXInterPoints* is not set, this array resource represents uniformly spaced samples of a bounded irregularly spaced data coordinate system along the X Axis. The elements of the array must be arranged in a monotonically increasing or decreasing order. The coordinate system bounds are defined as the values of the minimum and maximum elements of the array: in other words, its first and last elements. If you superimpose on this coordinate system a regular coordinate system such that the left end is 0 and the right end is the number of elements in the array minus 1, then each succeeding array element value represents the coordinate value of the irregular coordinate system at successive integral coordinate points (0, 1, 2, ...) of the overlaid regular coordinate system. Note that these integral coordinates correspond directly to the array element index (with an offset of 1 when using a Fortran interface). Based on this mapping, the **IrregularTransformation** sets up both a forward and a reverse transformation.

trXCoordPoints is ignored unless *trXAxisType* is set to `IrregularAxis`. If the array contains less than two points or is non-monotonic, a warning will be issued and *trXAxisType* will be reset to its default value: `LinearAxis`.

V4.1 Status Note 2

This resource may be intercepted or disabled by:

- **ContourPlot**
- **StreamlinePlot**
- **VectorPlot**
- **XyPlot**

Default: NULL

trXInterPoints

This array resource, which, if set, must contain the same number of points as *trXCoordPoints*, adds another step to the mapping from regular to irregular coordinates. As in the case when *trXInterPoints* is not set, you superimpose a bounded regular coordinate system on the irregular coordinate system. Successive elements of *trXInterPoints*, like those of *trXCoordPoints*, must increase or decrease monotonically. However, the element values have a range defined by the bounds of the superimposed regular coordinate system (0 through array size - 1). Now the location of the data coordinate point specified by each successive element of the *trXCoordPoints* is determined, not by direct correlation with its array index, but by the value of the corresponding element of the *trXInterPoints* array.

trXInterPoints is ignored unless *trXAxisType* is set to `IrregularAxis`. If *trXInterPoints* is found to be invalid, **IrregularTransformation** issues a warning message and computes the coordinate transformation as if it had not been set.

This resource may be intercepted or disabled by:

- **ContourPlot**
- **StreamlinePlot**
- **VectorPlot**
- **XyPlot**

Default: NULL

trXTensionF

Specifies the value of the tension parameter applied to the spline approximation used to set up the irregular transformation for the X Axis coordinate system. Small values (less than ~1.0) imply a relaxed approximation; large values (greater than ~5.0) imply a tight approximation. Note that large values of this parameter (>~10.0) may cause the transformation to become unstable or even fail.

Default: 2.0

trXSamples

Specifies the number of sample points over the X-Axis coordinate system that **IrregularTransformation** is to use in order to determine the reverse transformation. More sample points (usually) increase the accuracy of the reverse transformation, but at the cost of greater processing time for every transformation calculated.

This resource may be intercepted or disabled by:

- **ContourPlot**
- **StreamlinePlot**
- **VectorPlot**
- **XyPlot**

Default: 9

trYAxisType

This resource of type NhlTAxisType sets the type of transformation used for the Y Axis. It has three possible settings:

`IrregularAxis`
: Based on the values of the *trYCoordPoints* resource (and possibly the *trYInterPoints* resource), an irregular coordinate system is defined along the Y Axis. If *trYCoordPoints* is unset or found to be invalid, a warning will be issued and *trYAxisType* will be reset to its default value: `LinearAxis`.

`LogAxis`
: A logarithmic coordinate system is established for the Y Axis with boundaries defined by the values of *trYMinF* and *trYMaxF*. If *trYMinF* is less than or equal to 0.0, a warning will be issued and *trYAxisType* will be reset to its default value: `LinearAxis`.

`LinearAxis`
: A linear coordinate system is established for the Y Axis with boundaries defined by the values of *trYMinF* and *trYMaxF*.

This resource may be intercepted or disabled by:

- **ContourPlot**
- **StreamlinePlot**
- **VectorPlot**
- **XyPlot**

Default: `LinearAxis`

trYCoordPoints

If *trYInterPoints* is not set, this array resource represents uniformly spaced samples of a bounded, irregularly spaced data coordinate system along the Y Axis. The elements of the array must be arranged in a monotonically increasing or decreasing order. The coordinate system bounds are defined as the values of the minimum and maximum elements of the array: in other words, its first and last elements. If you superimpose on this coordinate system a regular coordinate system such that the bottom end is 0 and the top end is the number of elements in the array minus 1, then each succeeding array element value represents the coordinate value of the irregular coordinate system at successive integral coordinate points (0, 1, 2, ...) of the overlaid regular coordinate system. Note that these integral coordinates correspond directly to the array element index (with an offset of 1 when using a Fortran interface). Based on this mapping, the **IrregularTransformation** sets up both a forward and a reverse transformation.

trYCoordPoints is ignored unless *trYAxisType* is set to `IrregularAxis`. If the array contains less than two points or is non-monotonic, a warning will be issued and *trYAxisType* will be reset to its default value: `LinearAxis`.

This resource may be intercepted or disabled by:

- **ContourPlot**
- **StreamlinePlot**
- **VectorPlot**
- **XyPlot**

V4.1 Status Note 2

Default: NULL

trYInterPoints

This array resource, which, if set, must contain the same number of points as *trYCoordPoints*, adds another step to the mapping from regular to irregular coordinates. As in the case when *trYInterPoints* is not set, you superimpose a bounded regular coordinate system on the irregular coordinate system. Successive elements of *trYInterPoints*, like those of *trYCoordPoints*, must increase or decrease monotonically. However, the element values have a range defined by the bounds of the superimposed regular coordinate system (0 through array size -1). Now the location of the data coordinate point specified by each successive element of the *trYCoordPoints* is determined, not by direct correlation with its array index, but by the value of the corresponding element of the *trYInterPoints* array.

trYInterPoints is ignored unless *trYAxisType* is set to `IrregularAxis`. If *trYInterPoints* is found to be invalid, **IrregularTransformation** issues a warning message and computes the coordinate transformation as if it had not been set.

This resource may be intercepted or disabled by:

- **ContourPlot**
- **StreamlinePlot**
- **VectorPlot**
- **XyPlot**

Default: NULL

trYTensionF

Specifies the value of the tension parameter applied to the spline approximation used to set up the irregular transformation for the Y Axis coordinate system. Small values (less than ~1.0) imply a relaxed approximation; large values (greater than ~5.0) imply a tight approximation. Note that large values of this parameter (>~10.0) may cause the transformation to become unstable or even fail.

Default: 2.0

trYSamples

Specifies the number of sample points over the Y-Axis coordinate system that **IrregularTransformation** is to use in order to determine the reverse transformation. More sample points (usually) increase the accuracy of the reverse transformation, but at the cost of greater processing time for every transformation calculated.

This resource may be intercepted or disabled by:

- **ContourPlot**
- **StreamlinePlot**
- **VectorPlot**

- XyPlot

Default: 9

LogLinPlot class resource descriptions

The LogLinPlot class has no local resources.

LogLinTransformation class resource descriptions

trXLog

If this boolean resource is True, the X-Axis coordinate system is logarithmic; otherwise it is linear. When this value is set True, both *trXMinF* and *trXMaxF* are required to be greater than 0.0.

This resource may be intercepted or disabled by:

- **ContourPlot**
- **StreamlinePlot**
- **VectorPlot**
- **XyPlot**

Default: False

trYLog

If this boolean resource is True, the Y-Axis coordinate system is logarithmic; otherwise it is linear. When this value is set True, both *trYMinF* and *trYMaxF* are required to be greater than 0.0.

This resource may be intercepted or disabled by:

- **ContourPlot**
- **StreamlinePlot**
- **VectorPlot**
- **XyPlot**

Default: False

Transformation class resource descriptions

trXMinF

Specifies a minimum X coordinate value that defines the lower bound of the X-Axis coordinate system. If *trXMinF* is greater than *trXMaxF*, the values will be exchanged before they are used to establish coordinate boundaries. If the axis is logarithmic, *trXMinF* must be greater than 0.0. If the axis is irregular, *trXMinF* is constrained to lie within the bounds of the irregular coordinate system as established by the X-Axis coordinate array.

This resource may be intercepted or disabled by:
- **ContourPlot**
- **StreamlinePlot**
- **VectorPlot**
- **XyPlot**
- **MapTransformation**

Default: 0.0

trXMaxF

Specifies a maximum X coordinate value that defines the upper bound of the X-Axis coordinate system. If *trXMaxF* is less than *trXMinF*, the values will be exchanged before they are used to establish coordinate boundaries. If the axis is irregular, *trXMaxF* is constrained to lie within the bounds of the irregular coordinate system as established by the X-Axis coordinate array.

This resource may be intercepted or disabled by:
- **ContourPlot**
- **StreamlinePlot**
- **VectorPlot**
- **XyPlot**
- **MapTransformation**

Default: 1.0

trXReverse

If the axis is linear or logarithmic, setting *trXReverse* True results in a coordinate system that increases from right to left, rather than in the usual direction from left to right. If the axis is irregular, setting *trXReverse* True results in a reversal of the direction of increasing coordinates as initially defined by the ordering of the elements of the X-Axis coordinate array.

This resource may be intercepted or disabled by:
- **ContourPlot**
- **StreamlinePlot**
- **VectorPlot**
- **MapTransformation**

Default: False

trYMinF

Specifies a minimum Y coordinate value that defines the lower bound of the Y-Axis coordinate system. If *trYMinF* is greater than *trYMaxF*, the values will be exchanged before they are used to establish coordinate boundaries. If the axis is logarithmic, *trYMinF* must be greater than 0.0. If the axis is irregular, *trYMinF* is constrained to lie within the bounds of the irregular coordinate system as established by the Y-Axis coordinate array.

This resource may be intercepted or disabled by:

- **ContourPlot**
- **StreamlinePlot**
- **VectorPlot**
- **XyPlot**
- **MapTransformation**

Default: 0.0

trYMaxF

Specifies a maximum Y coordinate value that defines the upper bound of the Y-Axis coordinate system. If *trYMaxF* is less than *trYMinF*, the values will be exchanged before they are used to establish coordinate boundaries. If the axis is irregular, *trYMaxF* is constrained to lie within the bounds of the irregular coordinate system as established by the Y-Axis coordinate array.

This resource may be intercepted or disabled by:

- **ContourPlot**
- **StreamlinePlot**
- **VectorPlot**
- **XyPlot**
- **MapTransformation**

Default: 1.0

trYReverse

If the axis is linear or logarithmic, setting this *trYReverse* True results in a coordinate system that increases from right to left, rather than in the usual direction from left to right. If the axis is irregular, setting *trYReverse* True results in a reversal of the direction of increasing coordinates as initially defined by the ordering of the elements of the Y-Axis coordinate array.

This resource may be intercepted or disabled by:

- **ContourPlot**
- **StreamlinePlot**
- **VectorPlot**
- **MapTransformation**

Default: False

trLineInterpolationOn

This resource controls the rendering of the lines between defined points for **XyPlot** and for graphics primitives drawn using **NhlDataPolyline** or **NhlDataPolygon**. If set False, lines between the points are drawn as straight lines in NDC space. If set True, points are interpolated between the defined points in order to map the line into the data space.

This resource is disabled by:

- **MapTransformation**

V4.1.1 Status Note 1

Default: False

TextItem class resource descriptions

txString

Specifies the string that the **TextItem** is to draw. The string may contain Text Function Codes that allow you to control typographical attributes such as subscripting and superscripting, change fonts within a string, embed newlines, etc.

Default: <dynamic>
If this resource is not set, then the default value for this resource will be the name of the **TextItem** object.

txPosXF

Is the X location of the justification point in NDC coordinates.

Default: <dynamic>
If the **View** resources are used to position the **TextItem** then this resource will be determined from that placement, otherwise its default is 0.0.

txPosYF

Is the Y location of the justification point in NDC coordinates.

Default: <dynamic>
If the **View** resources are used to position the **TextItem** then this resource will be determined from that placement, otherwise its default is 1.0.

txAngleF

Sets the rotation angle of the text around the justification point. The angle is counterclockwise and 0 degrees is horizontal on the screen.

Default: 0.0

txFont

When the *txFontQuality* resource has the value `High`, this resource of type NhlTFont specifies the font used for the **TextItem**. The fonts may be categorized into two types according to the way their characters are rendered: they are either stroked using lines or filled as areas. Only the stroked fonts are affected by the resource, *txFontThicknessF*, which sets the thickness of the stroking line. When *txFontQuality* is set to `Medium` or `Low`, the *txFont* resource is ignored.

The default font, the *pwritx_database* font (font index 0), differs from all other fonts in that it intrinsically contains many more characters than the other fonts (564 as opposed to somewhere between 97 and 128). A number of the Text Function Codes allow special access to the characters of the *pwritx_database* font.

You may use either the integer index or the string font name to specify a value for *txFont*. Note that whenever you change a font, the low-level NCAR Graphics library must read in the new font data from a file. The following table lists each fonts by index and name, and also indicates the rendering method used for the font:

```
+--------------------------------------------------------+
|                HLU Fonts                               |
```

INDEX	TYPE	NAME
0	stroked	pwritx_database
1	stroked	default
2	stroked	cartographic_roman
3	stroked	cartographic_greek
4	stroked	simplex_roman
5	stroked	simplex_greek
6	stroked	simplex_script
7	stroked	complex_roman
8	stroked	complex_greek
9	stroked	complex_script
10	stroked	complex_italic
11	stroked	complex_cyrillic
12	stroked	duplex_roman
13	stroked	triplex_roman
14	stroked	triplex_italic
15	stroked	gothic_german
16	stroked	gothic_english
17	stroked	gothic_italian
18	stroked	math_symbols
19	stroked	symbol_set1
20	stroked	symbol_set2
21	filled	helvetica
22	filled	helvetica-bold
25	filled	times-roman
26	filled	times-bold
29	filled	courier
30	filled	courier-bold
33	filled	greek
34	filled	math-symbols

35	filled	text-symbols
36	filled	weather1
37	filled	weather2

Default: 0

txJust

This resource of type NhlTJustification sets the justification point of the **TextItem**.

Default: `CenterCenter`

txFontQuality

This resource of type NhlTFontQuality determines the quality of the font used to draw the **TextItem**. There are three choices:

`High`
: Draw characters using any of the stoked or filled fonts.

`Medium`
: Draw characters using a 94 character stroked font that is simpler than the `High` quality fonts, resulting in somewhat smaller metafiles. The *txFont* resource is ignored.

`Low`
: The characters are output as a string into the metafile. The quality of the output therefore depends on the fonts available to the metafile translator. When NCAR translators are used the font quality is similar to that of `Medium` text. The *txFont* resource is ignored.

Default: `High`

txFontColor

Integer value that is an index into parent workstations color map.

Default: 1

txFontHeightF

Height of text from bottom of average character to top of average character in NDC coordinates.

Default: .05

txFontAspectF

This resource sets the aspect ratio, defined as the height divided by the width of a reference character box within which all characters of the font will fit. Its value determines the width of the characters relative to *txFontHeightF*. If given the value 1.0, wide characters such as 'M' will be approximately square in shape. Values increasing from 1.0 result in thinner characters, while values decreasing from 1.0 result in wider characters.

Default: 1.3125

txFontThicknessF

Sets the thickness of the lines used to draw the characters of the **TextItem**. The value acts as a multiplier of a (device-dependent) unit thickness. This resource is ignored when the *txFont* specifies a filled font (font indexes 21-22, 25-26, 29-30, and 33-37).

Default: 1.0

txConstantSpacingF

If this resource has the value 0.0, **TextItem** draws characters with proportional spacing; in other words, each character is spaced according to its width relative to other characters in the font. If *txConstantSpacingF* has a value greater than 0.0, the distance from the beginning of one character to the beginning of the next character will be constant. The value of *txConstantSpacingF* sets this constant distance as a multiple of the width of a reference character box within which all characters of the font will fit. If *txConstantSpacingF* has the value 1.0, the constant distance will cause there to be almost no space between two wide characters such as 'M' placed next to each other. As *txConstantSpacingF* decreases from 1.0, characters will begin to overlap. As it increases from 1.0, characters become further separated from each other. Values less than 0.0 result in an error and are replaced with the default value.

Default: 0.0

txDirection

This resource of type NhlTTextDirection specifies the direction of the text drawn by the **TextItem**. There are three choices of which two are actually the same:

`Down`
: Each character is placed below the previous character in the text string.

`Up`
: Each character is placed below the previous character in the text string.

`Across`
: Each character is placed to the right of the previous character in the text string.

These descriptions apply before any rotation due to *txAngleF* is applied to the **TextItem**.

Default: `Across`

txFuncCode

Specifies the character used to delimit Text Function Codes embedded in the **TextItem** string.

Default: **:**

txPerimOn

This boolean resource that determines whether the **TextItem** draws a box around the perimeter of the text. The box may or may not have a solid fill background.

Default: False

txPerimColor

When *txPerimOn* is True, *txPerimColor* specifies the HLU color index used for the line around the perimeter of the text.

Default: `Foreground`

txPerimThicknessF

When *txPerimOn* is True, *txPerimThicknessF* specifies the thickness of the perimeter line around the **TextItem**. The value acts as a multiplier of a (device-dependent) unit thickness.

Default: 1.0

txPerimDashPattern

When *txPerimOn* is True, *txPerimDashPattern* specifies the HLU index of the dash pattern used to draw the perimeter line around the **TextItem**.

Default: 0

txPerimDashLengthF

When *txPerimOn* is True, *txPerimDashLengthF* specifies the length of each segment of the dash pattern used to draw the perimeter line around the **TextItem**.

Default: 0.15

txPerimSpaceF

When *txPerimOn* is True, *txPerimSpaceF* determines the spacing or margin between the text of the **TextItem** and perimeter outline as a fraction of the current *txFontHeightF*.

Default: 0.5

txBackgroundFillColor

When *txPerimOn* is True, *txBackgroundFillColor* sets the background color used as solid fill of the box defined by the **TextItem** perimeter. To leave the box unfilled, set *txBackgroundFillColor* to `Transparent` (-1).

Default: `Transparent`

VectorPlot class resource descriptions

vcVectorFieldData

Specifies the id of a **VectorField** data object. There is no default for this resource; it is the only resource that must be set in order for the **VectorPlot** object to draw a plot. You *may* create a **VectorPlot** object without setting the *vcVectorFieldData*, and auxiliary annotations such as tick marks and titles may appear as the result of a draw, but the **VectorPlot** itself will not show up. The **VectorField** object can provide either regularly spaced or irregular rectangular gridded data to the **VectorPlot** object, and it provides a number of resources for controlling the ingestion of the raw data.

Default: none

vcScalarFieldData

Specifies the id of a **ScalarField** data object. If this resource is set with a valid **ScalarField** object *and* the resource *vcUseScalarArray* is set True, **VectorPlot** will use the scalar array data for coloring the arrows used to render the vector field plot. In order for **VectorPlot** to consider the **ScalarField** object valid, it must find that it has the same number of elements along each dimension as are found in the **VectorField** object.

Note that currently the number of elements is checked only after strides and subsetting are applied. The data coordinate extents set for the **ScalarField** object are ignored. For now, the data coordinate extents of the processed **ScalarField** data are assumed to coincide with the data coordinate extents of the **VectorField** data.

Default: none

vcMapDirection

This resource controls whether the vector direction is mapped into the same coordinate space as the vector location, or whether it is rendered in a locally uniform cartesian coordinate space. This resource has an effect whenever a non-uniform transformation is in effect. These include most of the **MapTransformation** transformations and **IrregularTransformation** transformations. Also included are logarithmic transformations provided by the **LogLinTransformation** and even linear transformations when the X and Y unit sizes are different.

Default: True

vcPositionMode

This resource of type NhlTVectorPositionMode specifies how the vector arrow is positioned in the vector field plot relative to the actual data location. There are three possibilites:

`ArrowHead`
: The head of the arrow is placed at the data location.

`ArrowCenter`
: The center of the arrow is placed at the data location.

`ArrowTail`
: The tail of the arrow is placed at the data location.

Default: `ArrowCenter`

vcVectorDrawOrder

This resource of type NhlTDrawOrder determines when the vector arrows are drawn relative to the drawing of other elements of a composite plot. There are three choices:

`PreDraw`
: Draw vector arrows before the standard draw phase; the arrows will be overlaid by any subsequently drawn elements.

`Draw`
: Draw vector arrows during the standard draw phase; the arrows will overlay any elements drawn during the predraw phase, but will underlie elements drawn during the postdraw phase.

`PostDraw`
: Draw vector arrows after the standard draw; the arrows will overlay any elements drawn during the predraw and draw phases.

Default: `Draw`

vcGlyphStyle

This resource selects the style of glyph used to represent the vector magnitude and direction. There are three choices:

`LineArrow`
: Vectors are represented using an arrow-shaped polyline pointing in the direction of the vector. The arrow length and head size may vary with the magnitude of the vector. Resources prefixed by *vcLineArrow...* have an effect only when *vcGlyphStyle* is set to this value.

`FillArrow`
: Vectors are represented using a polygon and possibly a polyline edge that point in the direction of the vector. The length, width, and various parameters affecting the head size and overall shape may vary with the magnitude of the vector. Resources prefixed by *vcFillArrow...* have an effect only when *vcGlyphStyle* is set to this value.

`WindBarb`
: Vectors are represented using a standard wind barb glyph, composed of a shaft parallel to the

vector direction, and pennants and/or ticks spaced at even intervals along the shaft starting at the end nearest the direction from which the flow is coming. (For the purposes of the *vcPositionMode* resource, this end is the "tail" of the wind barb.) If the velocity is less than 2.5 in magnitude, a circle is drawn instead of the barbed shaft. Otherwise half ticks represent 5 units of magnitude, full ticks represent 10 units, and pennants represent 50 units. By convention, the units usually represent knots. The pennants are drawn using a filled polygon, while the ticks, the shaft, and the calm circle are all rendered with polylines. Unlike the other glyph styles, wind barbs maintain a basically uniform length for all magnitudes. Resources prefixed by *vcWindBarb...* have an effect only when *vcGlyphStyle* is set to this value.

This resource overrides and eventually is intended to replace the resource *vcFillArrowsOn*. *vcFillArrowsOn* can still be used to choose between line arrows and fill arrows, as long as you do not set *vcGlyphStyle* at the same time.

Default: `LineArrow`

vcMinDistanceF

This resource specifies a minimum distance in NDC space that is to separate the data locations of neighboring vectors. Vectors are elimininated from the plot until there are no vectors closer than the specified distance. Note that if the realized length of a vector glyph is long enough, it is still possible that it may overlap neighboring glyphs. However, the grid point locations are guaranteed to be no closer than the distance specified. When this resource has its default value, 0.0, the thinning algorithm is not applied.

Default: 0.0

vcMinMagnitudeF

This resource specifies a minimum magnitude for elements of the vector field. Vectors with magnitudes less than this value will not be rendered in the vector field plot.

Default: 0.0

vcMaxMagnitudeF

This resource specifies a maximum magnitude for elements of the vector field. Vectors with magnitudes greater than this value will not be rendered in the vector field plot.

Default: 0.0

vcRefMagnitudeF

This resource specifies the magnitude used as the reference magnitude used for the vector field plot. Vectors equal in magnitude to the reference magnitude will be rendered at the length specified by the resource *vcRefLengthF*. If *vcRefMagnitudeF* has its default value, 0.0, the maximum magnitude in the vector field will be used as the reference magnitude.

Default: 0.0

vcRefLengthF

If *vcGlyphStyle* is set to `LineArrow` or `FillArrow`, this resource specifies, in units of NDC, the length used to render vectors with a magnitude equal to the reference magnitude, as specified by *vcRefMagnitudeF*. If *vcGlyphStyle* is set to `WindBarb`, it specifies the length of the wind barb shaft plus the projected length of a full tick along the shaft axis. If *vcRefLengthF* is not specified, its value is dynamically calculated based on the number of elements in the vector field and the size of the **VectorPlot** viewport.

Default: <dynamic>

vcMinFracLengthF

If *vcGlyphStyle* is set to `LineArrow` or `FillArrow`, this resource specifies, as a fraction of the reference length (*vcRefLengthF*), the length used to render the minimum magnitude vector. If *vcMinMagnitudeF* is greater than 0.0, its value is used as the minimum magnitude; otherwise the magnitude of the smallest vector appearing in the plot is used as the minimum magnitude. The lengths of the remaining vectors are determined by scaling proportionally between this length and the reference length. Setting *vcMinFracLengthF* to the value 1.0 causes all vectors to be drawn with the same length. If *vcMinFracLengthF* has its default value, 0.0, the vector length is strictly proportional to its magnitude.

If *vcGlyphStyle* is set to `WindBarb`, the value of *vcMinFracLengthF* is effectively forced to the value 1.0, and its set value is ignored.

Default: 0.0

vcLevels

An array of floats containing values that determine the color of the vector arrows in the vector field plot. If *vcScalarFieldData* is valid and the resource *vcUseScalarArray* is set True, the level values are based on the scalar array data; otherwise the level values are based on the magnitudes of the vector elements in the vector field. If the *vcLevelSelectionMode* is `ExplicitLevels`, you may set these values yourself. Otherwise, the **VectorPlot** object sets the elements of this array.

Default: <dynamic>

vcLevelCount

A read-only resource set to the actual number of levels chosen during the level selection process.

Default: 16

vcLevelSelectionMode

This enumerated resource of type NhlTLevelSelectionMode provides four methods for selecting the levels used to color the vector arrows:

`AutomaticLevels`

Ordinarily this mode determines vector levels by picking a spacing value from a set of relatively "round" numbers scaled by powers of 10 to the range of the data. This set of numbers is as follows: 1.0, 2.0, 2.5, 4.0, 5.0. The number of levels chosen will be as close as possible to the value of *vcMaxLevelCount* without exceeding it. Once the spacing is chosen, the minimum vector level is set to the value of the least multiple of the spacing greater than the minimum data value. Likewise the maximum vector level becomes the greatest multiple of the spacing less than the maximum data value. Based on these values, **VectorPlot** sets the resources *vcLevelSpacingF*, *vcMinLevelValF*, and *vcMaxLevelValF* appropriately.

On the other hand, if you explicitly set the resource *vcLevelSpacingF* to a valid value greater than 0.0 and less than the range of the data, it will be used as the interval spacing. The minimum and maximum levels are calculated as before. If as a consequence, *vcMaxLevelCount* is less than the number of levels so specified, it will be set to the number of levels actually needed. However, if the choice of spacing causes the absolute maximum number of levels (currently 255) to be exceeded, **VectorPlot** will issue a warning message and recalculate the spacing as previously described.

In any case, **VectorPlot** sets the elements of the array resource *vcLevels* to the values of the vector levels chosen and the read-only resource *vcLevelCount* to the number of levels.

`ManualLevels`

`ManualLevels` mode bases the choice of vector levels on the values of the resources *vcLevelSpacingF*, *vcMinLevelValF*, and *vcMaxLevelValF*. Starting at *vcMinLevelValF*, vector levels are created at intervals spaced by the value of *vcLevelSpacingF* until *vcMaxLevelValF* is reached. The final vector level will always be *vcMaxLevelValF*. **VectorPlot** sets elements of the array resource *vcLevels* to the values of each vector level chosen and the read-only resource *vcLevelCount* to the number of levels. If the current value of *vcMaxLevelCount* is less than *vcLevelCount*, it is reset to the value of *vcLevelCount*. However, if the level count would exceed the absolute maximum number of levels, currently 255, **VectorPlot** issues a warning and chooses a new value of *vcLevelSpacingF* based on the value of *vcMaxLevelCount*.

If you choose `ManualLevels` selection mode when the **VectorPlot** object is created, and do not set *vcMinLevelValF*, **VectorPlot** will choose levels as if you had set `AutomaticLevels` mode. If you set *vcMinLevelValF* only, a default spacing is used, and the value of *vcMaxLevelValF* is determined as it would be for `AutomaticLevels` mode.

`ExplicitLevels`

This mode allows you to specify the value of each vector level by explicitly setting the contents of the *vcLevels* array. If you choose `ExplicitLevels` selection mode when creating a **VectorPlot** object, but do not specify the contents of the *vcLevels* array, **VectorPlot** will choose levels as if you had specified `AutomaticLevels` mode. Thereafter, when you set `ExplicitLevels` mode, **VectorPlot** uses the current contents of *vcLevels*, whether or not you set it explicitly.

If the number of elements in *vcLevels* exceeds the absolute maximum number of levels (currently 255), **VectorPlot** issues a warning and the mode defaults to `AutomaticLevels`. Note that **VectorPlot** always sorts the elements of *vcLevels* into a monotonically increasing sequence. After sorting, *vcMinLevelValF* is set equal to the value of first element of *vcLevels*, and *vcMaxLevelValF* is set to the value of the last element. *vcLevelSpacingF* is set to the average value of the spacing separating each level.

`EqualSpacedLevels`

For this mode, **VectorPlot** divides the range spanning the dataset minimum and maximum values into *vcMaxLevelCount*+1 equally spaced intervals. *vcLevelSpacingF* is set to the value of this interval. *vcMinLevelValF* is set to the value of the dataset minimum plus the value of *vcLevelSpacingF*. *vcMaxLevelSpacingF* is set to the value of the dataset maximum minus the value of *vcLevelSpacingF*. This results in *vcMaxLevelCount* levels; therefore **VectorPlot** sets the read-only resource *vcLevelCount* equal to *vcMaxLevelCount*.

Default: `AutomaticLevels`

vcMaxLevelCount

When the *vcLevelSelectionMode* is `AutomaticLevels` and *vcLevelSpacingF* is not explicitly set, **VectorPlot** picks a number of levels less than or equal to the current value of *vcMaxLevelCount*. If the *vcLevelSelectionMode* is `EqualSpacedLevels`, **VectorPlot** picks exactly *vcMaxLevelCount* levels. If *vcMaxLevelCount* exceeds the absolute maximum level count allowed by **VectorPlot** (currently 255), a warning is issued and the value is reset to this maximum. If *vcLevelSelectionMode* is `ManualLevels` or `ExplicitLevels` or `AutomaticLevels` with *vcLevelSpacingF* explicitly set, **VectorPlot** sets *vcMaxLevelCount* to the number of levels actually used if this number is greater than the current value of *vcMaxLevelCount*.

Default: 16

vcLevelSpacingF

When the *vcLevelSelectionMode* is `ManualLevels` or when the *vcLevelSelectionMode* is `AutomaticLevels`, and *vcLevelSpacingF* is explicitly set, *vcLevelSpacingF* determines the spacing between vector intervals. Otherwise, the **VectorPlot** object sets the value of *vcLevelSpacingF* based on the vector levels actually chosen. When the *vcLevelSelectionMode* is `ExplicitLevels`, *vcLevelSpacingF* will be set to the arithmetic average of the spacing between levels.

Default: 5.0

vcMinLevelValF

When the *vcLevelSelectionMode* is `ManualLevels`, the value of *vcMinLevelValF*, if set, determines the lowest vector level. Otherwise, **VectorPlot** sets the *vcMinLevelValF* to a value equal to lowest vector level picked.

Default: <dynamic>

vcMaxLevelValF

When the *vcLevelSelectionMode* is `ManualLevels`, the value of *vcMaxLevelValF*, if set, determines the highest vector level. Otherwise, **VectorPlot** sets the *vcMaxLevelValF* to a value equal to highest vector level picked.

Default: <dynamic>

vcLevelColors

If *vcGlyphStyle* is set to `LineArrow` and *vcMonoLineArrowColor* is False, this resource controls the color of the line-drawn vector arrows. If *vcGlyphStyle* is set to `FillArrow`, then this resource may control the fill and/or the edge color of the filled vector arrows, depending on the values of the resources *vcMonoFillArrowFillColor* and *vcMonoFillArrowEdgeColor*. If *vcGlyphStyle* is set to `WindBarb` and *vcMonoWindBarbColor* is False, this resource controls the color of the wind barb glyphs. Each element of this array resource of type NhlTColorIndexGenArray specifies a color index for vectors whose associated scalar data value is less than the corresponding element of the *vcLevels* array. Note that there is always one more color than there are vector levels. The first element of *vcLevelColors* specifies a color index for any vector with an associated data value less than the value of *vcMinLevelValF*. The highest currently used element of *vcLevelColors* specifies a color for vectors with associated data values greater than the value of *vcMaxLevelValF*. If the array is not set explicitly, it will default to a set of color indexes sequentially incremented beginning with color index 1. If the array currently contains fewer elements than *vcLevelCount* plus 1, more elements will be added to the array and given values according to the default color assignment scheme; existing elements with valid color index values will not be modified.

Default: <dynamic>

vcUseScalarArray

If this boolean resource is set True, the resource *vcScalarFieldData* is set with a valid **ScalarField** object, and the appropriate **VectorPlot** resources are set to enable multi-colored vector glyphs, then the **ScalarField** data are used to select the level (and hence the color index) assigned to each glyph. Otherwise, when multi-colored vector glyphs are enabled, vector coloring will be based on the magnitude of the vectors.

Default: False

vcScalarMissingValColor

When *vcUseScalarArray* is True, this resource sets the HLU index of the fill or line color used to draw vectors whose associated scalar value is missing. If *vcScalarMissingValColor* has the value `Transparent` (-1), the vector will not be drawn at all.

Default: `Foreground`

vcMonoLineArrowColor

When this resource is set True, line-drawn vector arrows are colored using a single HLU color index, as specified by the value of the scalar resource *vcLineArrowColor*. When False, the elements of the array resource *vcLevelColors* are used to set the color of each vector line individually based on the scalar quantity associated with the vector. This resource has an effect only when *vcGlyphStyle* is set to `LineArrow`.

Default: True

vcLineArrowColor

When *vcMonoLineArrowColor* is set True, this resource of type NhlTColorIndex sets a uniform color index for all lines used to draw vector arrows. If set to `Transparent` (-1), the line color index will be constrained to the value `Foreground` (1). This resource has an effect only when *vcGlyphStyle* is set to `LineArrow`.

Default: `Foreground`

vcLineArrowThicknessF

This resource sets the thickness of the line used to draw vector line arrows. The value acts as a multiplier of a (device-dependent) unit thickness. This resource has an effect only when *vcGlyphStyle* is set to `LineArrow`.

Default: 1.0

vcLineArrowHeadMinSizeF

Specifies a minimum length for the lines that form the point of the arrowhead of line-drawn vectors, as a fraction of the viewport width. Normally the arrowhead size is scaled proportionally to the length of the vector. This resource allows you to ensure that the arrowhead will remain recognizable even for very short vectors. If *vcArrowHeadMinSizeF* is set to the same value as *vcArrowHeadMaxSizeF*, all arrowheads in the vector plot will be drawn at the same size. Setting both resources to 0.0 will cause the arrows to be drawn without any heads at all. This resource has an effect only when *vcGlyphStyle* is set to `LineArrow`.

Default: 0.005

vcLineArrowHeadMaxSizeF

Specifies a maximum length for the lines that form the point of the arrowhead of line-drawn vectors, as a fraction of the viewport width. Normally the arrowhead size is scaled proportionally to the length of the vector. This resource allows you to ensure that the arrowheads do not become excessively large for high magnitude vectors. If *vcArrowHeadMinSizeF* is set to the same value as *vcArrowHeadMaxSizeF*, all arrowheads in the vector plot will be drawn at the same size. Setting both resources to 0.0 will cause the arrows to be drawn without any heads at all. This resource has an effect only when *vcGlyphStyle* is set to `LineArrow`.

Default: 0.05

vcFillArrowsOn

vcGlyphStyle overrides and eventually is intended to replace this resource. However, if you set *vcFillArrowsOn* and do not at the same time set *vcGlyphStyle*, you can use it to select between the `LineArrow` and `FillArrow` glyph styles. When set True, fill arrows are selected; when set False, line arrows are selected.

Default: False

vcMonoFillArrowFillColor

When set True, the fill color for all filled vector arrows is set to a single HLU color index, as specified by the value of the scalar resource *vcFillArrowFillColor*. When False, the elements of the array resource *vcLevelColors* specify the fill color of each arrow based on the scalar quantity associated with the vector. This resource has an effect only when *vcGlyphStyle* is set to `FillArrow`.

Default: True

vcFillArrowFillColor

When *vcMonoFillArrowFillColor* is set True, this resource of type NhlTColorIndex sets a uniform fill color index for all filled vector arrows. If set to `Transparent` (-1), the arrow will not be filled at all and only the perimeter will be drawn. This resource has an effect only when *vcGlyphStyle* is set to `FillArrow`.

Default: `Foreground`

vcFillOverEdge

If this boolean resource is set True, the perimeter outline of a filled vector arrow is drawn first, underneath the fill. In this case, the line must be drawn using a line thickness greater than 1.0 in order for the line to appear. The advantage of drawing the line underneath is that the full extent of the fill appears; when the line is drawn on top of the fill using a different color index, the fill color may be partially or completely obscured, especially for small vector arrows. This resource has an effect only when *vcGlyphStyle* is set to `FillArrow`.

Default: True

vcMonoFillArrowEdgeColor

When set True, the edge color for all filled vector arrows is set to a single HLU color index, as specified by the value of the scalar resource *vcFillArrowEdgeColor*. When False, the elements of the array resource *vcLevelColors* specify the edge color of each arrow based on the scalar quantity associated with the vector. This resource has an effect only when *vcGlyphStyle* is set to `FillArrow`.

Default: True

vcFillArrowEdgeColor

When *vcMonoFillArrowEdgeColor* is set True, this resource of type NhlTColorIndex sets a uniform color index for the edges of all filled vector arrows. If set to `Transparent` (-1), the edge line will not be drawn at all for filled vectors, assuming the fill color index is set to a value other than `Transparent`. This resource has an effect only when *vcGlyphStyle* is set to `FillArrow`.

Default: `Background`

vcFillArrowEdgeThicknessF

This resource sets the thickness of the lines used to form the edges of filled vector arrows. The value acts as a multiplier of a (device-dependent) unit thickness. If *vcFillOverEdge* is set True, note that the edge will not fully appear unless it is drawn using thickness greater than 1.0. This resource has an effect only when *vcGlyphStyle* is set to `FillArrow`.

Default: 2.0

vcFillArrowWidthF

This resource specifies the width of a filled vector arrow drawn at the reference length as a fraction of *vcRefLengthF*. (See figure.) If *vcFillArrowMinFracWidthF* has the value 0.0, then the ratio of the arrow width to the arrow length will be invariant for all arrows in the plot. The value of *vcFillArrowWidthF* is constrained to the range 0.0 through 1.0. This resource has an effect only when *vcGlyphStyle* is set to `FillArrow`.

Default: 0.1

vcFillArrowMinFracWidthF

This resource specifies the width of a filled arrow drawn at the minimum length, as a fraction of the width specified by *vcFillArrowWidthF*. (See figure.) If given the value 1.0, all arrows in the plot will be drawn at the same width, regardless of their length. The value of *vcFillArrowMinFracWidthF* is constrained to the range 0.0 through 1.0. This resource has an effect only when *vcGlyphStyle* is set to `FillArrow`.

Default: 0.25

vcFillArrowHeadXF

This resource specifies the distance from the point of the arrowhead of a filled vector arrow drawn at the reference length to its back tips along the centerline of the arrow. The value represents a fraction of the value of *vcRefLengthF* and determines the X component of the arrowhead size. (See figure.) If *vcFillArrowMinFracXF* has the value 0.0, then the ratio of the X component of the arrowhead size to the arrow length will be invariant for all vectors in the plot. The value of *vcFillArrowHeadXF* is constrained to the range 0.0 through 2.0. This resource has an effect only when *vcGlyphStyle* is set to `FillArrow`.

Default: 0.36

vcFillArrowHeadMinFracXF

This resource specifies the X component of the arrowhead size for a filled arrow drawn at the minimum length, as a fraction of the component length specified by *vcFillArrowHeadXF*. (See figure.) If given the value 1.0, the arrowhead X component will be the same for all arrows in the plot, regardless of their length. The value of *vcFillArrowHeadMinFracXF* is constrained to the range 0.0 through 1.0. This resource has an effect only when *vcGlyphStyle* is set to `FillArrow`.

Default: 0.25

vcFillArrowHeadInteriorXF

This resource specifies the distance from the point of the arrowhead of a filled vector arrow drawn at the reference length to the point where the arrowhead joins with the line extending to the tail of the arrow. The value represents a fraction of the value of *vcRefLengthF*. (See figure.) This distance is adjusted proportionally to the X component of the arrowhead size for all vector arrows. The value of *vcFillArrowHeadInteriorXF* is constrained to the range 0.0 through 1.0. This resource has an effect only when *vcGlyphStyle* is set to `FillArrow`.

Default: 0.33

vcFillArrowHeadYF

This resource specifies the perpendicular distance from one side of a filled vector arrow drawn at the reference length to one of the back tips of the arrowhead. The value represents a fraction of the value of *vcRefLengthF* and, when added to half the arrow width, determines the Y component of the arrowhead size. (See figure.) If both *vcFillArrowMinFracYF* and *vcFillArrowMinFracWidthF* have the value 0.0, then the ratio of the Y component of the arrowhead size to the arrow length will be invariant for all vectors in the plot. The value of *vcFillArrowHeadYF* is constrained to the range 0.0 through 1.0. This resource has an effect only when *vcGlyphStyle* is set to `FillArrow`.

Default: 0.12

vcFillArrowHeadMinFracYF

This resource, when added to the minimum width value, specifies the Y component length of the arrowhead size for a filled arrow drawn at the minimum length, as a fraction of the component length specified by *vcFillArrowHeadYF*. (See figure.) If given the value 1.0, the arrowhead Y component will extend the same distance perpendicularly from the edge of all arrows in the plot, regardless of their length and width. This can be a useful resource to adjust to ensure that the points of even very short vector arrows remain visible. The value of *vcFillArrowHeadMinFracYF* is constrained to the range 0.0 through 1.0. This resource has an effect only when *vcGlyphStyle* is set to `FillArrow`.

Default: 0.25

vcMonoWindBarbColor

When this resource is set True, wind barb glyphs are colored using a single HLU color index, as specified by the value of the scalar resource *vcWindBarbColor*. When False, the elements of the array resource *vcLevelColors* are used to set the color of each wind barb glyph individually based on the scalar quantity associated with the vector. This resource has an effect only when *vcGlyphStyle* is set to `WindBarb`.

Default: True

vcWindBarbColor

When *vcMonoWindBarbColor* is set True, this resource sets a uniform color index for all polygon and polyline elements used to draw wind barb glyphs. If set to `Transparent` (-1), the line color index will be constrained to the value `Foreground` (1). This resource has an effect only when *vcGlyphStyle* is set to `WindBarb`.

Default: `Foreground`

vcWindBarbLineThicknessF

This resource sets the thickness of the polyline elements used to draw wind barb glyphs. The value acts as a multiplier of a (device-dependent) unit thickness. This resource has an effect only when *vcGlyphStyle* is set to `WindBarb`.

Default: 1.0

vcWindBarbTickAngleF

This resource sets the angle of the wind barb ticks in degrees as measured clockwise from the vector direction. It also sets the angle between the hypotenuse of the triangle defining the pennant polygon and the vector direction. You can render southern hemisphere wind barbs, which by convention have their ticks and pennants on the other side of the shaft, by setting *vcWindBarbTickAngleF* to a negative value. This resource has an effect only when *vcGlyphStyle* is set to `WindBarb`.

Default: 62.0

vcWindBarbTickLengthF

This resource sets the length of the wind barb ticks as a fraction of the overall length of a wind barb. The wind barb length is defined as the length of the wind barb shaft plus the projection of a full wind barb tick along the axis of the shaft. Therefore, increasing this value, for a given vector reference length, has the effect of reducing the length of the shaft itself somewhat. You may need to increase *vcRefLengthF* to compensate. This resource also sets the hypotenuse length of the triangle defining the pennant polygon. This resource has an effect only when *vcGlyphStyle* is set to `WindBarb`.

Default: 0.3

vcWindBarbTickSpacingF

This resource sets the distance between adjacent wind barbs ticks along the wind barb shaft as a fraction of the overall wind barb length. Half this distance is used as the spacing between adjacent wind barb pennants. Note that **VectorPlot** does nothing to prevent ticks or pennants from continuing off the end of the shaft if high magnitude flow fields are rendered. This resource has an effect only when *vcGlyphStyle* is set to `WindBarb`.

Default: 0.125

vcWindBarbCalmCircleSizeF

This resource sets the diameter of the circle used to represent small vector magnitudes as a fraction of the overall wind barb length. This resource has an effect only when *vcGlyphStyle* is set to `WindBarb`.

Default: 0.25

vcWindBarbScaleFactorF

This resource specifies a factor by which magnitudes passed to the wind barb drawing routines are to be scaled. It can be used to convert vector data given in other units into the conventional units used with wind barbs, which is knots. Note that this scale factor resource is entirely independent of the *vcMagnitudeScaleFactorF* and *vcMagnitudeScaleValueF* resources which influence numerical strings in annotations associated with **VectorPlot**. This resource has an effect only when *vcGlyphStyle* is set to `WindBarb`.

Default: 1.0

vcUseRefAnnoRes

If this boolean resource is True, the values for many resources belonging to the reference vector annotation are copied to the minimum vector annotation and the zero field annotation, allowing you to ensure consistency in the appearance of each of these annotations without the need to set each resource individually. The following text attribute resource values are copied to the corresponding resources of both the minimum vector annotation and the zero field annotation:

- *vcRefAnnoFontHeightF*
- *vcRefAnnoTextDirection*
- *vcRefAnnoFont*
- *vcRefAnnoFontColor*

- *vcRefAnnoFontAspectF*
- *vcRefAnnoFontThicknessF*
- *vcRefAnnoFontQuality*
- *vcRefAnnoConstantSpacingF*
- *vcRefAnnoFuncCode*
- *vcRefAnnoBackgroundColor*
- *vcRefAnnoPerimOn*
- *vcRefAnnoPerimSpaceF*
- *vcRefAnnoPerimColor*
- *vcRefAnnoPerimThicknessF*

The following resources are copied only to the corresponding minimum vector annotation resource:

- *vcRefAnnoArrowSpaceF*
- *vcRefAnnoArrowMinOffsetF*
- *vcRefAnnoArrowLineColor*
- *vcRefAnnoArrowFillColor*
- *vcRefAnnoArrowEdgeColor*
- *vcRefAnnoArrowUseVecColor*

Note that the zonal location resources, angle, the string contents, the "on" switches, and the explicit magnitude resources are not copied.

Default: False

vcRefAnnoOn

If this boolean resource is set False, **VectorPlot** will not draw the reference vector annotation.

Default: True

vcRefAnnoOrientation

NOT YET IMPLEMENTED

This resource of type NhlTOrientation specifies whether the elements of the reference vector annotation (two strings with an arrow representing a vector magnitude in between) are stacked vertically or placed side by side horizontally.

Default: `Vertical`

vcRefAnnoExplicitMagnitudeF

By default, the arrow in the reference vector annotation is drawn at the size of the current **VectorPlot** reference magnitude. However, if you set this resource to a value greater than its default value of 0.0, the size of the arrow will reflect the vector magnitude represented by this value.

Default: 0.0

vcRefAnnoArrowLineColor

If *vcRefAnnoArrowUseVecColor* is set False, this resource specifies the HLU line color index used to render the arrow in the reference annotation. If *vcRefAnnoArrowUseVecColor* is set True, and either *vcUseScalarArray* or *vcMonoLineArrowColor* is True, the line color index is determined from the value of *vcLineArrowColor*. If *vcRefAnnoArrowUseVecColor* is set True otherwise, the line color index is determined based on the magnitude of the reference annotation vector and the *vcLevelColors* array resource.

Default: `Foreground`

vcRefAnnoArrowFillColor

If *vcRefAnnoArrowUseVecColor* is set False, this resource specifies the HLU fill color index used to render the arrow in the reference annotation. If *vcRefAnnoArrowUseVecColor* is set True, and either *vcUseScalarArray* or *vcMonoFillArrowFillColor* is True, the fill color index is determined from the value of *vcFillArrowFillColor*. If *vcRefAnnoArrowUseVecColor* is set True otherwise, the fill color index is determined based on the magnitude of the reference annotation vector and the *vcLevelColors* array resource. This resource has an effect only when *vcGlyphStyle* is set to `FillArrow`.

Default: `Foreground`

vcRefAnnoArrowEdgeColor

If *vcRefAnnoArrowUseVecColor* is set False, this resource specifies the HLU line color index used to render the arrow in the reference annotation. If *vcRefAnnoArrowUseVecColor* is set True, and either *vcUseScalarArray* or *vcMonoLineArrowColor* is True, the line color index is determined from the value of *vcLineArrowColor*. If *vcRefAnnoArrowUseVecColor* is set True otherwise, the line color index is determined based on the magnitude of the reference annotation vector and the *vcLevelColors* array resource.

Default: `Foreground`

vcRefAnnoArrowUseVecColor

If this resource is set True, **VectorPlot** will base the coloring of the reference arrow on the colors used to draw the vector plot arrows as follows: if coloring by magnitude is in effect (*vcUseScalarArray* and *vcMonoVectorColor* are both set False), then the color will be that assigned to vectors representing the magnitude of the reference annotation vector; otherwise, the color will be determined from the current values assigned to *vcLineArrowColor* and/or *vcFillArrowFillColor*. If *vcRefAnnoArrowUseVecColor* is False, then *vcRefAnnoArrowColor* controls the color of the reference annotation arrow.

Default: True

vcRefAnnoArrowAngleF

This resource specifies the angle, in degrees, of the arrow used in the reference vector annotation.

Default: 0.0

vcRefAnnoArrowSpaceF

This resource specifies the minimum amount of space reserved for the reference vector annotation arrow in units of the reference annotation font height. More space may be used if the arrow extent (in units of the font height) plus twice the value of *vcRefAnnoArrowMinOffsetF* exceeds this value.

Default: 2.0

vcRefAnnoArrowMinOffsetF

This resource specifies the minimum amount of space between the reference vector annotation arrow and the text on either side in units of the reference annotation font height.

Default: 0.25

vcRefAnnoString1On

If this boolean resource is set False, **VectorPlot** will not display a string above or to the left (depending on the current value of *vcRefAnnoOrientation*) of the reference annotation arrow. Otherwise, the string will be displayed.

Default: True

vcRefAnnoString1

Specifies the string to use above or to the left (depending on the current value of *vcRefAnnoOrientation*) of the arrow in the reference vector annotation. The string may contain function codes and/or substitution substrings.

vcMagnitudeFormat determines the format of all numbers related to the vector field, and all numbers, except for the number generated from the substitution substring MSF, are scaled by the value of *vcMagnitudeScaleFactorF*. *vcScalarValueFormat* determines the format of all numbers related to the scalar field, and all numbers, except for the number generated from the substitution substring SSF, are scaled by the value of *vcScalarValueScaleFactorF*.

Default: "VMG"

vcRefAnnoString2On

If this boolean resource is set False, **VectorPlot** will not display a string below or to the right (depending on the current value of *vcRefAnnoOrientation*) of the reference annotation arrow. Otherwise, the string will be displayed.

Default: True

vcRefAnnoString2

Specifies the string to use above or to the left (depending on the current value of *vcRefAnnoOrientation*) of the arrow in the reference vector annotation. The string may contain function codes and/or substitution substrings.

vcMagnitudeFormat determines the format of all numbers related to the vector field, and all numbers, except for the number generated from the substitution substring MSF, are scaled by the value of *vcMagnitudeScaleFactorF*. *vcScalarValueFormat* determines the format of all numbers related to the scalar field, and all numbers, except for the number generated from the substitution substring SSF, are scaled by the value of *vcScalarValueScaleFactorF*.

Default: "Reference Vector"

vcRefAnnoFontHeightF

This resource controls the height, in NDC units, of characters used in the text of the reference vector annotation. The character width scales proportionally, unless you also modify the aspect ratio using the *vcRefAnnoFontAspectF* resource. The reference annotation font height scales with changes to the viewport width, unless you explicitly set *vcRefAnnoFontHeightF* during the same call.

Default: <dynamic>

vcRefAnnoTextDirection

This resource of type NhlTTextDirection specifies the direction of the text in the reference vector annotation. There are two choices:

`Down`

Each character is placed below the previous character in the text string.

`Across`

Each character is placed to the right of the previous character in the text string.

Default: `Across`

vcRefAnnoFont

This resource of type NhlTFont specifies the font used to render the vector reference annotation.

Default: "pwritx"

vcRefAnnoFontColor

This resource specifies the HLU color index used to render vector reference annotation text.

Default: `Foreground`

vcRefAnnoFontAspectF

This resource determines the shape of the reference vector annotation characters. Values increasing from 1.0 result in characters that appear thinner. Values decreasing from 1.0 make the characters appear wider. Values less than or equal to 0.0 result in a WARNING message and a restoration of the default value.

Default: 1.3125

vcRefAnnoFontThicknessF

Sets the thickness of the line used to draw vector reference annotation text. The value acts as a multiplier of a (device-dependent) unit thickness. This resource is ignored when the *vcRefAnnoFont* specifies a filled font (font indexes 21-22, 25-26, 29-30, and 33-37).

Default: 1.0

vcRefAnnoFontQuality

This resource of type NhlTFontQuality determines the quality of the font used to draw the **VectorPlot** vector reference annotation.

Default: `High`

vcRefAnnoConstantSpacingF

Normally when *vcRefAnnoFontQuality* is set to `High`, **VectorPlot** draws the reference vector annotation text using proportional spacing. Setting the *vcRefAnnoConstantSpacingF* to a value greater than 0.0 overrides this behavior. Instead, the distance from the start of one character to the next is computed by multiplying a single standard character width by the value of *vcRefAnnoConstantSpacingF*. When *vcRefAnnoConstantSpacingF* has a value between 0.0 and 1.0, characters will overlap. A value of 1.0 implies, on average, no space between characters, while values increasing from 1.0 cause the space between characters to grow. This parameter is ignored when *vcRefAnnoFontQuality* is not `High`. Values less than 0.0 result in an error and are replaced with the default value.

Default: 0.0

vcRefAnnoAngleF

NOT YET IMPLEMENTED.

Default: 0.0

vcRefAnnoFuncCode

This resource of type NhlTCharacter sets the function code character for the reference annotation strings.

Default: **:**

vcRefAnnoBackgroundColor

This resource sets the background color used to fill the box surrounding the vector reference annotation. If you do not want the box to be filled at all, set *vcRefAnnoBackgroundColor* to `Transparent` (-1).

Default: `Background`

vcRefAnnoPerimOn

vcRefAnnoPerimOn is a boolean resource that determines whether **VectorPlot** will draw an outline around the perimeter of the box surrounding the reference vector annotation. If set False, no outline will be drawn.

Default: True

vcRefAnnoPerimSpaceF

vcRefAnnoPerimSpaceF determines the spacing or margin between the text of the reference vector annotation and its edge as a fraction of current reference annotation font height.

Default: 0.33

vcRefAnnoPerimColor

This resource sets the HLU color index used to draw the perimeter of the vector reference annotation.

Default: `Foreground`

vcRefAnnoPerimThicknessF

This resource determines the thickness of the perimeter line around the vector reference annotation. The value acts as a multiplier of a (device-dependent) unit thickness.

Default: 1.0

vcRefAnnoZone

VectorPlot implements the vector reference annotation as an embedded annotation. *vcRefAnnoZone* specifies the **PlotManager** zone assigned to the vector reference annotation. The PlotManager Location Control Model requires this resource to allow control of the reference annotation consistent with other annotations. If *vcRefAnnoZone* is set to 0, the positional origin is the center of the plot viewport; otherwise it is on or outside one of the sides of the viewport.

Default: 3

vcRefAnnoSide

This resource of type NhlTPosition determines where to place the vector reference annotation in relation to the sides of the plot object. The PlotManager Location Control Model requires this resource to allow control of the vector reference annotation consistent with other annotations. Its value determines a coordinate system origin and the direction of the positional resources *vcRefAnnoParallelPosF* and *vcRefAnnoOrthogonalPosF*. It also constrains the value of *vcRefAnnoJust* appropriately. There are four settings, which behave as follows, unless *vcRefAnnoZone* is set to one of the special zones (0 or 1):

`Top`

The **PlotManager** locates the vector reference annotation relative to a line paralleling the top viewport boundary. *vcRefAnnoOrthogonalPosF* increases in the direction of increasing

NDC Y-Axis values and is constrained to positive values. *vcRefAnnoParallelPosF* increases in the direction of increasing NDC X-Axis values. *vcRefAnnoJust* is constrained to `BottomRight`, `BottomCenter`, or `BottomLeft`.

`Bottom`

The **PlotManager** locates the vector reference annotation relative to a line paralleling the bottom viewport boundary. *vcRefAnnoOrthogonalPosF* increases in the direction of decreasing NDC Y-Axis values and is constrained to positive values. *vcRefAnnoParallelPosF* increases in the direction of increasing NDC X-Axis values. *vcRefAnnoJust* is constrained to `TopRight`, `TopCenter`, or `TopLeft`.

`Right`

The **PlotManager** locates the vector reference annotation relative to a line paralleling the right viewport boundary. *vcRefAnnoOrthogonalPosF* increases in the direction of increasing NDC X-Axis values and is constrained to positive values. *vcRefAnnoParallelPosF* increases in the direction of increasing NDC Y-Axis values. *vcRefAnnoJust* is constrained to `TopLeft`, `CenterLeft`, or `BottomLeft`.

`Left`

The **PlotManager** locates the vector reference annotation relative to a line paralleling the left viewport boundary. *vcRefAnnoOrthogonalPosF* increases in the direction of decreasing NDC X-Axis values and is constrained to positive values. *vcRefAnnoParallelPosF* increases in the direction of increasing NDC Y-Axis values. *vcRefAnnoJust* is constrained to `TopRight`, `CenterRight`, or `BottomRight`.

If **vcRefAnnoZone** is set to 0, the **PlotManager** locates the vector reference annotation relative to the viewport center. If **vcRefAnnoZone** is 1, the direction of the *vcRefAnnoOrthogonalPosF* is opposite to the specification given above. Also if the **vcRefAnnoZone** is either 0 or 1, *vcRefAnnoJust* is not constrained, and *vcRefAnnoOrthogonalPosF* may take on negative values.

Default: `Bottom`

vcRefAnnoJust

This resource of type NhlTJustification, after constraint to an appropriate value based on *vcRefAnnoSide*, sets the justification of the vector reference annotation with respect to its base location. The PlotManager Location Control Model requires this resource to allow control of the vector reference annotation consistent with other annotations.

Default: `TopRight`

vcRefAnnoParallelPosF

vcRefAnnoParallelPosF specifies the coordinate of the base location of the vector reference annotation parallel to the current *vcRefAnnoSide* and directed toward increasing NDC values. The PlotManager Location Control Model requires this resource to allow control of the vector reference annotation consistent with other annotations.

Default: 1.0

vcRefAnnoOrthogonalPosF

vcRefAnnoOrthogonalPosF sets the coordinate of the base location of the vector reference annotation orthogonal to the current *vcRefAnnoSide* and directed away from the center of the viewport. The PlotManager Location Control Model requires this resource to allow control of the vector reference annotation consistent with other annotations.

Default: 0.02

vcMinAnnoOn

If this boolean resource is set False, **VectorPlot** will not draw the minimum vector annotation.

Default: True

vcMinAnnoOrientation

NOT YET IMPLEMENTED

This resource of type NhlTOrientation specifies whether the elements of the minimum vector annotation (two strings with an arrow representing a vector magnitude in between) are stacked vertically or placed side by side horizontally.

Default: `Vertical`

vcMinAnnoExplicitMagnitudeF

By default, the arrow in the minimum vector annotation is drawn at the size of the current **VectorPlot** minimum magnitude. However, if you set this resource to a value greater than its default value of 0.0, the size of arrow will reflect the vector magnitude represented by this value.

Default: 0.0

vcMinAnnoArrowLineColor

If *vcMinAnnoArrowUseVecColor* is set False, this resource specifies the HLU line color index used to render the arrow in the reference annotation. If *vcMinAnnoArrowUseVecColor* is set True, and either *vcUseScalarArray* or *vcMonoLineArrowColor* is True, the line color index is determined from the value of *vcLineArrowColor*. If *vcMinAnnoArrowUseVecColor* is set True otherwise, the line color index is determined based on the magnitude of the reference annotation vector and the *vcLevelColors* array resource.

Default: `Foreground`

vcMinAnnoArrowFillColor

If *vcMinAnnoArrowUseVecColor* is set False, this resource specifies the HLU fill color index used to render the arrow in the reference annotation. If *vcMinAnnoArrowUseVecColor* is set True, and either *vcUseScalarArray* or *vcMonoFillArrowFillColor* is True, the fill color index is determined from the value of *vcFillArrowFillColor*. If *vcMinAnnoArrowUseVecColor* is set True otherwise, the fill color index is determined based on the magnitude of the reference annotation vector and the *vcLevelColors* array resource. This resource has an effect only when *vcGlyphStyle* is set to `FillArrow`.

Default: `Foreground`

vcMinAnnoArrowEdgeColor

If *vcMinAnnoArrowUseVecColor* is set False, this resource specifies the HLU line color index used to render the arrow in the reference annotation. If *vcMinAnnoArrowUseVecColor* is set True, and either *vcUseScalarArray* or *vcMonoLineArrowColor* is True, the line color index is determined from the value of *vcLineArrowColor*. If *vcMinAnnoArrowUseVecColor* is set True otherwise, the line color index is determined based on the magnitude of the reference annotation vector and the *vcLevelColors* array resource.

Default: `Foreground`

vcMinAnnoArrowUseVecColor

If this resource is set True, **VectorPlot** will base the coloring of the minimum arrow on the colors used to draw the vector plot arrows as follows: if coloring by magnitude is in effect (*vcUseScalarArray* and *vcMonoVectorColor* are both set False), then the color will be that assigned to vectors representing the magnitude of the minimum annotation vector; otherwise, the color index will be determined from the current values assigned to *vcLineArrowColor* and/or *vcFillArrowFillColor*. If *vcMinAnnoArrowUseVecColor* is False, then *vcMinAnnoArrowColor* controls the color of the minimum annotation arrow.

Default: True

vcMinAnnoArrowAngleF

This resource specifies the angle, in degrees, of the arrow used in the minimum vector annotation.

Default: 0.0

vcMinAnnoArrowSpaceF

This resource specifies the minimum amount of space reserved for the reference vector annotation arrow in units of the minimum annotation font height. More space may be used if the arrow extent (in units of the font height) plus twice the value of *vcMinAnnoArrowMinOffsetF* exceeds this value.

Default: 2.0

vcMinAnnoArrowMinOffsetF

This resource specifies the minimum amount of space between the minimum vector annotation arrow and the text on either side in units of the minimum annotation font height.

Default: 0.25

vcMinAnnoString1On

If this boolean resource is set False, **VectorPlot** will not display a string above or to the left (depending on the current value of *vcMinAnnoOrientation*) of the minimum annotation arrow. Otherwise, the string will be displayed.

Default: True

vcMinAnnoString1

Specifies the string to use above or to the left (depending on the current value of *vcMinAnnoOrientation*) of the arrow in the minimum vector annotation. The string may contain function codes and/or substitution substrings.

vcMagnitudeFormat determines the format of all numbers related to the vector field, and all numbers, except for the number generated from the substitution substring MSF, are scaled by the value of *vcMagnitudeScaleFactorF*. *vcScalarValueFormat* determines the format of all numbers related to the scalar field, and all numbers, except for the number generated from the substitution substring SSF, are scaled by the value of *vcScalarValueScaleFactorF*.

Default: "VMG"

vcMinAnnoString2On

If this boolean resource is set False, **VectorPlot** will not display a string below or to the right (depending on the current value of *vcMinAnnoOrientation*) of the minimum annotation arrow. Otherwise, the string will be displayed.

Default: True

vcMinAnnoString2

Specifies the string to use above or to the left (depending on the current value of *vcMinAnnoOrientation*) of the arrow in the minimum vector annotation. The string may contain function codes and/or substitution substrings.

vcMagnitudeFormat determines the format of all numbers related to the vector field, and all numbers, except for the number generated from the substitution substring MSF, are scaled by the value of *vcMagnitudeScaleFactorF*. *vcScalarValueFormat* determines the format of all numbers related to the scalar field, and all numbers, except for the number generated from the substitution substring SSF, are scaled by the value of *vcScalarValueScaleFactorF*.

Default: "Minimum Vector"

vcMinAnnoFontHeightF

This resource controls the height, in NDC units, of characters used in the text of the minimum vector annotation. The character width scales proportionally, unless you also modify the aspect ratio using the *vcMinAnnoFontAspectF* resource. The minimum annotation font height scales with changes to the viewport width, unless you explicitly set *vcMinAnnolFontHeightF* during the same call.

Default: <dynamic>

vcMinAnnoTextDirection

This resource of type NhlTTextDirection specifies the direction of the text in the minimum vector annotation. There are two choices:

`Down`
: Each character is placed below the previous character in the text string.

`Across`
: Each character is placed to the right of the previous character in the text string.

Default: `Across`

vcMinAnnoFont

This resource of type NhlTFont specifies the font used to render the vector minimum annotation.

Default: "pwritx"

vcMinAnnoFontColor

This resource specifies the HLU color index used to render vector minimum annotation text.

Default: `Foreground`

vcMinAnnoFontAspectF

This resource determines the shape of the minimum vector annotation characters. Values increasing from 1.0 result in characters that appear thinner. Values decreasing from 1.0 make the characters appear wider. Values less than or equal to 0.0 result in a WARNING message and a restoration of the default value.

Default: 1.3125

vcMinAnnoFontThicknessF

Sets the thickness of the line used to draw vector minimum annotation text. The value acts as a multiplier of a (device-dependent) unit thickness. This resource is ignored when the *vcMinAnnoFont* specifies a filled font (font indexes 21-22, 25-26, 29-30, and 33-37).

Default: 1.0

vcMinAnnoFontQuality

This resource of type NhlTFontQuality determines the quality of the font used to draw the **VectorPlot** vector minimum annotation.

Default: `High`

vcMinAnnoConstantSpacingF

Normally when *vcMinAnnoFontQuality* is set to `High`, **VectorPlot** draws the minimum vector annotation text using proportional spacing. Setting the *vcMinAnnoConstantSpacingF* to a value greater than 0.0 overrides this behavior. Instead, the distance from the start of one character to the next is computed by multiplying a single standard character width by the value of *vcMinAnnoConstantSpacingF*. When *vcMinAnnoConstantSpacingF* has a value between 0.0 and 1.0, characters will overlap. A value of 1.0 implies, on average, no space between characters, while values increasing from 1.0 cause the space between characters to grow. This parameter is ignored when *vcMinAnnoFontQuality* is not `High`. Values less than 0.0 result in an error and are replaced with the default value.

Default: 0.0

vcMinAnnoAngleF

NOT YET IMPLEMENTED.

Default: 0.0

vcMinAnnoFuncCode

This resource of type NhlTCharacter sets the function code character for the minimum vector annotation strings.

Default: **:**

vcMinAnnoBackgroundColor

This resource sets the background color used to fill the box surrounding the vector minimum annotation. If you do not want the box to be filled at all, set *vcMinAnnoBackgroundColor* to `Transparent` (-1).

Default: `Background`

vcMinAnnoPerimOn

vcMinAnnoPerimOn is a boolean resource that determines whether **VectorPlot** will draw an outline around the perimeter of the box surrounding the minimum vector annotation. If set False, no outline will be drawn.

Default: True

vcMinAnnoPerimSpaceF

vcMinAnnoPerimSpaceF determines the spacing, or margin, between the text of the minimum vector annotation and its edge as a fraction of current minimum annotation font height.

Default: 0.33

vcMinAnnoPerimColor

This resource sets the HLU color index used to draw the perimeter of the vector minimum annotation.

Default: `Foreground`

vcMinAnnoPerimThicknessF

This resource determines the thickness of the perimeter line around the vector minimum annotation. The value acts as a multiplier of a (device-dependent) unit thickness.

Default: 1.0

vcMinAnnoZone

VectorPlot implements the vector minimum annotation as an embedded annotation. *vcMinAnnoZone* specifies the **PlotManager** zone assigned to the vector minimum annotation. The PlotManager Location Control Model requires this resource to allow control of the minimum annotation consistent with other annotations. If *vcMinAnnoZone* is set to 0, the positional origin is the center of the plot viewport; otherwise it is on or outside one of the sides of the viewport.

Default: 3

vcMinAnnoSide

This resource of type NhlTPosition determines where to place the vector minimum annotation in relation to the sides of the plot object. The PlotManager Location Control Model requires this resource to allow control of the vector minimum annotation consistent with other annotations. Its value determines a coordinate system origin and the direction of the positional resources *vcMinAnnoParallelPosF* and *vcMinAnnoOrthogonalPosF*. It also constrains the value of *vcMinAnnoJust* appropriately. There are four settings, which behave as follows, unless *vcMinAnnoZone* is set to one of the special zones (0 or 1):

`Top`

The **PlotManager** locates the vector minimum annotation relative to a line paralleling the top viewport boundary. *vcMinAnnoOrthogonalPosF* increases in the direction of increasing NDC Y-Axis values and is constrained to positive values. *vcMinAnnoParallelPosF* increases in the direction of increasing NDC X-Axis values. *vcMinAnnoJust* is constrained to `BottomRight`, `BottomCenter`, or `BottomLeft`.

`Bottom`

The **PlotManager** locates the vector minimum annotation relative to a line paralleling the bottom viewport boundary. *vcMinAnnoOrthogonalPosF* increases in the direction of decreasing NDC Y-Axis values and is constrained to positive values. *vcMinAnnoParallelPosF* increases in the direction of increasing NDC X-Axis values. *vcMinAnnoJust* is constrained to `TopRight`, `TopCenter`, or `TopLeft`.

`Right`

The **PlotManager** locates the vector minimum annotation relative to a line paralleling the right viewport boundary. *vcMinAnnoOrthogonalPosF* increases in the direction of increasing NDC X-Axis values and is constrained to positive values. *vcMinAnnoParallelPosF* increases in the direction of increasing NDC Y-Axis values. *vcMinAnnoJust* is constrained to `TopLeft`, `CenterLeft`, or `BottomLeft`.

`Left`

The **PlotManager** locates the vector minimum annotation relative to a line paralleling the left viewport boundary. *vcMinAnnoOrthogonalPosF* increases in the direction of decreasing NDC X-Axis values and is constrained to positive values. *vcMinAnnoParallelPosF* increases in the direction of increasing NDC Y-Axis values. *vcMinAnnoJust* is constrained to `TopRight`, `CenterRight`, or `BottomRight`.

If **vcMinAnnoZone** is set to 0, The **PlotManager** locates the vector minimum annotation relative to the viewport center. If **vcMinAnnoZone** is 1, the direction of the *vcMinAnnoOrthogonalPosF* is opposite to the specification given above. Also if the **vcMinAnnoZone** is either 0 or 1, *vcMinAnnoJust* is not constrained, and *vcMinAnnoOrthogonalPosF* may take on negative values.

Default: `Bottom`

vcMinAnnoJust

This resource of type NhlTJustification, after constraint to an appropriate value based on *vcMinAnnoSide*, sets the justification of the vector minimum annotation with respect to its base location. The PlotManager Location Control Model requires this resource to allow control of the vector minimum annotation consistent with other annotations.

Default: `TopRight`

vcMinAnnoParallelPosF

vcMinAnnoParallelPosF specifies the coordinate of the base location of the vector minimum annotation parallel to the current *vcMinAnnoSide* and directed toward increasing NDC values. The PlotManager Location Control Model requires this resource to allow control of the vector minimum annotation consistent with other annotations.

Default: 1.0

vcMinAnnoOrthogonalPosF

vcMinAnnoOrthogonalPosF sets the coordinate of the base location of the vector minimum annotation orthogonal to the current *vcMinAnnoSide* and directed away from the center of the viewport. The PlotManager Location Control Model requires this resource to allow control of the vector minimum annotation consistent with other annotations.

Default: 0.02

vcNoDataLabelOn

This boolean resource, when set True, causes a label to appear when a **vectorplot** is drawn without any data having been provided. Except for the label string, all attributes of this label, including its position, are set using resources belonging to the constant field label. When set False, no such label appears.

Default: True

vcNoDataLabelString

This resource contains the string that appears in the 'No Data' label if you draw a **VectorPlot** object without providing any data. No substitution substrings are allowed in this label, since all the substitutions depend on data being available. Except for the boolean switch that turns it on and off, all attributes of this label, including its position, are set using resources belonging to the constant field label.

Default: "NO VECTOR DATA"

vcZeroFLabelOn

The **VectorPlot** object draws a zero field label annotation only when *vcZeroFLabelOn* is set True and the **ScalarField** data are determined to consist entirely of zero magnitude vectors within the limits of the available precision.

Default: True

vcZeroFLabelString

Specifies the string to use when drawing a zero field label. The string may contain function codes.

Default: "ZERO FIELD"

vcZeroFLabelTextDirection

This resource of type NhlTTextDirection specifies the direction of the text in the zero field label. There are two choices:

`Down`

Each character is placed below the previous character in the text string.

`Across`

Each character is placed to the right of the previous character in the text string.

These descriptions apply before rotation due to *vcZeroFieldAngleF*.

Default: `Across`

vcZeroFLabelFontHeightF

This resource controls the height, in NDC units, of characters used in the text of the zero field label. The character width scales proportionally, unless you also modify the aspect ratio using the *vcZeroFLabelFontAspectF* resource. The zero label text height scales with changes to the viewport width, unless you explicitly set *vcZeroFLabelFontHeightF* during the same call.

Default: <dynamic>

vcZeroFLabelFont

This resource of type NhlTFont specifies the font used to render the zero field label.

Default: 0

vcZeroFLabelFontColor

This resource specifies the HLU color index used to render zero field label text.

Default: True

vcZeroFLabelFontAspectF

This resource determines the shape of the zero field label characters. Values increasing from 1.0 result in characters that appear thinner. Values decreasing from 1.0 make the characters appear wider. Values less than or equal to 0.0 result in a WARNING message and a restoration of the default value.

Default: 1.3125

vcZeroFLabelFontThicknessF

Sets the thickness of the line used to draw zero field label text. The value acts as a multiplier of a (device-dependent) unit thickness. This resource is ignored when the *vcZeroFLabelFont* specifies a filled font (font indexes 21-22, 25-26, 29-30, and 33-37).

Default: 1.0

vcZeroFLabelFontQuality

This resource of type NhlTFontQuality determines the quality of the font used to draw the **VectorPlot** zero field label.

Default: `High`

vcZeroFLabelConstantSpacingF

Normally when *vcZeroFFontQuality* is set to `High`, the **VectorPlot** object writes zero field label text with proportional spacing. Setting the *vcZeroFLabelConstantSpacingF* to a value greater than

0.0 overrides this behavior. Instead, the distance from the start of one character to the next is computed by multiplying a single standard character width by the value of *vcZeroFConstantSpacingF*. When *vcZeroFConstantSpacingF* has a value between 0.0 and 1.0, characters will overlap. A value of 1.0 implies, on average, no space between characters, while values increasing from 1.0 cause the space between characters to grow. This parameter is ignored when *vcZeroFLabelFontQuality* is not `High`. Values less than 0.0 result in an error and are replaced with the default value.

Default: 0.0

vcZeroFLabelAngleF

This resource specifies the angle, in degrees, of the zero field label text and its surrounding box.

Default: 0.0

vcZeroFLabelFuncCode

This resource of type NhlTCharacter sets the function code character that the low level utilities will use when parsing the zero field label string.

Default: **:**

vcZeroFLabelBackgroundColor

This resource sets the background color used to fill the box surrounding the zero field label. If you do not want the box to be filled at all, set *vcZeroFLabelBackgroundColor* to `Transparent` (-1).

Default: `Background`

vcZeroFLabelPerimOn

vcZeroFLabelPerimOn is a boolean resource that determines whether **VectorPlot** will draw an outline around the perimeter of the box surrounding vector zero field label. If set False, no outline will be drawn.

Default: True

vcZeroFLabelPerimSpaceF

vcZeroFLabelPerimSpaceF determines the spacing, or margin, between the text of the zero field label and the edge of the zero field label box as a fraction of the current label text height.

Default: 0.33

vcZeroFLabelPerimColor

This resource sets the HLU color index used to draw the perimeter of the zero field label box.

Default: `Foreground`

vcZeroFLabelPerimThicknessF

This resource determines the thickness of the perimeter line around the zero field label box. The value acts as a multiplier of a (device-dependent) unit thickness.

Default: 1.0

vcZeroFLabelZone

VectorPlot implements the zero field label as an embedded annotation. *vcZeroFLabelZone* specifies the **PlotManager** zone assigned to the zero field annotation. The PlotManager Location Control Model requires this resource to allow control of the zero field label consistent with other annotations. If *vcZeroFLabelZone* is set to 0, the positional origin is the center of the plot viewport; otherwise it is on or outside one of the sides of the viewport.

If you create a **VectorPlot** object without an active **PlotManager**, by setting *tfPlotManagerOn* False, then **VectorPlot** manages the zero field annotation by itself. In this case, the *vcZeroFLabelZone* resource is not as meaningful.

Default: 0

vcZeroFLabelSide

This resource of type NhlTPosition determines where to place the zero field annotation in relation to the sides of the plot object. The PlotManager Location Control Model requires this resource to allow control of the zero field label consistent with other annotations. Its value determines a coordinate system origin and the direction of the positional resources *vcZeroFLabelParallelPosF* and *vcZeroFLabelOrthogonalPosF*. It also constrains the value of the *vcZeroFLabelJust* appropriately. There are four settings, which behave as follows, unless *vcZeroFLabelZone* is set to one of the special zones (0 or 1):

`Top`

The **PlotManager** locates the zero field label annotation relative to a line paralleling the top viewport boundary. *vcZeroFLabelOrthogonalPosF* increases in the direction of increasing NDC Y-Axis values and is constrained to positive values. *vcZeroFLabelParallelPosF* increases in the direction of increasing NDC X-Axis values. *vcZeroFLabelJust* is constrained to `BottomRight`, `BottomCenter`, or `BottomLeft`.

`Bottom`

The **PlotManager** locates the zero field label annotation relative to a line paralleling the bottom viewport boundary. *vcZeroFLabelOrthogonalPosF* increases in the direction of decreasing NDC Y-Axis values and is constrained to positive values. *vcZeroFLabelParallelPosF* increases in the direction of increasing NDC X-Axis values. *vcZeroFLabelJust* is constrained to `TopRight`, `TopCenter`, or `TopLeft`.

`Right`

The **PlotManager** locates the zero field label annotation relative to a line paralleling the right viewport boundary. *vcZeroFLabelOrthogonalPosF* increases in the direction of increasing NDC X-Axis values and is constrained to positive values. *vcZeroFLabelParallelPosF* increases in the direction of increasing NDC Y-Axis values. *vcZeroFLabelJust* is constrained to `TopLeft`, `CenterLeft`, or `BottomLeft`.

`Left`

The **PlotManager** locates the zero field label annotation relative to a line paralleling the left viewport boundary. *vcZeroFLabelOrthogonalPosF* increases in the direction of decreasing NDC X-Axis values and is constrained to positive values. *vcZeroFLabelParallelPosF* increases in the direction of increasing NDC Y-Axis values. *vcZeroFLabelJust* is constrained to `TopRight`, `CenterRight`, or `BottomRight`.

If **vcZeroFLabelZone** is set to 0, The **PlotManager** locates the zero field label annotation relative to the viewport center. If **vcZeroFLabelZone** is 1, the direction of the *vcZeroFLabelOrthogonalPosF* is opposite to the specification given above. Also if the **vcZeroFLabelZone** is either 0 or 1, *vcZeroFLabelJust* is not constrained, and *vcZeroFLabelOrthogonalPosF* may take on negative values.

Default: `Bottom`

vcZeroFLabelParallelPosF

vcZeroFLabelParallelPosF specifies the coordinate of the base location of the zero field label annotation parallel to the current *vcZeroFLabelSide* and directed toward increasing NDC values. The PlotManager Location Control Model requires this resource to allow control of the zero field label consistent with other annotations.

Default: 0.0

vcZeroFLabelOrthogonalPosF

vcZeroFLabelOrthogonalPosF sets the coordinate of the base location of the zero field label annotation orthogonal to the current *vcZeroFLabelSide* and directed away from the center of the viewport. The PlotManager Location Control Model requires this resource to allow control of the zero field label consistent with other annotations.

Default: 0.0

vcZeroFLabelJust

This resource of type NhlTJustification, after constraint to an appropriate value based on *vcZeroFLabelSide*, sets the justification of the zero field label annotation with respect to its base location. The PlotManager Location Control Model requires this resource to allow control of the zero field label consistent with other annotations.

Default: `CenterCenter`

vcMagnitudeScalingMode

This resource of type NhlTScalingMode determines how numbers representing magnitudes in the vector field data are scaled when they appear in the annotations provided by **VectorPlot**. There are five choices:

`ScaleFactor`

The **VectorPlot** object divides the magnitude by the value of *vcLabelScaleValueF* to obtain the number that appears in the label.

`ConfineToRange`

The **VectorPlot** object uses *vcLabelScaleValueF* as an exclusive upper bound. A scale factor is selected such that the scaled maximum magnitude will be less than *vcLabelScaleValueF*, but greater than or equal to *vcLabelScaleValueF* divided by 10.0. As an example, setting *vcLabelScaleValueF* to 100.0 would result in the number used to represent the scaled maximum magnitude falling in the range 10.0 to 99.999....

`TrimZeros`

The **VectorPlot** object selects a scale factor such that the number representing the scaled maximum magnitude will have the fewest possible extra zeros. If the maximum magnitude were 245000, for instance, it would scale to 245. If it were 0.000245, it would scale to 0.245. (The zero conventionally placed before the decimal point of a number less than 1.0 is not considered.)

`MaxSigDigitsLeft`

The **VectorPlot** object selects a scale factor such that the number representing the scaled maximum magnitude will have its rightmost significant digit directly to the left of the decimal point. The number of significant digits is determined by the format string resource, *vcMaxDataValueFormat*.

`AllIntegers`

The **VectorPlot** object selects a scale factor such that the numbers representing the vector levels (as determined by examination of the values of the *vcLevels* array) are reduced to integers.

In each case, the scale factor selected is placed into the read-only resource, *vcMagnitudeScaleFactorF*.

Default: `ScaleFactor`

vcMagnitudeScaleValueF

The interpretation of *vcMagnitudeScaleValueF* depends on the value of the enumerated resource *vcMagnitudeScalingMode*. When *vcMagnitudeScalingMode* is set to `ScaleFactor`, values are divided by the value of *vcMagnitudeScaleValueF* to obtain the numbers appearing in **VectorPlot** object labels. If *vcMagnitudeScalingMode* is set to `ConfineToRange`, the *vcMagnitudeScaleValueF* represents an exclusive upper bound for the numbers used to represent the data values. For other values of *vcMagnitudeScalingMode*, the *vcMagnitudeScaleValueF* resource is ignored.

Default: 1.0

vcMagnitudeScaleFactorF

vcMagnitudeScaleFactorF is a read-only resource that contains the current scale factor that applies to annotations with numbers representing magnitudes in the **VectorField** data object. Multiplying the numbers in the annotations by the value of *vcMagnitudeScaleFactorF* gives the actual magnitudes the numbers represent. **VectorPlot** calculates its value based on the maximum vector magnitude, the current mode of the enumerated resource, *vcMagnitudeScalingMode*, and perhaps the value of *vcMagnitudeScaleValueF*. Its value may also be influenced by the number of significant digits specified by the format specification resource, *vcMagnitudeFormat*.

Default: 1.0

vcMagnitudeFormat

The *vcMagnitudeFormat* resource is a string that specifies the printing format for vector field values according to the HLU Floating Point Format Specification scheme. This resource controls the formatting of any number representing **VectorField** data when used in a **VectorPlot** annotation.

Default: "*+^sg"

vcScalarValueScalingMode

This resource of type NhlTScalingMode determines how to scale numbers related to the scalar field data used to color the vectors when these numbers appear in the annotations provided by **VectorPlot**. There are five choices:

`ScaleFactor`

The **VectorPlot** object divides the scalar value by the value of *vcLabelScaleValueF* to obtain the number that appears in the label.

`ConfineToRange`

The **VectorPlot** object uses *vcLabelScaleValueF* as an exclusive upper bound. A scale factor is selected such that the scaled value representing the scalar maximum absolute value will be less than *vcLabelScaleValueF*, but greater than or equal to *vcLabelScaleValueF* divided by 10.0. As an example, setting *vcLabelScaleValueF* to 100.0 would result in the number used to represent the maximum absolute value falling in the range 10.0 to 99.999....

`TrimZeros`

The **VectorPlot** object selects a scale factor such that the number representing the value with the maximum absolute value will have the fewest possible extra zeros. If the maximum absolute value were 245000, for instance, it would scale to 245. If it were 0.000245, it would scale to 0.245. (The zero conventionally placed before the decimal point of a number less than 1.0 is not considered.)

`MaxSigDigitsLeft`

The **VectorPlot** object selects a scale factor such that the number representing the value with the maximum absolute value will have its rightmost significant digit directly to the left of the decimal point. The number of significant digits is determined by the format string resource, *vcMaxDataValueFormat*.

`AllIntegers`

The **VectorPlot** object selects a scale factor such that the numbers representing vector levels (as determined by examination of the values of the *vcLevels* array) are reduced to integers.

In each case, the scale factor selected is placed into the read-only resource, *vcScalarValueScaleFactorF*.

Default: `ScaleFactor`

vcScalarValueScaleValueF

The interpretation of *vcScalarValueScaleValueF* depends on the value of the enumerated resource *vcScalarValueScalingMode*. When *vcScalarValueScalingMode* is set to `ScaleFactor`, values are divided by the value of *vcScalarValueScaleValueF* to obtain the numbers appearing in **VectorPlot** object labels. If *vcScalarValueScalingMode* is set to `ConfineToRange`, the *vcScalarValueScaleValueF* represents an exclusive upper bound for the numbers used to represent the data values. For other values of *vcScalarValueScalingMode*, the *vcScalarValueScaleValueF* resource is ignored.

Default: 1.0

vcScalarValueScaleFactorF

vcScalarValueScaleFactorF is a read-only resource that contains the current scale factor that applies to annotations with numbers representing values from the associated **ScalarField** object. Multiplying the numbers in the annotations by the value of *vcScalarValueScaleFactorF* gives the actual scalar values the numbers represent. **VectorPlot** calculates its value based on the maximum absolute value in the scalar field data, the current mode of the enumerated resource, *vcScalarValueScalingMode*, and perhaps the value of *vcScalarValueScaleValueF*. Its value may also be influenced by the number of significant digits specified by the format specification resource, *vcScalarValueFormat*.

Default: 1.0

vcScalarValueFormat

The *vcScalarValueFormat* resource is a string that specifies the printing format for vector field values according to the HLU Floating Point Format Specification scheme. This resource controls the formatting of any number representing **VectorField** data when used in a **VectorPlot** annotation.

Default: "*+^sg"

vcExplicitLabelBarLabelsOn

This boolean resource allows you control the labels that appear in the **VectorPlot LabelBar** explicitly. When set True, **VectorPlot** does not block the **LabelBar** resources *lbLabelStrings* and *lbLabelAlignment*. Therefore you can directly control both the contents of the **LabelBar's** label strings and their alignment with respect to the label boxes. When *vcExplicitLabelBarLabelsOn* is set False, **VectorPlot** sets both of these resources based on the current vector line labels and the value of the resource *vcLabelBarEndLabelsOn*. If you set this resource True but do not set the *lbLabelStrings* array resource, **VectorPlot** will set it for you one time. This allows you to get an initial set of strings as a starting point for any customization you want to perform.

Default: False

vcLabelBarEndLabelsOn

When this boolean resource is set True and *vcExplicitLabelBarLabelsOn* is False, **VectorPlot** creates labels for the two ends of the **LabelBar**. The label at one end will be a string representation of a minimum value and at the other end will be a string representation of the maximum value. These values represent either a vector magnitude or, if *vcUseScalarArray* is set True, some other scalar value. Both strings will be formatted according to the format specification in effect for the other labels provided by **VectorPlot** to the **LabelBar**.

Default: False

vcLabelsOn

If this resource is set True, **VectorPlot** will place labels representing the vector magnitude next to each arrow in the vector field plot. These labels are primarily intended as a debugging aid, since in order to avoid excessive overlap, you must typically set the label text size too small to be readable without magnification. For this reason, only a few resources are available to control the appearance of these labels. They are rendered using low quality text.

Default: False

vcLabelsUseVectorColor

If this resource is set True, **VectorPlot** draws the vector magnitude labels using the color of the corresponding vector. Otherwise the labels are drawn using the color index specified by *vcLabelFontColor*.

Default: False

vcLabelFontColor

If *vcLabelsUseVectorColor* is set False, this resource specifies the HLU color index used to draw the vector magnitude labels.

Default: `Foreground`

vcLabelFontHeightF

This resource specifies the NDC height of characters used for vector magnitude labels. The label font height scales with changes to the viewport width, unless you explicitly set *vcLabelFontHeightF* during the same call.

Default: <dynamic>

VectorField class resource descriptions

vfDataArray

This resource specifies a 3-dimensional array containing the data comprising a vector field. You must specify either *vfDataArray* or the two resources *vfUDataArray* and *vfVDataArray* in order to create a **VectorField** object. The slowest varying dimension (first dimension in C, third dimension in Fortran) must contain two elements, representing the two components of the field. The first element represents the U data and the second the V data (before the effect of *vfExchangeUVData* is taken into account). The fastest varying dimension represents the X Axis of the data coordinate system; *vfXC...* **VectorField** resources always relate to this dimension. The middle dimension represents the data coordinate Y Axis; *vfYC...* resources always relate to this dimension.

Although the **VectorField** object always converts the data into type NhlTFloat before passing them along to recipient objects, it accepts data arrays containing data of any of the following types:

- NhlTByte
- NhlTCharacter
- NhlTShort
- NhlTInteger
- NhlTLong
- NhlTFloat
- NhlTDouble

Default: NULL

vfUDataArray

If *vfDataArray* is not set, then this resource and *vfVDataArray* must both be specified in order to create a **VectorField** object. Together, these resources specify the data defining a vector field using two arrays specifying the vector components. The fastest varying dimension (second dimension in C, first dimension in Fortran) represents the X Axis of the data coordinate system; *vfXC...* **VectorField** resources always relate to this dimension. The other dimension represents the data coordinate Y Axis; *vfYC...* resources always relate to this dimension.

You may set *vfUDataArray* using any of the data types **VectorField** accepts for the *vfDataArray* resource.

Default: NULL

vfVDataArray

If *vfDataArray* is not set, then this resource and *vfUDataArray* must both be specified in order to create a **VectorField** object. Together, these resources specify the data defining a vector field using two arrays specifying the vector components. The fastest varying dimension (second dimension in C, first dimension in Fortran) represents the X Axis of the data coordinate system; *vfXC...* **VectorField** resources always relate to this dimension. The other dimension represents the data coordinate Y Axis; *vfYC...* resources always relate to this dimension.

You may set *vfVDataArray* using any of the data types **VectorField** accepts for the *vfDataArray* resource.

Default: NULL

vfXArray

The values of the *vfXArray* resource sequentially represent the X-Axis coordinate of each successive element along the fastest-varying dimension of the data. If *vfXArray* is set to a non-`NULL` value, **VectorField** checks to ensure first that it has the same number of elements as the X-axis dimension of the data and second that it is monotonically increasing or descreasing. If the input array fails either of these tests, *vfXArray* is reset to its default value of `NULL`, and the data are treated as if each array element were equally spaced along the X Axis between the values of *vfXCStartV* and *vfXCEndV*. If the input array is valid, **VectorField** replaces the current value of *vfXCStartV* with the value of the first element of *vfXArray*, and *vfXCEndV* with the value of the last element. You can specify *vfXArray* using a single-dimensioned array of any of the types allowed for *vfDataArray*.

V4.1 Status Note 1

Default: NULL

vfYArray

The values of the *vfYArray* resource sequentially represent the Y-Axis coordinate of each successive element along the middle or slowest-varying dimension of the data. If *vfYArray* is set to a non-`NULL` value, **VectorField** checks to ensure first that it has the same number of elements as the Y-axis dimension of the data and second that it is monotonically increasing or descreasing. If the input array fails either of these tests, *vfYArray* is reset to its default value of `NULL`, and the data are treated as if each array element were equally spaced along the Y Axis between the values of *vfYCStartV* and *vfYCEndV*. If the input array is valid, **VectorField** replaces the current value of *vfYCStartV* with the value of the first element of *vfYArray*, and *vfYCEndV* with the value of the last element. You can specify *vfYArray* using a single-dimensioned array of any of the types allowed for *vfDataArray*.

V4.1 Status Note 1

Default: NULL

vfPolarData

If this resource is set True, the arrays passed into **VectorField** are assumed to represent the vector field in terms of magnitude and angular direction rather than by orthogonal vector components. By default the U array is interpreted as the magnitude and the V array as direction. However, if the resource *vfExchangeUVData* is set True, the V array will be assumed to contain the magnitude and the U array the direction.

Default: False

vfCopyData

The boolean resource *vfCopyData* allows you to control whether the **VectorField** object always copies the elements of the *vfDataArray* into a private memory area. If you set *vfCopyData* False, **VectorField** will not copy the *vfXArray* or the *vfYArray* data and will try to avoid making a copy of the *vfDataArray* data. You are then responsible for ensuring that the data memory locations for these arrays are preserved intact for the lifetime of the **VectorField**. In return, you may save dynamic memory space. Note that a number of circumstances require **VectorField** to copy the data regardless. These include supplying data of any type other than NhlTFloat, requesting that the data dimensions be exchanged, or setting a stride value greater than 1 in either the X or the Y dimension. Also, if you specify a data array subset and one or both of the coordinate array resources (*sfXArray* and/or *sfYArray*) is non-`NULL`, copies are made of the subsetted part of the coordinate array.

Default: True

vfExchangeDimensions

When this boolean resource is set True, **VectorField** exchanges the X and Y dimensions of the data before passing them to the recipient object. This will have the effect of turning the plot on its side when rendered by a plot object. Note that since the U and V components implicitly exist in the coordinate system that is exchanged, these arrays must be exchanged at the same time in order maintain an equivalent relationship between the elements of the field. You can do this by inverting the value of the resource *vfExchangeUVData*.

Whenever *vfExchangeDimensions* is True, **VectorField** must copy the data, regardless of the setting of the *vfCopyData* resource.

Default: False

vfExchangeUVData

When this boolean resource is set True, **VectorField** exchanges the roles of the U and V arrays. If you supply the data using the single array resource *vfDataArray*, the two elements of the slowest varying dimension are traded. If *vfExchangeDimensions* is set True, you must invert the current value of *vfExchangeUVData* in order to preserve the current relationship of the elements of the vector field. If *vfPolarData* is set True, then this resource controls which array is used for the magnitude and which for the angular direction.

Default: False

vfSingleMissingValue

When this boolean resource is set True, **VectorField** assumes that any missing values in both the U and V arrays are represented by the same sentinel value. In this case, you need set only one of the resources *vfMissingUValueV* and *vfMissingVValueV* to indicate the sentinel missing value. If both are set, then the value given by *vfMissingVValueV* is ignored. When *vfSingleMissingValue* is set False, then the U and V array are examined separately for their individual missing values, as specified by the resources *vfMissingUValueV* and *vfMissingVValueV*.

Default: False

vfMissingUValueV

If *vfSingleMissingValue* is False and you set *vfMissingUValueV* to a non-NULL value, **VectorField** treats occurrences of its value in the U data array as representing an undefined data point. If left unset, **VectorField** assumes that all values in the U data array are valid.

If *vfSingleMissingValue* is True and you set *vfMissingUValueV* to a non-NULL value, **VectorField** treats occurrences of its value in either the U or V data arrays as representing an undefined data point. If *vfMissingUValueV* is not set, but *vfMissingVValueV* is set, then occurrences of the value of *vfMissingVValueV* in either array causes the data point to be treated as undefined. If neither *vfMissingUValueV* nor *vfMissingVValueV* is set, then **VectorField** assumes that all values in the U data array are valid.

You may set *vfMissingVValueV* using any of the data types **VectorField** accepts for the *vfDataArray* resource.

Default: NULL

vfMissingVValueV

If *vfSingleMissingValue* is False and you set *vfMissingVValueV* to a non-NULL value, **VectorField** treats occurrences of its value in the V data array as representing an undefined value. If left unset, **VectorField** assumes that all values in the V data array are valid.

If *vfSingleMissingValue* is True and you set *vfMissingVValueV* to a non-NULL value, its value is ignored unless *vfMissingUValueV* is not set. If *vfMissingUValueV* is not set, **VectorField** treats occurrences of the value of *vfMissingVValueV* in either the U or V data arrays as representing an undefined data point. If neither *vfMissingUValueV* nor *vfMissingVValueV* is set, then **VectorField** assumes that all values in the U data array are valid.

You may set *vfMissingVValueV* using any of the data types **VectorField** accepts for the *vfDataArray* resource.

Default: NULL

vfMagMinV

By default the **VectorField** object sets *vfMagMinV* to the smallest vector magnitude computed from the individual vector components in the data arrays. However, if you set the value of this resource, **VectorField** assumes that the set value is, in fact, the minimum magnitude. You may set *vfMagMinV* using any of the data types **VectorField** accepts for the *vfDataArray* resource.

Default: <dynamic>

vfMagMaxV

By default the **VectorField** object sets *vfMagMaxV* to the largest vector magnitude computed from the individual vector components in the data arrays. However, if you set the value of this resource, **VectorField** assumes that the set value is, in fact, the maximum magnitude. You may set *vfMagMaxV* using any of the data types **VectorField** accepts for the *vfDataArray* resource.

Default: <dynamic>

vfUMinV

By default the **VectorField** object sets *vfUMinV* to the smallest value in the U data array. However, if you set the value of this resource, **VectorField** does not process the array to find the minimum value. Instead it assumes that the set value is, in fact, the minimum value. You may set *vfUMaxV* using any of the data types **VectorField** accepts for the *vfDataArray* resource.

Default: <dynamic>

vfUMaxV

By default the **VectorField** object sets *vfUMaxV* to the largest value in the U data array. However, if you set the value of this resource, **VectorField** does not process the array to find the maximum value. Instead it assumes that the set value is, in fact, the maximum value. You may set *vfUMaxV* using any of the data types **VectorField** accepts for the *vfDataArray* resource.

Default: <dynamic>

vfVMinV

By default the **VectorField** object sets *vfVMinV* to the smallest value in the V data array. However, if you set the value of this resource, **VectorField** does not process the array to find the minimum value. Instead it assumes that the set value is, in fact, the minimum value. You may set *vfVMaxV* using any of the data types **VectorField** accepts for the *vfDataArray* resource.

Default: <dynamic>

vfVMaxV

By default the **VectorField** object sets *vfVMaxV* to the largest value in the V data array. However, if you set the value of this resource, **VectorField** does not process the array to find the maximum value. Instead it assumes that the set value is, in fact, the maximum value. You may set *vfVMaxV* using any of the data types **VectorField** accepts for the *vfDataArray* resource.

Default: <dynamic>

vfXCStartV

This resource specifies the location along the X Axis of the first element of the fastest-varying dimension of the data. If the array resource *vfXArray* is set and found to be valid, **VectorField** overwrites the current value of *vfXCStartV*, replacing it with the value of the first element of *vfXArray*. Otherwise, you can set *vfXCStartV* to any value (presumably meaningful in the context of the your data). If *vfXCStartV* is greater than *vfXCEndV*, the direction of the data locations along the X Axis is opposite the direction of the positive X Axis. If *vfXArray* is `NULL` or invalid and *vfXCStartV* is not specified, **VectorField** sets *vfXCStartV* to the value 0. You may set *vfXCStartV* using any of the data types **VectorField** accepts for the *vfDataArray* resource.

Default: <dynamic>

vfXCEndV

This resource specifies the location along the X Axis of the last element of the fastest-varying dimension of the data. If the array resource *vfXArray* is set and found to be valid, **VectorField** overwrites the current value of *vfXCEndV*, replacing it with the value of the last element of *vfXArray*. Otherwise, you can set *vfXCEndV* to any value (presumably meaningful in the context of the your data). If *vfXCStartV* is greater than *vfXCEndV*, the direction of the data locations along the X Axis is opposite the direction of the positive X Axis. If *vfXArray* is `NULL` or invalid and *vfXCEndV* is not specified, **VectorField** sets *vfXCEndV* to the size of the faster-varying data dimension minus one. You may set *vfXCEndV* using any of the data types **VectorField** accepts for the *vfDataArray* resource.

Default: <dynamic>

vfYCStartV

This resource specifies the location along the Y Axis of the first element of the middle or slowest-varying dimension of the data. If the array resource *vfYArray* is set and found to be valid, **VectorField** overwrites the current value of *vfYCStartV*, replacing it with the value of the first element of *vfYArray*. Otherwise, you can set *vfYCStartV* to any value (presumably meaningful in the context of the your data). If *vfYCStartV* is greater than *vfYCEndV*, the direction of the data locations along the Y Axis is opposite the direction of the positive Y Axis. If *vfYArray* is `NULL` or invalid and *vfYCStartV* is not specified, **VectorField** sets *vfYCStartV* to the value 0. You may set *vfYCStartV* using any of the data types **VectorField** accepts for the *vfDataArray* resource.

Default: <dynamic>

vfYCEndV

This resource specifies the location along the Y Axis of the last element along the middle or slowest-varying dimension of the data. If the array resource *vfYArray* is set and found to be valid, **VectorField** overwrites the current value of *vfYCStartV*, replacing it with the value of the last element of *vfYArray*. Otherwise, you can set *vfYCStartV* to any value (presumably meaningful in the context of the your data). If *vfYCStartV* is greater than *vfYCEndV*, the direction of the data locations along the Y Axis is opposite the direction of the positive Y Axis. If *vfYArray* is `NULL` or invalid and *vfYCEndV* is not specified, **VectorField** sets *vfYCEndV* to the size of the slower-varying data dimension minus one. You may set *vfYCEndV* using any of the data types **VectorField** accepts for the *vfDataArray* resource.

Default: <dynamic>

vfXCStartSubsetV

vfXCStartSubsetV specifies the starting location along the X Axis of a rectangular sub-array of the data. *vfXCStartSubsetV* should be set to a value within the data coordinate extent as established by the values of *vfXCStartV* and *vfXCEndV*. If *vfXCStartSubsetV* is outside the coordinate extent, **VectorField** issues a warning and sets it to the value of *vfXCStartV*. If the values of *vfXCStartSubsetV* and *vfXCEndSubsetV* are oppositely directed from the direction established by *vfXCStartV* and *vfXCEndV*, **VectorField** issues a warning and exchanges their values. When you do not explicitly set a value for *vfXCStartSubsetV*:

- at initialization or when the X-Axis dimension size changes, if *vfXCStartIndex* is also not set, *vfXCStartSubsetV* takes the value of *vfXCStartV*;
- else if *vfXCStartIndex* is set, or there is a change to *vfXCStartV*, *vfXCEndV*, or *vfXArray*, *vfXCStartSubsetV* takes the data coordinate value indexed by the value of the *vfXCStartIndex* resource;
- else *vfXCStartSubsetV* retains its current value.

Since the data grid consists of data located at discrete points, the location specified by *vfXCStartSubsetV* might well fall between two data points. In this case, **VectorField** selects a beginning index for the array subset such that the requested location is included within the subset. Thus the subset array, as delivered to the receiving object, may represent a slightly larger area of the data coordinate space than requested. Get the value of the read-only resource *vfXCActualStartF* in order to find the exact start of the subset along the X Axis. You may set *vfXCSubsetStartV* using any of the data types that **VectorField** accepts for the *vfDataArray* resource.

Default: <dynamic>

vfXCEndSubsetV

vfXCEndSubsetV specifies the ending location along the X Axis of a rectangular sub-array of the data. It should be set to a value within the data coordinate extent as established by the values of *vfXCStartV* and *vfXCEndV*. If it is outside the coordinate extent, **VectorField** issues a warning and sets it to the value of *vfXCEndV*. If the values of *vfXCStartSubsetV* and *vfXCEndSubsetV* are oppositely directed from the direction established by *vfXCStartV* and *vfXCEndV*, **VectorField** issues a warning and exchanges their values. When you do not explicitly set a value for *vfXCEndSubsetV*:

- at initialization or when the X-Axis dimension size changes, if *vfXCEndIndex* is not set, *vfXCEndSubsetV* takes the value of *vfXCEndV*;
- else if *vfXCEndIndex* is set, or there is a change to *vfXCStartV*, *vfXCEndV*, or *vfXArray*, *vfXCEndSubsetV* takes the data coordinate value indexed by the value of *vfXCEndIndex*;
- else *vfXCEndSubsetV* retains its current value.

Since the data grid consists of data located at discrete points, the location specified by *vfXCEndSubsetV* might well fall between two data points. In this case, **VectorField** selects an ending index for the array subset such that the requested location is included within the subset. Thus the subset array, as delivered to the receiving object, may represent a slightly larger area of the data coordinate space than requested. Get the value of the read-only resource *vfXCActualEndF* in order to find the exact end of the subset along the X Axis. You may set *vfXCSubsetEndV* using any of the data types that **VectorField** accepts for the *vfDataArray* resource.

Default: <dynamic>

vfYCStartSubsetV

vfYCStartSubsetV specifies the starting location along the Y Axis of a rectangular sub-array of the data. It should be set to a value within the data coordinate extent as established by the values of *vfYCStartV* and *vfYCEndV*. If it is outside the coordinate extent, **VectorField** issues a warning and sets it to the value of *vfYCStartV*. If the values of *vfYCStartSubsetV* and *vfYCEndSubsetV* are oppositely directed from the direction established by *vfYCStartV* and *vfYCEndV*, **VectorField** issues a warning and exchanges their values. When you do not explicitly set a value for *vfYCStartSubsetV*:

- at initialization or when the Y-Axis dimension size changes, if *vfYCStartIndex* is not set, *vfYCStartSubsetV* takes the value of *vfYCStartV*;
- else if *vfYCStartIndex* is set, or there is a change to *vfYCStartV*, *vfYCEndV*, or *vfYArray*, *vfYCStartSubsetV* takes the data coordinate value indexed by the value of *vfYCStartIndex*;
- else *vfYCStartSubsetV* retains its current value.

Since the data grid consists of data located at discrete points, the location specified by *vfYCStartSubsetV* might well fall between two data points. In this case, **VectorField** selects a beginning index for the array subset such that the requested location is included within the subset. Thus the subset array, as delivered to the receiving object, may represent a slightly larger area of the data coordinate space than requested. Get the value of the read-only resource *vfYCActualStartF* in order to find the exact start of the subset along the Y Axis. You may set *vfYCSubsetStartV* using any of the data types that **VectorField** accepts for the *vfDataArray* resource.

Default: <dynamic>

vfYCEndSubsetV

vfYCEndSubsetV specifies the ending location along the Y Axis of a rectangular sub-array of the data. It should be set to a value within the data coordinate extent as established by the values of *vfYCStartV* and *vfYCEndV*. If it is outside the coordinate extent, **VectorField** issues a warning and sets it to the value of *vfYCEndV*. If the values of *vfYCStartSubsetV* and *vfYCEndSubsetV* are oppositely directed from the direction established by *vfYCStartV* and *vfYCEndV*, **VectorField** issues a warning and exchanges their values. When you do not explicitly set a value for *vfYCEndSubsetV*:

- at initialization or when the Y-Axis dimension size changes, if *vfYCEndIndex* is not set, *vfYCEndSubsetV* takes the value of *vfYCEndV*;
- else if *vfYCEndIndex* is set, or there is a change to *vfYCStartV*, *vfYCEndV*, or *vfYArray*, *vfYCEndSubsetV* takes the data coordinate value indexed by the value of *vfYCEndIndex*;
- else *vfYCEndSubsetV* retains its current value.

Since the data grid consists of data located at discrete points, the location specified by *vfYCEndSubsetV* might well fall between two data points. In this case, **VectorField** selects an ending index for the array subset such that the requested location is included within the subset. Thus the subset array, as delivered to the receiving object, may represent a slightly larger area of the data coordinate space than requested. Get the value of the read-only resource *vfYCActualEndF* in order to find the exact end of the subset along the Y Axis. You may set *vfYCSubsetEndV* using any of the data types that **VectorField** accepts for the *vfDataArray* resource.

Default: <dynamic>

vfXCStartIndex

This resource specifies the smaller of two indexes along the X-Axis dimension bounding a rectangular sub-array of the data. Explicitly setting *vfXCStartSubsetV* overrides any value set for *vfXCStartIndex*, and causes it to be set to the array index value associated with the closest data point such that the requested data coordinate location is included within the subset. Otherwise, when you set *vfXCStartIndex*, the data coordinate value indexed by its value is placed into *vfXCStartSubsetV*. *vfXCStartIndex* retains its value when the data change, unless the X-Axis dimension size also changes. In this case, assuming neither *vfXCStartSubsetV* nor *vfXCStartIndex* is explicitly set at the same time, *vfXCStartIndex* is set to 0.

Default: <dynamic>

vfXCEndIndex

This resource specifies the larger of two indexes along the X-Axis dimension bounding a rectangular sub-array of the data. Explicitly setting *vfXCEndSubsetV* overrides any value set for *vfXCEndIndex*, and causes it to be set to the array index value associated with the closest data point such that the requested data coordinate location is included within the subset. Otherwise, when you set *vfXCEndIndex*, the data coordinate value indexed by its value is placed into *vfXCEndSubsetV*. *vfXCEndIndex* retains its value when the data change, unless the X-Axis dimension size also changes. In this case, assuming neither *vfXCEndSubsetV* nor *vfXCEndIndex* is explicitly set at the same time, *vfXCEndIndex* is set to the size of the X-Axis dimension minus one.

Default: <dynamic>

vfYCStartIndex

This resource specifies the smaller of two indexes along the Y-Axis dimension bounding a rectangular sub-array of the data. Explicitly setting *vfYCStartSubsetV* overrides any value set for *vfYCStartIndex*, and causes it to be set to the array index value associated with the closest data point such that the requested data coordinate location is included within the subset. Otherwise, when you set *vfYCStartIndex*, the data coordinate value indexed by its value is placed into *vfYCStartSubsetV*. *vfYCStartIndex* retains its value when the data change, unless the Y-Axis dimension size also changes. In this case, assuming neither *vfYCStartSubsetV* nor *vfYCStartIndex* is explicitly set at the same time, *vfYCStartIndex* is set to 0.

Default: <dynamic>

vfYCEndIndex

This resource specifies the larger of two indexes along the Y-Axis dimension bounding a rectangular sub-array of the data. Explicitly setting *vfYCEndSubsetV* overrides any value set for *vfYCEndIndex*, and causes it to be set to the array index value associated with the closest data point such that the requested data coordinate location is included within the subset. Otherwise, when you set *vfYCEndIndex*, the data coordinate value indexed by its value is placed into *vfYCEndSubsetV*. *vfYCEndIndex* retains its value when the data change, unless the Y-Axis dimension size also changes. In this case, assuming neither *vfYCEndSubsetV* nor *vfYCEndIndex* is explicitly set at the same time, *vfYCEndIndex* is set to the size of the Y-Axis dimension minus one.

Default: <dynamic>

vfXCActualStartF

When a subset of the data is specified by location in data coordinate space using *vfXCEndSubsetV*, this location might well fall between two data points. In this case, **VectorField** selects an ending index for the array subset such that the requested location is included within the subset. Thus the subset array, as delivered to the receiving object, may represent a slightly larger area of the data coordinate space than requested. The read-only resource *vfXCActualEndF* always contains the actual ending location of the data subset along the X Axis.

Default: <dynamic>

vfXCActualEndF

When a subset of the data is specified by location in data coordinate space using *vfYCStartSubsetV*, this location might well fall between two data points. In this case, **VectorField** selects a beginning index for the array subset such that the requested location is included within the subset. Thus the subset array, as delivered to the receiving object, may represent a slightly larger area of the data coordinate space than requested. The read-only resource *vfYCActualStartF* always contains the actual starting location of the data subset along the Y Axis.

Default: <dynamic>

vfYCActualStartF

When a subset of the data is specified by location in data coordinate space using *vfYCStartSubsetV*, this location might well fall between two data points. In this case, **VectorField** selects a beginning index for the array subset such that the requested location is included within the subset. Thus the subset array, as delivered to the receiving object, may represent a slightly larger area of the data coordinate space than requested. The read-only resource *vfYCActualStartF* always contains the actual starting location of the data subset along the Y Axis.

Default: <dynamic>

vfYCActualEndF

When a subset of the data is specified by location in data coordinate space using *vfYCEndSubsetV*, this location might well fall between two data points. In this case, **VectorField** selects an ending index for the array subset such that the requested location is included within the subset. Thus the subset array, as delivered to the receiving object, may represent a slightly larger area of the data coordinate space than requested. The read-only resource *vfYCActualEndF* always contains the actual ending location of the data subset along the Y Axis.

Default: <dynamic>

vfXCStride

When *vfXCStride* has a value greater than one, the **VectorField** object creates a new array containing only columns indexed along the X dimension by successive multiples (starting with 0) of the *vfXCStride* value added to the starting index. The starting index is based on the current state of the applicable subsetting resources; it is 0 if no array subsetting is in effect. The resulting array is passed to the receiving plot object.

Note that if ending index minus starting index is not an exact multiple of the *vfXCStride* value, then the data coordinate range may be reduced slightly.

Default: 1

vfYCStride

When *vfYCStride* has a value greater than one, the **VectorField** object creates a new array containing only rows indexed along the Y dimension by successive multiples (starting with 0) of the *vfYCStride* value added to the starting index. The starting index is based on the current state of the applicable subsetting resources; it is 0 if no array subsetting in effect. The resulting array is passed to the receiving plot object.

Note that if ending index minus starting index is not an exact multiple of the *vfYCStride* value, then the data coordinate range may be reduced slightly.

Default: 1

View class resource descriptions

vpXF

vpXF specifies the location of left edge of the **View** object's bounding box in NDC space.

Default: 0.2

vpYF

vpYF specifies the location of top edge of the **View** object's bounding box in NDC space.

Default: 0.8

vpWidthF

vpWidthF specifies the width of **View** object's bounding box in NDC units.

Default: 0.6

vpHeightF

vpHeightF specifies the height of **View** object's bounding box in NDC units.

Default: 0.6

vpOn

Set *vpOn* False to disable the **Draw** method of a **View** object. When its **Draw** method is disabled, the **view**, its **View** class children, its overlays, and its added annotations do not appear on the **workstation** when a **Draw** is executed.

Default: True

vpKeepAspect

While the boolean resource *vpKeepAspect* is True, the **View** object keeps its initial shape (aspect ratio); however you may modify its size resources. If you modify either or both the size resources, *vpWidthF* and *vpHeightF*, **View** will constrain its new bounding box to the largest box with an aspect ratio matching the original shape that can be inscribed within a box of the specified size. When *vpKeepAspect* is False, **View** places no constraints on the shape of its bounding box when you modify the size resources.

Default: False

vpUseSegments

When the boolean resource *vpUseSegments* is set True for a **View** class object, the object will, if it is able, create a segment and draw into it while it draws to the designated **Workstation** object. The segment is stored as a file, and contains the low-level commands required to re-create the object, possibly with transformations to the position, size, or shape applied. When you next draw the object, assuming none of the object's resources other than *vpXF*, *vpYF*, *vpWidthF*, or *vpHeightF* have been modified, it will recreate its image based on information stored in the segment. Using segments can substantially shorten the time required to perform a draw when the plot contains elements, such as filled maps, that require considerable computation to generate initially. Note that because the transformations differ slightly, a segment drawn at a different size from the size at which it was created may not match in every detail the plot resulting from a new draw of the object at that size.

Default: False

vpAnnoManagerId

If the **View** object is currently functioning as an external annotation of a Plot Object, this read-only resource contains the id of the **AnnoManager** object used to manage the **View** object's location and size. If the **View** object is not currently an annotation, the value of the resource is set to `NullObjId` (0).

Default: False

NcgmWorkstation class resource descriptions

wkMetaName

Specifies the name and path of the output metafile. By default a file called *gmeta* will be created in the current working directory. The **NcgmWorkstation** object will attempt to resolve pathnames that start with the "~" character.

Default: "gmeta"

PSWorkstation class resource descriptions

wkPSFormat

This resource is used to specify which PostScript format should be produced. Currently, the **PSWorkstation** can produce generic "ps", EPS (Encapsulated PostScript) "eps", and EPSI (Encapsulated PostScript Interchange format) "epsi".

EPS is a specialized format used primarily for importing PostScript into other applications. EPS files contain information in them, that allow an importing application to know the extent of the marks on the PostScript page. They are also restricted to a single page of output in them.

EPSI are EPS files that have a "preview bitmap" that represents the PostScript image contained in the file. This bitmap can be used by an importing application to display quickly a picture of the imported file. This is used by applications that do not have a built-in PostScript interpreter. As of Version 4.0, the "preview bitmap" output by the **PSWorkstation** does not represent the picture contained in the EPSI file, but rather is simply a bitmap containing the message: **APPLICATION CANNOT DISPLAY NCAR GRAPHIC HERE. VIEW THE POSTSCRIPT OUTPUT TO VERIFY SUCCESSFUL IMPORT.**

Default: "ps"

wkVisualType

This resource is used to indicate if the PostScript file should produce "color", or "monochrome" output. Color output can be displayed on a grayscale printer, and the colors will be mapped to shades of gray. If "monochrome" output is selected, all colors, except the background color, will be mapped to the foreground color. If you are plotting a color image on a grayscale printer and find the results unsatisfactory since colors are getting mapped to light shades of gray that are difficult to see, it may be better to produce the picture in "monochrome".

Default: "color" **wkColorModel**

This resource specifies the color model used to describe colors in the output PostScript file. The choices are "rgb" for the Red-Green-Blue color model or "cmyk" for the Cyan-Magenta-Yellow-Black color model. Note that this resource affects the output file only. Elements of the *wkColorMap* array be set using the RGB model or named colors only.

Default: "rgb"

wkOrientation

This resource is used to indicate if the PostScript output should be produced in "portrait" mode (pictures displayed along the width of the paper or viewing window) or "landscape" mode (pictures displayed along the length).

Default: "portrait"

wkPSFileName

This resource is used to specify the name of the PostScript file to output. If a *wkPSFileName* of "stdout" is given, the **PSWorkstation** understands the output should be sent to the stream associated with the `stdout` FILE pointer in the UNIX environment. The **PSWorkstation** object will attempt to resolve pathnames that start with the "~" character.

Default: <dynamic>

This resource defaults to the *name* of the **PSWorkstation** object with the *wkPSFormat* as a suffix. For example, "workname.ps".

wkFullBackground

If this resource is True, then the entire output page will have the background color applied to it. If it is False, then only the normalized device coordinate viewport will be effected by the background color. The background is the first thing drawn on the page. Therefore, if this resource is modified after something has been drawn, it will not change the background for the current page. It will effectively change for the next page of the output.

Default: False

wkPSResolution

This resource indicates the resolution of the internal coordinate space of the PostScript file. This resource is used to increase or decrease the resolution of the output. The effective resolution you get on a particular PostScript printer or screen display depends on the resolution of the internal PostScript coordinate system as well as the resolution of the output device. The units of this resource are compatible with DPI (Dots Per Inch). If wkPSResolution is set to a number that is greater than or equal to the resolution of the printer, then you can be sure that you are making full use of the resolution of the printer. It is important to realize that printers that have a lower resolution can print PostScript files that have a higher internal coordinate space resolution without problems, since the internal coordinate space resolution just indicates the resolution of the points within the file's coordinate space. If the PostScript is generated with a lower internal coordinate space resolution then the printer, the full resolution of the printer will not be utilized.

One advantage of lowering the *wkPSResolution* would be to make the actual size of the PostScript file smaller, although the difference is rather modest.

Default: 1800

Positioning output on the page

The following resources specify where on the output page, the normalized device coordinate viewport should be placed. They indicate the Lower X, Lower Y, Upper X, and Upper Y values of the viewport. The values are specified in the default PostScript coordinate space where one unit corresponds to 1/72 of an inch. This means that an 8.5" x 11" page would be addressed from (0,0) in the lower left corner to (612,792) in the upper right corner (The page is always viewed in portrait mode for this purpose). In order to maintain aspect ratio (insure that circles do not become ellipses, and squares do not become rectangles), these values should indicate a square area of the page.

wkDeviceLowerX

Defines the Lower X value of the normalized device coordinate viewport on the page. It is specified in PostScript coordinate space where one unit corresponds to 1/72 of an inch. 0 addresses the left side of an 8.5" x 11" page, and 612 addresses the right side. For details on using the PostScript workstation positioning resources, see the appropriate section in the NCAR GKS User's Guide for NCAR GKS-0A Graphics. It is particularly important to consult the section in the GKS manual on using the full page if you are using the PostScript workstation positioning resources with a PostScript landscape workstation.

Default: 36

wkDeviceLowerY

Defines the Lower Y value of the normalized device coordinate viewport on the page. It is specified in PostScript coordinate space where one unit corresponds to 1/72 of an inch. 0 addresses the bottom side of an 8.5" x 11" page, and 792 addresses the top side.

Default: 126

wkDeviceUpperX

Defines the Upper X value of the normalized device coordinate viewport on the page. It is specified in PostScript coordinate space where one unit corresponds to 1/72 of an inch. 0 addresses the left side of an 8.5" x 11" page, and 612 addresses the right side.

Default: 576

wkDeviceUpperY

Defines the Upper Y value of the normalized device coordinate viewport on the page. It is specified in PostScript coordinate space where one unit corresponds to 1/72 of an inch. 0 addresses the bottom side of an 8.5" x 11" page, and 792 addresses the top side.

Default: 666

Workstation class resource descriptions

wkColorMap

Use this resource to set or retrieve the colormap for a **Workstation** object. To set this resource, use a two-dimensional array of type NhlTFloat, an array of color specification formats, or a pre-defined colormap name. If you retrieve this resource, it will be a two-dimensional array of type NhlTFloat.

For a two-dimensional array, the fastest varying dimension contains three elements that define (in order) the red, green, and blue components of a single color. Each component should be within the range 0.0 through 1.0 inclusive. If the red component of a color is specified with a negative number, then the color is considered to be a missing value, and that index of the colormap will default. In all other cases, an error message will be generated if values are outside the range of 0.0 through 1.0 inclusive.

For an array of color specification formats, each element must be enclosed in double quotes and be either an RGB triplet as described above, or a named color from the $NCARG_ROOT/lib/ncarg/database/rgb.txt file.

For a pre-defined colormap name, set the resource to one of the following pre-defined colormap names:

"default"
: The "default" colormap contains 32 entries. The first two entries default as described above. Indexes 2 through 25 define a cyclic colormap. This is a colormap with a uniform distribution of colors, but each successive element is choosen to be as different as possible from the previous element. Indexes 26 through 31 define a grayscale cyclic colormap.

"cyclic"
: The "cyclic" colormap contains eight entries. The first two entries default as described above. The remaining entries are red, green, blue, yellow, cyan, and magenta.

"gscyclic"
: The "gscyclic" colormap is a Gray Scale (gs) version of the "cyclic" colormap. It also contains eight entries. The first two entries default as described above. The next six entries are different shades of gray, with each successive value chosen to contrast as much as possible with the previous entry.

"gsltod"
: The "gsltod" colormap is a Gray Scale colormap, from Light TO Dark (ltod). It contains 33 entries. The background color is set to black. The foreground color is set to white. Each successive entry after that gets slightly darker until entry 32 which is nearly black.

"gsdtol"

The "gsdtol" colormap is a Gray Scale colormap, from Dark TO Light (dtol). It contains 33 entries, and is the inverse of the "gsltod" colormap.

"uniform"

The "uniform" colormap contains 175 entries. The first two entries default as described above. The remaining entries define a uniform colormap with red varying fastest, green next, and blue slowest. It starts with the maximium values for each (skipping white since it is either the foreground or the background), and then cycles through five levels of red, seven levels of green, and five levels of blue.

"temp1"

The "temp1" colormap contains 63 entries. The first two entries default as described above. The remaining entries vary from white to blue to green to red. This colormap may be useful for displaying plots with temperature-related information.

"psgcap"

The "psgcap" colormap contains 240 entries. It is based on the colormap that is used in the color PostScript graphcap for ctrans.

If you do not set *wkColorMap*, the **Workstation** object will use a default color map. The first triple of the *wkColorMap* array (index 1 of the slow-varying dimension using the Fortran interface, or index 0 using the C interface) is the background color for the **Workstation** and is accessed as HLU color index 0. In the default colormap, the color is dynamically determined based on the **Workstation** subclass. For example, the background color for the **XWorkstation** is black, but the background color for the **PSWorkstation** is white. The background color can also be set using the *wkBackgroundColor* resource as a convenience.

The second triple of the *wkColorMap* array (index 2 of the slow-varying dimension using the Fortran interface, or index 1 using the C interface) is the foreground color for the **Workstation** and is accessed as HLU color index 1. The foreground color is also dynamically determined. It defaults to either white or black, whichever is determined to contrast more with the background color.

Default: "default"

wkColorMapLen

This read-only resource allows you to determine the number of elements currently in the *wkColorMap* without retrieving the whole table. This is one more than the largest currently defined index.

Default: <dynamic>

wkBackgroundColor

This resource can be a one-dimensional array containing three elements of type NhlTFloat that specify (in order) the red, green, and blue components (an RGB triplet) of the **Workstation** background color, or a string containing a color name from the file $NCARG_ROOT/lib/ncarg/database/rgb.txt. You may set *wkBackgroundColor* only when the **Workstation** object is created. The background color is accessed for drawing purposes using HLU color index 0 (or the defined constant `NhlBACKGROUND` when using the C language interface). If defining this resource with an RGB triplet, each element must be in the range 0.0 through 1.0 inclusive. If any element is invalid, the **Workstation** object issues a warning, and reverts to the default background color value.

Default: (0.0, 0.0, 0.0) (Black)

wkForegroundColor

This resource can be a one-dimensional array containing three elements of type NhlTFloat that specify (in order) the red, green, and blue components (an RGB triplet) of the **Workstation** foreground color, or a string containing a color name from the file $NCARG_ROOT/lib/ncarg/database/rgb.txt. The foreground color is equivalent to HLU color index 1 (the defined constant `NhlFOREGROUND` when using the C language interface). It is the first element of the *wkColorMap* array (index 0 of the array when using the C interface). Unlike *wkBackgroundColor*, you may set *wkForegroundColor* at any time during the life of the **Workstation** object.

If you set *wkForegroundColor* and *wkColorMap* at the same time, the set value of the first element of *wkColorMap* will be replaced with the value of *wkForegroundColor*. If neither *wkForegroundColor* nor *wkColorMap* is set when the **Workstation** object is created, the foreground is determined dynamically.

Default: <dynamic>
If background color is determined to be closer to black than white, the foreground color is set to white (1.0, 1.0, 1.0); otherwise the foreground color is set to black (0.0, 0.0, 0.0).

wkDashTableLength

This read-only resource contains the number of currently available unique dash patterns.

Default: <dynamic>

wkFillTableLength

This read-only resource contains the number of currently available unique fill patterns.

Default: <dynamic>

wkMarkerTableLength

This read-only resource contains the number of currently available unique markers.

Default: <dynamic>

wkGksWorkId

This read-only resource contains the low-level GKS workstation identifier. It can be used to mix calls to the low-level NCAR Graphics library and the GKS library with the HLU library. The HLU library expects the GKS workstation to be in a deactivated state. Therefore, any set of low-level calls occurring between HLU library calls should be bracketed by a GKS activate workstation call and a GKS deactivate workstation call.

Default: <dynamic>
This value is determined by the low-level GKS library.

wkDefGraphicStyleId

This read-only resource contains the default **GraphicStyle** object that is created along with the **Workstation** itself. If you call an *immediate mode* graphics primitive function without specifying an explicit **GraphicStyle** object, the **Workstation** uses the default **GraphicStyle** object to determine the attributes used to render the graphics primitive. You may modify these attributes by retrieving the default **GraphicStyle** object, and then setting its resources as desired.

Default: <dynamic>

XWorkstation class resource descriptions

wkWindowId

This resource allows X window programs to pass in to the HLU library an X-Window Id into which output will be drawn. If the aspect ratio of the window is not square NCAR graphics will inscribe a square output frame centered in the window. If you do not set *wkWindowId*, the XWorkstation object will create and manage its own window. This resource can only be set on create.

Default: NULL

wkXColorMode

This resource is used to specify what type of X color allocation scheme the **XWorkstation** object should use. If this resource is set to "share", then the **XWorkstation** object will attempt to use the "default" colormap for the display. Since most X displays only use an 8 bit - PseudoColor colormap, only 256 unique colors can be displayed at a time. This means that it is unlikely that the **XWorkstation** will be able to have 256 unique colors, since other applications will take some of the available colors.

If this resource is set to "private", then the **XWorkstation** object will create a colormap to be used exclusively with the output window. This means that a full 256 colors will be available to the **XWorkstation** output window. However, this can have some interesting side effects for other windows. The colormap for the **XWorkstation** output window will be installed when that window has the "focus". When it doesn't have the "focus", it's colormap will not be installed. Therefore, the **XWorkstation** output will not look correct unless its window has the "focus". Also, when it has the "focus", other windows on the display are likely to look strange.

If this resource is set to "mixed", then the **XWorkstation** object will attempt to use the "default" colormap as long as it can. Once a color allocation from the X server fails, it will create a private colormap, and behave as if this resource were set to "private". This change is completely dynamic, and the user doesn't have any control as to exactly when the object switches from using the "default" colormap, to using a "private" one. However, everything drawn up to that point will still be seen within the X window in the correct colors as long as the window has the "focus".

Default: <dynamic>
wkXColorMode defaults to "mixed" unless the *wkWindowId* resource is set, when it is forced to "share".

wkPause

This enables and disables the workstation pause feature. When True the **NhlFrame** call will pause until the user has pressed a key or pressed a mouse button. If the *wkWindowId* is set when the window is created, causing the **XWorkstation** to draw into an externally created window, *wkPause* is constrained to False. Otherwise a conflict would result in the event management of the window.

Default: <dynamic>
wkPause defaults to True unless the *wkWindowId* resource is set.

Workspace class resource descriptions

wsMaximumSize

This resource specifies the total size in bytes that is allowed to be allocated at any single time for all workspaces managed by the **Workspace** object during the execution of an HLU program. If the **Workspace** object is asked to allocate or reallocate a workspace such that this value would be exceeded, the **Workspace** refuses to attempt the allocation and returns a fatal error. In order to modify *wkMaximumSize* you must first get the id of the **Workspace** object using the **NhlGetWorkspaceObjectId** function.

Default: 16777216

wsThresholdSize

Whenever the sum of the size in bytes of all currently allocated workspaces exceeds the value of the resource *wsThresholdSize*, the **Workspace** object attempts to reduce the total allocation below the threshold size by freeing workspaces that are not currently in use. If it is necessary to keep the data in a workspace, the contents of the workspace will first be written to a file. The total size allocated to workspaces may exceed *wkThresholdSize* if the total size of workspaces currently in use exceeds the threshold size. In order to modify *wkThresholdSize* you must first get the id of the **Workspace** object using the **NhlGetWorkspaceObjectId** function.

Default: 4194304

wsCurrentSize

You can determine the total size currently allocated for all workspaces managed by the **Workspace** object at any point in an HLU program by retrieving the value of the read-only resource *wkCurrentSize*. To do this, you must first get the id of the **Workspace** object using the **NhlGetWorkspaceObjectId** function.

Default: 0

XyPlot class resource descriptions

xyCoordData

Specifies the array of data object ids that provide the X/Y coordinate pairs to be plotted. This resource is a data resource partially managed with the help of the **DataComm** superclass.

Default: NULL

xyCoordDataSpec

This resource is a retrievable resource only. It is used to allow the user a greater amount of configurability for how each coordinate pair array is displayed in the **XyPlot**. After adding DataItems to the **XyPlot** xyCoordData resource, this resource can be retrieved to get an array of **XyDataSpec** object ids. This array is ordered so that the elements of this array correspond directly with the elements of the xyCoordData array resource. The XyDataSpec resources can be set to each object in this array. This will determine exactly how the corresponding **DataItem** in the xyCoordData resource will be rendered in the **XyPlot**.

Default: NULL

xyCurveDrawOrder

This resource of type NhlTDrawOrder determines when the curves representing X/Y coordinate data are drawn relative to other elements of the plot. Ordinarily, you only need to adjust the draw order of the curves when an **XyPlot** is combined with other plots in an overlay sequence. There are three choices:

`PreDraw`
: Draw the **XyPlot** curves before the standard draw; the lines will be overlaid by any subsequently drawn elements.

`Draw`
: Draw the curves during the standard draw; the lines will overlay any elements drawn during the predraw phase, but will underlie elements drawn during the postdraw phase.

`PostDraw`
: Draw the curves after the standard draw; the lines will overlay any elements drawn during the predraw and draw phases.

Default: `Draw`

xyXStyle

Sets the style for the X-Axis of the **XyPlot**. The styles are `Log`, `Linear`, `Irregular`, `Time`, and `Geographic`. These are identical to the styles available through the **TickMark** object. **XyPlot** does not allow the programmer to set the top and bottom styles directly like it is done in the **TickMark** object. To achieve multiple scales (i.e. Pressure on top, Km on bottom), either the tmXBMode or the tmXTMode resource can be set to `Explicit`, and the alternate scale can be provided using the tmXBValues and tmXBLabels resources or the tmXTValues and tmXTLabels resources.

Default: `Linear`

xyYStyle

Sets the style for the Y Axis of the **XyPlot**. The styles are `Log`, `Linear`, `Irregular`, `Time`, and `Geographic`. These are identical to the styles available through the **TickMark** object. **XyPlot** does not allow the programmer to set the left and right styles directly like it is done in the **TickMark** object. To achieve multiple scales (i.e. Pressure on left, Km on right), either the tmYBMode or the tmYTMode resource can be set to `Explicit`, and the alternate scale can be provided using the tmYBValues and tmYBLabels resources or the tmYTValues and tmXTLabels resources.

Default: `Linear`

xyXIrregularPoints

An array of floats that specify a discrete representation of an irregular coordinate system. The values in this array must be monotonically increasing or decreasing. This resource is only used if xyXStyle is `Irregular`.

Default: NULL

xyYIrregularPoints

An array of floats that specify a discrete representation of an irregular coordinate system. The values in this array must be monotonically increasing or decreasing. This resource is only used if xyYStyle is `Irregular`.

Default: NULL

xyXIrrTensionF

Specifies the tension to apply to the spline approximation used to determine the transformation for the xyXIrregularPoints. This resource is only used if xyXStyle is `Irregular`.

Default: 2.0

xyYIrrTensionF

Specifies the tension to apply to the spline approximation used to determine the transformation for the xyYIrregularPoints. This resource is only used if xyYStyle is `Irregular`.

Default: 2.0

xyComputeXMin

If this resource is True, then the trXMinF resource will be recalculated every time the xyCoordData resource is modified. If this resource is False, then the *trXMinF* resource will not be recalculated each time the *xyCoordData* resource is modified.

Default: <dynamic>

If the programmer specifies the *trXMinF* resource, then the default for this resource will be False. If the programmer doesn't specify the *trXMinF* resource, then the default for this resource will be True.

xyComputeXMax

If this resource is True, then the trXMaxF resource will be recalculated every time the xyCoordData resource is modified. If this resource is False, then the *trXMaxF* resource will not be recalculated each time the *xyCoordData* resource is modified.

Default: <dynamic>

If the programmer specifies the *trXMaxF* resource, then the default for this resource will be False. If the programmer doesn't specify the *trXMaxF* resource, then the default for this resource will be True.

xyComputeYMin

If this resource is True, then the trYMinF resource will be recalculated every time the xyCoordData resource is modified. If this resource is False, then the *trYMinF* resource will not be recalculated each time the *xyCoordData* resource is modified.

Default: <dynamic>

If the programmer specifies the *trYMinF* resource, then the default for this resource will be False. If the programmer doesn't specify the *trYMinF* resource, then the default for this resource will be True.

xyComputeYMax

If this resource is True, then the trYMaxF resource will be recalculated every time the xyCoordData resource is modified. If this resource is False, then the *trYMaxF* resource will not be recalculated each time the *xyCoordData* resource is modified.

Default: <dynamic>
If the programmer specifies the *trYMaxF* resource, then the default for this resource will be False. If the programmer doesn't specify the *trYMaxF* resource, then the default for this resource will be True.

XyDataSpec class resource descriptions

xyDashPattern

This resource determines the dash pattern index that all the curves derived from the associated data object will be drawn with if xyMonoDashPattern is True.

Default: `SolidLine`

xyDashPatterns

An array of integer dash pattern indexes to use when drawing the corresponding curves in the associated data object. If there are more curves than the number of dash pattern indexes in this resource, the remaining curves will be drawn using the xyDashPattern resource.

This resource is only used if the xyMonoDashPattern resource is False.

Default: NULL

xyMonoDashPattern

If this resource is True, then all the curves in the associated data object will be drawn with the dash pattern specified by the xyDashPattern resource. Otherwise, the curves are drawn with the dash pattern specified by the corresponding index in the xyDashPatterns array resource.

Default: False

xyMarkLineMode

This resource determines if markers, curves, or both markers and curves will be displayed for all coordinate arrays derived from the associated data object if xyMonoMarkLineMode is True.

Default: `Lines`

xyMarkLineModes

An array of NhlMarkLineMode enumerations to use when drawing the corresponding coordinate pair arrays in the associated data object. If there are more coordinate pair arrays than the number of elements in this resource, the remaining coordinate pair arrays will be drawn using the value of the xyMarkLineMode resource.

Default: NULL

xyMonoMarkLineMode

If this resource is True, then all the coordinate pair arrays in the associated data object will be drawn as specified by the xyMarkLineMode resource. Otherwise, the coordinate pair arrays are drawn in the NhlTMarkLineMode specified by the corresponding index in the xyMarkLineModes array resource.

Default: False

xyExplicitLegendLabels

An array of character strings to use as labels in the **XyPlot** legend. The legend labels will default to the values in the xyExplicitLabels resource. If there are more coordinate pair arrays than the number of elements in this resource, the remaining coordinate pair arrays will also be labeled with the corresponding value of the xyExplicitLabels resource.

Default: NULL

xyLineColor

This resource determines what color index all the curves derived from the associated data object will be drawn with if xyMonoLineColor is True.

Default: `Foreground`

xyLineColors

An array of color indexes to use when drawing the corresponding curves in the associated data object. If there are more curves than the number of color indexes in this resource, the remaining curves will be drawn using the xyLineColor resource.

This resource is only used if the xyMonoLineColor resource is False.

Default: NULL

xyMonoLineColor

If this resource is True, then all the curves in the associated data object will be drawn in the color specified by the xyLineColor resource. Otherwise, the curves are drawn in the color specified by the corresponding index in the xyLineColors array resource.

Default: False

xyLineDashSegLenF

This resource indicates the length of each segment of a dash pattern. It is the length in NDCs before the dash pattern repeats itself. This resource automatically scales with changes in the size of the viewport of the **XyPlot**. If line labels are being drawn, this is also the length between the end of one label and the start of the next.

Default: <dynamic>
This resource defaults to 25 percent of the average of the XyPlot's width and height.

xyLineLabelFontHeightF

This allows the user to set the size of the characters used for the curve labels when the data-dependent resource xyLabelMode is set to LETTERED or CUSTOM. The size of the font is expressed in NDCs and scales with changes in the size of the viewport of the **XyPlot** object.

Default: <dynamic>
This resource defaults to 1.5 percent of the average of the XyPlot's width and height.

xyLineLabelFontColor

This resource indicates what color index to draw the line labels if xyLabelMode is set to `Lettered` or `Custom`. This value is only used if xyMonoLineLabelFontColor is True, or the xyLineLabelFontColors array doesn't specify enough values.

Default: `Foreground`

xyLineLabelFontColors

This resource indicates what color index to draw the line labels in if xyLabelMode is set to `Lettered` or `Custom`. It is an array of NhlColorIndexes that indicate the color for the line labels in the corresponding coordinate arrays in the associated data object. If there are more coordinate arrays than the number of NhlColorIndexes in this resource, the remaining coordinate arrays will be drawn using the xyLineLabelFontColor resource.

Default: NULL

xyMonoLineLabelFontColor

If this resource is True, then all the line labels for the corresponding coordinate arrays in the associated data object will be drawn with the color specified by the xyLineLabelFontColor resource. Otherwise, each coordinate array's markers are drawn with the color specified by the corresponding index in the xyLineLabelFontColors array resource.

Default: False

xyLabelMode

This can be set to either `NoLabels`, `Lettered`, or `Custom`. `NoLabels` means that the curves from the associated **DataItem** will not be labeled. `Lettered` means that the curves will be labeled with capital letters in the order in which they appear in the associated **DataItem**. The current letter is retained so that if the next **XyDataSpec** in the list is using `Lettered`, the letter will not be reused until the entire alphabet has been used. (Note: This means that `Lettered` curves do not necessarily retain the same label when data are added or removed from the xyCoordData resource. `Custom` means that the curves will be labeled with the corresponding strings provided by the xyExplicitLabels resource. If there is no corresponding string, the curves will be labeled by using the *name* of the **XyDataSpec** object and appending the curve index.

Default: `NoLabels`

xyExplicitLabels

An array of character strings to place inside the corresponding curve in the associated **DataItem**. If there are more curves than the number of strings in this resource, the remaining curves will be labeled by using the *name* of the **XyDataSpec** object and appending the curve index.

Default: NULL

xyLineThicknessF

This resource sets the linewidth scale factor for curves derived from the associated data object if xyMonoLineThickness is True.

Default: 1.0

xyLineThicknesses

An array of floating point numbers that are used as linewidth scale factors when drawing the corresponding curves in the associated data object. If there are more curves than the number of floating point numbers in this resource, the remaining curves will be drawn using the xyLineThicknessF resource.

This resource is only used if the xyMonoLineThickness resource is False.

Default: NULL

xyMonoLineThickness

If this resource is True, then all the curves in the associated data object will be drawn with the width factor specified by the xyLineThicknessF resource. Otherwise, the curves are drawn with the thickness specified by the corresponding index in the xyLineThicknesses array resource.

Default: False

xyMarker

This resource indicates which marker index all the coordinate pair arrays derived from the associated data object will be drawn with if xyMonoMarker is True.

Default: 0

xyMarkers

An array of marker indexes to use when drawing the corresponding coordinate arrays in the associated data object. If there are more coordinate arrays than the number of enumerations in this resource, the remaining coordinate arrays will be drawn using the value of the xyMarker resource.

Default: NULL

xyMonoMarker

If this resource is True, then all the coordinate arrays in the associated data object will be drawn with the marker index specified by the xyMarker resource. Otherwise, the coordinate arrays are drawn with the marker specified by the corresponding index in the xyMarkers array resource.

Default: False

xyMarkerColor

This resource indicates what color index all markers drawn using the coordinate arrays from the associated data object will be drawn with if xyMonoMarkerColor is True.

Default: `Foreground`

xyMarkerColors

An array of color indexes to use when drawing the markers for the corresponding coordinate arrays in the associated data object. If there are more coordinate arrays than the number of color indexes in this resource, the remaining coordinate arrays will be drawn using the xyMarkerColor resource.

Default: NULL

xyMonoMarkerColor

If this resource is True, then all the markers for the corresponding coordinate arrays in the associated data object will be drawn in the color specified by the xyMarkerColor resource. Otherwise, each coordinate array's markers are drawn in the color specified by the corresponding index in the xyMarkerColors array resource.

Default: False

xyMarkerSizeF

This resource indicates the size for all markers drawn using the coordinate arrays from the associated data object if xyMonoMarkerSize is True.

The size is in NDCs.

Default: .01

xyMarkerSizes

An array of floats that indicate the size for the markers in the corresponding coordinate arrays in the associated data object. If there are more coordinate arrays than the number of floats in this resource, the remaining coordinate arrays will be drawn using the xyMarkerSizeF resource.

The size is in NDCs.

Default: NULL

xyMonoMarkerSize

If this resource is True, then all the markers for the corresponding coordinate arrays in the associated data object will be drawn with the size specified by the xyMarkerSizeF resource. Otherwise, each coordinate array's markers are drawn with the size specified by the corresponding index in the xyMarkerSizes array resource.

Default: False

xyMarkerThicknessF

This resource sets the linewidth scale factor for markers derived from the associated data object if xyMonoMarkerThickness is True.

Default: 1.0

xyMarkerThicknesses

An array of floating point numbers that are used as linewidth scale factors to use when drawing the markers in the corresponding coordinate pair arrays in the associated data object. If there are more coordinate pair arrays than the number of floating point numbers in this resource, the remaining coordinate pair arrays will be drawn using the xyMarkerThicknessF resource.

This resource is only used if the xyMonoMarkerThickness resource is False.

Default: NULL

xyMonoMarkerThickness

If this resource is True, then all the curves in the associated data object will be drawn with the width factor specified by the xyMarkerThicknessF resource. Otherwise, the curves are drawn with the thickness specified by the corresponding index in the xyMarkerThicknesses array resource.

Default: False

xyLineLabelFont

This resource of type NhlTFont specifies the font used to render line labels.

Note that there is only a scalar version of this resource. If the **XyPlot** object references more than one **DataItem** object, the **XyDataSpec** object referenced by the first element of the *xyCoordDataSpec* array will provide the resource value for all line labels.

Default: "pwritx"

xyLineLabelFontAspectF

This resource determines the shape of the line label characters. Values increasing from 1.0 result in thinner characters. Values decreasing from 1.0 make the characters wider. Values less than or equal to 0.0 result in a WARNING message and a restoration of the default value.

Note that there is only a scalar version of this resource. If the **XyPlot** object references more than one **DataItem** object, the **XyDataSpec** object referenced by the first element of the *xyCoordDataSpec* array will provide the resource value for all line labels.

Default: 1.3125

xyLineLabelFontThicknessF

Sets the thickness of the line used to draw line label text. The value acts as a multiplier of a (device-dependent) unit thickness. This resource is ignored when the *xyLineLabelLabelFont* specifies a filled font (font indexes 21-22, 25-26, 29-30, and 33-37).

Note that there is only a scalar version of this resource. If the **XyPlot** object references more than one **DataItem** object, the **XyDataSpec** object referenced by the first element of the *xyCoordDataSpec* array will provide the resource value for all line labels.

Default: 1.0

xyLineLabelFontQuality

This resource of type NhlTFontQuality determines the quality of the font used to draw **XyPlot** line labels.

Note that there is only a scalar version of this resource. If the **XyPlot** object references more than one **DataItem** object, the **XyDataSpec** object referenced by the first element of the *xyCoordDataSpec* array will provide the resource value for all line labels.

Default: `High`

xyLineLabelConstantSpacingF

Normally when *xyLineLabelFontQuality* is set to `High`, the **XyPlot** object writes line label text with proportional spacing. Setting the *xyLineLabelConstantSpacingF* to a value greater than 0.0 overrides this behavior. Instead, the distance from the start of one character to the next is computed by multiplying a single standard character width by the value of *xyLineLabelConstantSpacingF*. When *xyLineLabelConstantSpacingF* has a value between 0.0 and 1.0, characters will overlap. A value of 1.0 implies, on average, no space between characters, while values increasing from 1.0 cause the space between characters to grow. This parameter is ignored when *xyLineLabelFontQuality* is not `High`. Values less than 0.0 result in an error and are replaced with the default value.

Note that there is only a scalar version of this resource. If the **XyPlot** object references more than one **DataItem** object, the **XyDataSpec** object referenced by the first element of the *xyCoordDataSpec* array will provide the resource value for all line labels.

Default: 0.0

xyLineLabelFuncCode

This resource of type NhlTCharacter sets the function code character that the low level utilities will use when parsing the label string.

Note that there is only a scalar version of this resource. If the **XyPlot** object references more than one **DataItem** object, the **XyDataSpec** object referenced by the first element of the *xyCoordDataSpec* array will provide the resource value for all line labels.

Default: **:**

Obj class resource descriptions

The Obj class has no public resources.

HLU Types Alphabetized

- NhlTAlternatePlace
- NhlTAnnotationDisplayMode
- NhlTAxisType
- NhlTBoolean
- NhlTBooleanGenArray
- NhlTByte
- NhlTByteGenArray
- NhlTcaCastMode
- NhlTCharacter
- NhlTCharacterGenArray
- NhlTcnHighLowLabelOverlapMode
- NhlTcnLevelUseMode
- NhlTcnLevelUseModeGenArray
- NhlTcnLineLabelPlacementMode
- NhlTColorIndex
- NhlTColorIndexGenArray
- NhlTDashIndex
- NhlTDashIndexGenArray
- NhlTDouble
- NhlTDoubleGenArray
- NhlTDrawOrder
- NhlTErrorTypes
- NhlTFillIndex
- NhlTFillIndexGenArray
- NhlTFloat
- NhlTFloatGenArray
- NhlTFont
- NhlTFontQuality
- NhlTGenArray
- NhlTInteger
- NhlTIntegerGenArray
- NhlTJustification
- NhlTlbBoxSizingMode
- NhlTlbLabelAlignmentMode
- NhlTLevelSelectionMode
- NhlTlgItemPlacementMode

- NhlTlgLabelAlignmentMode
- NhlTLineLabelMode
- NhlTLong
- NhlTLongGenArray
- NhlTMapBoundarySets
- NhlTMapDataBaseVersion
- NhlTMapGridMaskMode
- NhlTMapLimitMode
- NhlTMapShapeMode
- NhlTMarkerIndex
- NhlTMarkerIndexGenArray
- NhlTMarkLineMode
- NhlTMarkLineModeGenArray
- NhlTObjId
- NhlTObjIdGenArray
- NhlTOrientation
- NhlTPointer
- NhlTPointerGenArray
- NhlTPosition
- NhlTProjection
- NhlTPSFormat
- NhlTScalingMode
- NhlTShort
- NhlTShortGenArray
- NhlTSpecifiedFillPriority
- NhlTString
- NhlTStringGenArray
- NhlTTextDirection
- NhlTTickMarkMode
- NhlTTickMarkStyle
- NhlTVariable
- NhlTVectorGlyphStyle
- NhlTVectorPositionMode
- NhlTVisualType
- NhlTWorkOrientation
- NhlTXColorMode

HLU Enumeration Values Alphabetized

- NhlABOVEITEMS (1, "AboveItems")
- NhlACROSS (1, "Across")
- NhlADJUSTVP (8,9, "AdjustVP")
- NhlADJUSTVPOMITOVERHL (10,11, "AdjustVPOmitOverHL")
- NhlALLBOUNDARIES (5, "AllBoundaries")
- NhlALLINTEGERS (4, "AllIntegers")

- NhlALWAYS (1, "Conditional")
- NhlANGLES (2, "Angles")
- NhlARROWCENTER (2, "ArrowCenter")
- NhlARROWHEAD (0, "ArrowHead")
- NhlARROWTAIL (1, "ArrowTail")
- NhlAUTOMATIC (0, "Automatic")
- NhlAUTOMATICLEVELS (0, "AutomaticLevels")
- NhlAZIMUTHALEQUIDISTANT (4, "AzimuthalEquidistant")
- NhlBACKGROUND (0)
- NhlBELOWITEMS (2, "BelowItems")
- NhlBOTTOM (1, "Bottom")
- NhlBOTTOMAXIS (4, "BottomAxis")
- NhlBOTTOMCENTER (5, "BottomCenter")
- NhlBOTTOMLEFT (2, "BottomLeft")
- NhlBOTTOMRIGHT (8, "BottomRight")
- NhlBOXCENTERS (0, "BoxCenters")
- NhlCENTER (4, "Center")
- NhlCENTERCENTER (4, "CenterCenter")
- NhlCENTERLEFT (1, "CenterLeft")
- NhlCENTERRIGHT (7, "CenterRight")
- NhlCOLOR (0, "Color"
- NhlCOMPUTED (2, "Computed")
- NhlCONDITIONAL (2, "Always")
- NhlCONFINETORANGE (1, "ConfineToRange")
- NhlCONSTANT (0, "Constant")
- NhlCORNERS (5, "Corners")
- NhlCUSTOM (2, "Custom")
- NhlCYLINDRICALEQUIDISTANT (8, "CylindricalEquidistant")
- NhlDOWN (0, "Down")
- NhlDRAW (1, "Draw")
- NhlEPS (1, "Eps")
- NhlEPSI (2, "Epsi")
- NhlEQUALSPACEDLEVELS (3,"EqualSpacedLevels")
- NhlEXPLICIT (2, "Explicit")
- NhlEXPLICITLEVELS (2, "ExplicitLevels")
- NhlEXPLICITPLACEMENT (1, "ExplicitPlacement")
- NhlEXPLICITSIZING (1, "ExplicitSizing")
- NhlEXTERNALEDGES (2, "ExternalEdges")
- NhlFATAL (-4, "Fatal")
- NhlFILLARROW (1, "FillArrow")
- NhlFIXEDASPECTFITBB (1, "FixedAspectFitBB")
- NhlFIXEDASPECTNOFITBB (2, "FixedAspectNoFitBB")
- NhlFOREGROUND (1)
- NhlFREEASPECT (0, "FreeAspect")
- NhlGEOGRAPHIC (3, "Geographic")
- NhlGEOPHYSICAL (1, "Geophysical")
- NhlGEOPHYSICALANDUSSTATES (4, "GeophysicalAndUSStates")
- NhlGEOPHYSICALPRIORITY (0, "GeophysicalPriority")

- NhlGNOMONIC (3, "Gnomonic")
- NhlHIGH (0, "High")
- NhlHOLLOWFILL (-1)
- NhlHORIZONTAL (0, "Horizontal")
- NhlIGNOREOVERLAP (0,1, "IgnoreOverlap")
- NhlINFO (-2, "Info")
- NhlINTERIOREDGES (1, "InteriorEdges")
- NhlIRREGULAR (2, "Irregular")
- NhlIRREGULARAXIS (0, "IrregularAxis")
- NhlITEMCENTERS (0, "ItemCenters")
- NhlLABELONLY (2, "LabelOnly")
- NhlLAMBERTCONFORMAL (9, "LambertConformal")
- NhlLAMBERTEQUALAREA (2, "LambertEqualArea")
- NhlLANDSCAPE (6, "Landscape")
- NhlLATLON (1, "LatLon")
- NhlLEFT (3, "Left")
- NhlLEFTAXIS (1, "LeftAxis")
- NhlLETTERED (1, "Lettered")
- NhlLINEANDLABEL (3, "LineAndLabel")
- NhlLINEAR (1, "Linear")
- NhlLINEARAXIS (1, "LinearAxis")
- NhlLINEARROW (0, "LineArrow")
- NhlLINEONLY (1, "LineOnly")
- NhlLINES (0, "Lines")
- NhlLOG (0, "Log")
- NhlLOGAXIS (2, "LogAxis")
- NhlLOW (2, "Low")
- NhlMANUAL (1, "Manual")
- NhlMANUALLEVELS (1, "ManualLevels")
- NhlMARKERS (1, "Markers")
- NhlMARKLINES (2, "MarkLines")
- NhlMASKFILLAREA (5, "MaskFillArea")
- NhlMASKLAND (3, "MaskLand")
- NhlMASKMASKAREA (6, "MaskMaskArea")
- NhlMASKNONE (0, "MaskNone")
- NhlMASKNOTLAND (4, "MaskNotLand")
- NhlMASKNOTOCEAN (2, "MaskNotOcean")
- NhlMASKOCEAN (1, "MaskOcean")
- NhlMAXIMALAREA (0, "MaximalArea")
- NhlMAXSIGDIGITSLEFT (3, "MaxSigDigitsLeft")
- NhlMEDIUM (1, "Medium")
- NhlMERCATOR (7, "Mercator")
- NhlMOLLWEIDE (6, "Mollweide")
- NhlMONOCHROME (3, "Monochrome")
- NhlMULTIPLEVECTORS (2, "MultipleVectors")
- NhlNATIONAL (2, "National")
- NhlNCARG4_0 (0, "Ncarg4_0")
- NhlNCARG4_1 (1, "Ncarg4_1")

- NhlNDC (4, "NDC")
- NhlNEVER (0, "Never")
- NhlNOBOUNDARIES (0, "NoBoundaries")
- NhlNOCREATE (-1, "NoCreate")
- NhlNOERROR (-1, "NoError")
- NhlNOLABELS (0, "NoLabels")
- NhlNOLINE (0, "NoLine")
- NhlNONE (0, "None")
- NhlNPC (3, "NPC")
- NhlOMITOVERHL (2,3, "OmitOverHL")
- NhlOMITOVERVP (4,5, "OmitOverVP")
- NhlOMITOVERVPANDHL (6,7, "OmitOverVPAndHL")
- NhlORTHOGRAPHIC (0, "Orthographic")
- NhlPOINTS (6, "Points")
- NhlPOLITICALPRIORITY (1, "PoliticalPriority")
- NhlPORTRAIT (0, "Portrait")
- NhlPOSTDRAW (2, "PostDraw")
- NhlPREDRAW (0, "PreDraw")
- NhlPRIVATE (1, "Private")
- NhlPS (0, "Ps")
- NhlRANDOMIZED (1, "Randomized")
- NhlRIGHT (2, "Right")
- NhlRIGHTAXIS (2, "RightAxis")
- NhlSATELLITE (5, "Satellite")
- NhlSCALEFACTOR (0, "ScaleFactor")
- NhlSHARE (0, "Share")
- NhlSINGLEVECTOR (1,"SingleVector")
- NhlSOLIDFILL (0)
- NhlSOLIDLINE (0)
- NhlSPLITVECTORS (3,"SplitVectors")
- NhlSTEREOGRAPHIC (1, "Stereographic")
- NhlTIME (4, "Time")
- NhlTOP (0, "Top")
- NhlTOPAXIS (3, "TopAxis")
- NhlTOPCENTER (3, "TopCenter")
- NhlTOPLEFT (0, "topLeft")
- NhlTOPRIGHT (6, "TopRight")
- NhlTRANSPARENT (-1)
- NhlTRIMZEROS (2, "TrimZeros")
- NhlUNIFORMPLACEMENT (0, "UniformPlacement")
- NhlUNIFORMSIZING (0, "UniformSizing")
- NhlUSSTATES (3, "USStates")
- NhlVERTICAL (1, "Vertical")
- NhlWARNING (-3, "Warning")
- NhlWINDBARB (0, "WindBarb")
- NhlWINDOW (7, "Window")

HLU class page format

This module describes the layout of the reference page for the HLU classes.

Class name

This section provides a brief overview about the class.

Synopsis

This section describes the class's header files, class name, class pointer, Fortran class function, superclass, and composite class.

Header file

C developers should include this file in applications that use this class.

Class name

This is the name of the class described in the current module. The class name can be used in a resource file to specify a default for all objects of that class.

Class pointer

This tag is used in C programs where a class identifier is needed. For example, the NhlCreate call requires a class pointer to identify the type of object to create.

Fortran class function

This is the Fortran external function name. This tag is also used in the Fortran NhlFCreate call to identify the type of object to create. This tag must be declared as "EXTERNAL" at the beginning of a Fortran program.

Superclass

This identifies the class "above" the class in the class hierarchy chart. All resources defined in a superclass are available to the current class.

Composite class

This identifies the composite classes that help define the functionality of the current class. A subset of the resources defined in a composite class are available to the current class.

Class-defined types

This section lists the relevant C types for the current class. Note that for enumerated type definitions, the enumerator, an integer value, and a C comment are listed in the definition section. These values can be used in different functions for setting resources. For example, Resources that have an enumerated type can be set using the SetInteger function with the enumerated value or the integer value, or using the SetString function with the string value. Note that the set of characters that are enclosed in quotes in the C comment field is the string representation of the enumerated value. This string value can be used in the SetString function, and the string value is case insensitive.

For more information, see the discussion of enumerated types in the Resource types module.

Resources

This section lists the local, composite, and superclass resources for the class.

Local Resources

This section provides a table that lists the local resource names and their class, type, default value, and access method.

- Name - This is the resource name that is used in resource files as well as function calls to control characteristics of an HLU object. Clicking on the resource name will link you to a complete description of the resource and its valid values.
- Class - The class table entry specifies the name of the class that the resource belongs to. The class name can be used in resource files to set all the resources that belong to a particular class. For example, the text item resource, txFont, has a class name of "Font". This class name could be used in a resource file to set all resources in the Font class to a single type, such as helvetica.

  ```
  *.Font   helvetica
  ```

- Type - The type lists the HLU type of the resource. The type entries are linked to their type definitions.
- Default - The default column lists the default value for the resource.
- Access - The access column of the table specifies how the resource can be accessed.
  - R - specifies that the resource can be used in a resource file.
  - C - specifies that the resource can be set at create time with the NhlCreate (C) or NhlFCreate (Fortran) calls.
  - S - specifies that the resource can be set programmatically using the NhlSetValues (C) or NhlFSetValues (Fortran) calls.
  - G - specifies that the resource can be read programmatically using the NhlGetValues (C) or NhlFGetValues (Fortran) calls.

Composite Resources

This section provides links to the classes that provide the composite resources. Also, descriptions of any special limitations the class places on the use of the composite class resources is included.

Superclass resources

This section provides links to the resources of each of the superclasses.

Description

This section gives a detailed overview of how to use the class, and it explains the functionality that is available.

Support functions

This section lists the HLU functions that are defined for the class.

Status

This section lists the current status of the class implementation.

See also

This section lists related topics.

Copyright

This is the copyright notice.

Format conversion specification strings

Description

The HLU library provides facilities for creating formatted strings representing numerical data items using a format conversion specifier that is modeled after and extends C language print format specifiers. It supports very detailed control of the appearance of numerical output, including access to typographical features such as superscripting for exponents. Specifying formatting requirements in this fashion is somewhat tedious and is seen as an interim solution for the HLU library. Eventually, a **Format** object class may be created that will allow you to specify format conversions using resources tied to each formatting option. However, the string conversion specification does have the virtue of compactness and will continue to be supported. Currently, the HLU library only supports formatting for floating point values. However, reasonable output can be generated for integers as well if they are first converted to an equivalent floating point value. Both the **ContourPlot** and **TickMark** classes use the HLU format conversion facilities to control the appearance of their numerical labels. As yet, there are no public entry points allowing the user to directly specify format conversions.

Note that a number of the format options relate to an output field width defined in terms of a character count within which the converted number is to fit. These options do not have well-defined behavior for variable-width (proportionally spaced) fonts. Also, as in standard C (and unlike Fortran), the conversion is printed even if it contains more characters than the number specified for the field width.

Within a string, a C-language conversion specifier is signaled by the presence of an '%' character. In the HLU library, you supply conversion specifier strings as the value of string resources specifying the format for some set of numbers that will appear in the output plot. In this context, a leading signal character is unnecessary and is not understood. When HLU conversion specifiers are recognized within strings, they will be signaled--as in C--using the the '%' character.

After the signal character, C conversion specifiers consist of a number of 'flag' characters followed by one or two conversion 'fields': for floating point numbers these specify the output field width and (depending on what follows) either the number of significant digits or the number of digits following the decimal point. A C conversion specifier string ends with a one-character conversion specifier, possibly preceded by a one-character conversion qualifier. HLU conversion specifiers slightly modify and significantly extend this general model. The HLU format specification eliminates the alternate interpretation of the second C conversion field, but uses additional flag characters to signal several more conversion fields. As in C, the conversion string ends with a one-character conversion specifier, although currently the HLU library recognizes fewer conversion specifiers than C and does not support any conversion qualifier characters.

The default approach of the HLU library, when only a minimal format conversion string is specified, is to output the value using the least number of characters possible. It must be noted that although the HLU format specification is in general a superset of corresponding features from the C language standard, it does not claim to follow the C standard in every detail.

Flag characters

In general, the HLU library tries to give the format flag characters used by C the same basic meaning as C does. This table of format flag characters in the HLU library gives their conventional names and meaning:

Format flag characters

CHARACTER	NAME and EFFECT
+	Plus: generate a leading plus sign for positive values.
-	Minus: left-justify the conversion within the field.
#	Pound: force a trailing decimal point.
0	Zero: first, fill with zeros to account for all significant digits to the right of the decimal point; then, if field width is greater than the current string length, fill to field width, left or right depending on the justification.
<space>	Space: generate a leading space for positive values if the '+' flag is not also present.
!	Exclamation: If exponential format is in

	effect force mantissas of 1 to print.
,	Comma: use comma (',') as the decimal point instead of dot ('.').
@	At-sign: force initial or trailing zero where a decimal point would otherwise be the first or last character.
&	Ampersand: the character following this flag becomes the fill character used instead of spaces to fill in otherwise unoccupied places within the width of the field.

Note that the way HLU handles filling of the output field using zeros ('0') is different from the way it fills using arbitrary characters specified with the Ampersand flag. When zero is the fill character, sign characters ('+' or '-' or <space>) precede any leading zeros. When any other fill character is used, the sign characters follow any leading instances of the fill character.

Conversion fields

The use of the term *field* could be somewhat confusing in this context. What is meant here are certain options specified using numbers within the format conversion string itself. Some of these fields specify attributes of the "field" within which the converted number is printed in the output plot. Wherever the context does not make it clear, the phrases "conversion field" and "output field" will be used to differentiate between the two uses of the word.

As with C-language format conversion specifiers, any digits immediately following the recognized flag characters (other than an initial zero ('0'), since it is a flag character) specify the width in characters of the output field. All other conversion fields are preceded by a special character signaling the conversion field type, and may appear in any order following any flags and/or field width specification digits. The digits making up each conversion field are interpreted as a single integer read from left to right. Note that for all conversion fields, an asterisk ('*') character may take the place of the digits, and implies that the field value is to be determined dynamically. The code internal to the HLU object that calls the actual formatting routine would then be allowed to supply a value for the conversion field based on its own criteria. If it chooses not to supply a value for any dynamic conversion field, the formatting routine will act as if the field had not been specified at all. As a special shortcut, you can give *all* otherwise unspecified conversion fields the dynamic attribute by either replacing or following the the output field width conversion field with the characters '*+'. Some of the conversion fields have default values; others are ignored when left unspecified. Here is a table of the conversion fields:

Conversion fields	
SIGNAL CHAR	NAME and MEANING
<none>	Field width: specifies the output field width in characters. By default, the output field width is equal to the width required to represent the string. The field width conversion field must directly follow any format conversion flags.

.	Significant digits: specifies the number of significant digits used to express the output value. Note that for the 'f', 'e', and 'E' conversion specifiers this is a departure from the C standard where the '.' field specifies the number of digits to the right of the decimal point. Use the leftmost significant digit field (';') in conjunction with the significant digit field to specify a constant number of fraction digits. Remember that zeroes to the right of the rightmost non-zero digit will not be included in the output string unless the zero flag character ('0') is specified. Default: 6
;	Leftmost significant digit: specifies the digit position (in a decimal system) that is to be considered the leftmost significant digit. The units position (the digit directly to the left of the decimal point) is counted as digit position 0, and positions are counted positively towards the left. This conversion field allows you to format a group of numbers in a uniform fashion, usually based on the value of the largest number in the group. To achieve this, you would normally set the field using the dynamic specifier--the asterisk character ('*')--so that the code invoking the formatting routine can specify this conversion field based on its knowledge of each number's value relative to the group of numbers with which it is associated. If left unspecified, the leftmost significant digit is the digit position of the number's actual most significant digit.
^	Exponent field specifier: this conversion field is unusual in that one or two exponent flag characters may precede the field value. Directly after the signal character, either the letter 'a' or the letter 's' may appear. Next may appear the character '+'. If no letter appears, the exponent field is written in the form 'exxx...' or 'Exxx...', depending, as in the C language, on the case of the final conversion specifier character. The letter 'a' means that the exponent is written in 'asterisk' format: 'x10**xxx...'. The letter 's' causes it to be written as a superscript using low-level library function codes. This format cannot be rendered in ascii, but looks approximately like 'x10^^^'. The plus character indicates that positive exponent values should be preceded by a '+' sign. The numerical value specifies the number of digits to use in the exponent. If unspecified, the exponent will be expressed using the fewest digits possible.
?	Exponent switch length: When the conversion specifier is 'g' or 'G', this conversion

	field specifies that the non-exponential format should be used as long as it requires no more characters than the field's value to express. Otherwise, the format using the fewest characters should be used. Default: 5
~	Point Position: If an output field width is specified this conversion field specifies the position, counting from the left edge of output field, where the decimal point is to appear. If it is not possible to place the decimal point in the requested position, it will be placed as near to it as possible. Note that use of this conversion field overrides any justification flags. As long as a fixed font is used, this field allows you to align vertical columns of numbers based on the decimal point position.

Conversion specifiers

Following the conversion fields, an HLU conversion string ends with a one-character conversion string. Five characters are currently recognized:

Conversion specifiers

CHARACTER	NAME and MEANING
f	Floating point format: the number is expressed in non-exponential format regardless of its size.
e	Exponential format, lowercase: the number is expressed using a exponential format. If the letters 'a' or 's' do not appear as part of the exponential conversion field, the exponent is preceded by the character 'e' (lowercase).
E	Exponential format, uppercase: the number is expressed using a exponential format. If the letters 'a' or 's' do not appear as part of the exponential conversion field, the exponent is preceded by the character 'E' (uppercase).
g	Generic format, lowercase: if the number can be expressed in a non-exponential format using fewer digits than the value specified by the exponent switch length conversion field (default: 5), then the non-exponential format is used. Otherwise the format using the fewest characters is used. If the exponential format is used and the letters 'a' or 's' do not appear as part of the exponential conversion field, the exponent is preceded by the character 'e' (lowercase).
G	Generic format, uppercase: if the number can be expressed in a non-exponential format using

```
|                |  fewer digits than the value specified by the    |
|                |  exponent switch length conversion field         |
|                |  (default: 5), then the non-exponential          |
|                |  format is used. Otherwise the format using      |
|                |  the fewest characters is used. If the           |
|                |  exponential format is used and the              |
|                |  letters 'a' or 's' do not appear as part of     |
|                |  the exponential conversion field, the exponent  |
|                |  is preceded by the character 'E' (uppercase).   |
+----------------------------------------------------------------+
```

Text function codes

Description

This page describes the syntax and capabilities of function codes that you can embed within a TextItem string. It is a slightly modified version of the description of function codes in the Programmer's Document for the low-level NCAR Graphics utility Plotchar. You will need to understand function codes in order to modify the TextItem object string for any of the following purposes:

- To select characters from fonts other than the "normal" font that is set using the resource txFont.
- To select the "level" on which characters are to be written: superscript level, subscript level, or normal level. Subscripting and superscripting may be carried to as many as five levels, as in "A to the B to the C to the D to the E to the F". Changing level usually also changes font, case, and size (if using the pwritx_database font) or just size (if using any of the other fonts), but all such changes may be overridden by function codes forcing the desired values.)
- To take complete control of the positioning of characters relative to one another, including the creation of multi-line TextItem object strings through the use of embedded "carriage returns".
- To specify individual characters from the pwritx_database font, using octal numbers as function codes.
- To "zoom" characters in either width or height, or both.

How the text scanner operates

As the characters of txString are scanned from left to right, the scanner is always in one of two states: either it is looking for characters to be drawn, or it is looking for characters to be interpreted as function codes. As the scan begins, the scanner is looking for characters to be drawn. Each occurrence of the function-code signal character (specified using the txFuncCode resource--a colon by default) flips the state of the scanner. Thus, if txString were set to "`ABC:L:DEF`", 'A', 'B', and 'C' will be treated as characters to be drawn, 'L' will be treated as a function-code character, and 'D', 'E', and 'F' will be treated as characters to be drawn. (In this particular example, since the function code 'L' requests lower case, what would be drawn is the string "ABCdef".)

An example of a string containing function codes is as follows:

> :PRU:A:S:2:N:+B:S:2:N:

This might be read by a knowledgeable user as "Principal Roman Upper A, Superscript level, 2, Normal level, plus B, Superscript level, 2, Normal level" and it writes the equivalent of "A squared plus B squared". Because of the way the colons are paired, the characters 'P', 'R', 'U', 'S', 'N', 'S', and 'N' are function codes; all other characters are characters to be written.

No punctuation is needed between function codes except for a comma or a blank between adjacent numbers; however, extra commas and blanks may be used to improve readability. The function codes described below are the only legal ones; any other characters in a function-code string will be ignored, but an error message will be printed.

Font selection

Use these function codes to change the font:

- F or F0 or F*n*, where *n* is one of the legal font numbers.

The default at the beginning of a string is to use the font implied by the resource txFont. (By default, txFont is set to zero, selecting the pwritx_database font.) The function code 'F', without a following integer, implies a return to the font specified by the txFont resource.

Font definitions

These function codes affect the selection of characters from the pwritx_database font:

- R selects the Roman font.
- G selects the Greek font.

The default at the beginning of a string is to use the Roman font.

Size definitions

These function codes affect the selection of character size and style when pwritx_database font is used and only the size when any other font is used.

- P selects characters of principal size.
- I selects characters of indexical size.
- K selects characters of cartographic size.

The default at the beginning of a string is to use characters of principal size. When you use the pwritx_database font, these function codes affect the style of the character as well as its size. That is because the pwritx_database font is in reality a concatenation of several different fonts. However, when other fonts are used, these codes actually do just affect the size of the character. For example, if txString is "`:F16K:ABC`", the characters drawn will be those defined by font 16 (gothic_english) corresponding to the uppercase characters 'A','B', and 'C', but they will be reduced in size to approximately match the size of the cartographic characters from the pwritx_database font.

Case definitions

These function codes affect the selection of characters from the pwritx_database font. (An "L" function code will also affect the selection of characters from the other fonts. For example, if txString is set to "`:F16L:ABC`", the characters drawn will be those defined by font 16 (gothic_english) corresponding to the lowercase characters 'a', 'b', and 'c'.)

- U or U*n* selects uppercase.
- L or L*n* selects lowercase.

If U is followed by a number *n* (with no intervening comma), then *n* characters will be drawn in uppercase, and subsequent characters will be in lowercase. (The U*n* option is particularly useful for capitalizing sentences.)

If L is followed by a number *n*, then *n* characters will be drawn in lowercase, and subsequent characters will be in uppercase.

The default at the beginning of a string is uppercase.

Direction definitions

These function codes determine where each character is placed relative to the previous one:

- A causes characters to be written "across" the frame.
- D or D*n* causes characters to be written "down" the frame.

The default at the beginning of a string is to write across the frame.

If D appears without a following integer *n* or if *n* is zero, characters will be written "down" until an 'A' is encountered. If *n* appears, *n* characters will be written "down" and subsequent characters will be written "across" the frame.

The terms "across" and "down" are to be interpreted to mean "in the direction *txAngleF*" and "in the direction *txAngleF* - 90", respectively. Each character has a left edge center point, a right edge center point, a bottom edge center point, and a top edge center point, as defined by information in the database of the font from which it came. These points are not necessarily on the actual edges of the character, but on the edges of a rectangle containing the character; the points are also not necessarily exactly centered on the edges of the rectangle. Characters are always oriented so that the vector from the left edge center point to the right edge center point is in the direction *txAngleF*. When characters are written across the frame (assuming that the constant spacing parameter txConstantSpacingF is zero), each character is placed so that its left edge center point is coincident with the right edge center point of the previous character. Similarly, when characters are being written down the frame, each character is placed so that its top edge center point is coincident with the bottom edge center point of the previous character.

Level definitions

These function codes determine the level on which characters are to be written:

- B or B*n* selects subscript level.
- S or S*n* selects superscript level.

- E means "end of subscripting or superscripting".
- N causes a return to normal level.

If S or B is followed by a positive integer *n*, then *n* characters will be drawn as specified above, after which size, case and position will be reset to that of the base character. Do not overlap level definitions given for a specified number of characters (as, for example, in the string "`:S3:A:B2:BC`", in which the three-character count for superscripting is not satisfied at the time the two-character count for subscripting is seen).

The function code E terminates the current level of subscripting or superscripting, restoring the size, case, and position to that of the base character. The function code N terminates the current level of subscripting or superscripting, restoring the size and case to that of the base character, but the position as required to continue writing at the level of the base character without writing over the subscripts or superscripts.

Examples:

- To write "10 to the 10 to the 100", use the string "`10:S:10:S:100`". Up to five levels are allowed.
- To write "X sub 2, cubed", use the string "`X:B1:2:S1:3`". Note that when the character count in the "B1" is satisfied, the original base character is reinstated, so that we can add another subscript or superscript to it.
- To write "X sub 2, cubed, times Y cubed", use the string "`X:B1:2:S:3:N:Y:S:3`". The function code N returns to normal level in such a way that the character Y does not overwrite the 2 and the 3.
- To write "X to the power A sub 1", followed by "something else", use the string "`X:S:A:B:1:NN:something else`". Note that two function code N’s are used, one to terminate writing the subscript 1 and another to terminate writing the superscript "a sub 1" and leave the current position such that what follows won’t go on top of the "a sub 1".

Coordinate definitions

These function codes adjust the current position, so that the next character will be drawn offset from the position it would otherwise have had:

- H*n* or H*n*Q causes a "horizontal" move.
- V*n* or V*n*Q causes a "vertical" move.
- C causes a "carriage return" before the next character is drawn.

An H*n* or H*n*Q increments the current position in the direction specified by *txAngleF*. H*n* shifts the position by *n* digitization units and H*n*Q shifts the position by *n* blank widths; in either case, *n* can be negative.

A V*n* or V*n*Q increments the current position in the direction specified by *txAngleF* + 90. V*n* shifts the position by *n* digitization units and V*n*Q shifts the position by *n* blank heights; in either case, *n* can be negative.

A C causes a return to either the initial position, if there have been no other C function codes, or to the position following the last occurrence of a C function code, otherwise. The position is then offset by one blank height in the direction *txAngleF* -90. For example, the string "`ABC:C:DEF`" causes the characters ABC to be written and ten causes the characters DEF to be written directly under the ABC.

Character zooming

These function codes may be used to alter the aspect ratio of subsequent characters:

- X*n* or X*n*Q causes character width to zoom to "*n*" percent of normal.
- Y*n* or Y*n*Q causes character height to zoom to "*n*" percent of normal.
- Z*n* or Z*n*Q causes character width and height to zoom to "*n*" percent of normal.

Using X*n* or X*n*Q will cause subsequent characters to be made *n*/100 times as wide (zoomed in the direction *txAngleF*). A suffixed Q is ignored. If "*n*" is omitted or zero, 100 is assumed.

Using Y*n* or Y*n*Q will cause subsequent characters to be made *n*/100 times as high (zoomed in the direction *txAngleF* + 90). A suffixed Q will cause a shift of character position in the direction *txAngleF* + 90 sufficient to keep the bases of the characters properly aligned. If "*n*" is omitted or zero, 100 is assumed.

Using Z*n* or Z*n*Q will cause subsequent characters to be made *n*/100 times as wide and high (zoomed in both of the directions *txAngleF* and *txAngleF* + 90). A suffixed Q will cause a shift of character position in the direction *txAngleF* + 90 sufficient to keep the bases of the characters properly aligned. If "*n*" is omitted or zero, 100 is assumed.

Direct character access

Octal numbers may be used as function codes to select particular characters from the pwritx_database font. The octal number for a given character is the sum of a font index (0 for Roman or 600 for Greek), a size index (0 for Principal, 200 for Indexical, or 400 for Cartographic), a case index (0 for Upper or 100 for Lower), and the octal equivalent of a character index (1-32 for A through Z, 33-44 for 0 through 9, or 45-57 for the individual characters +, -, *, /, (,), $, =, blank, comma, and period). For example, the string "`:GIU:*`" is the same as the string "`:1047:`" (600+200+0+47=1047, octal.) To find the character produced for a given octal code, refer to the plots produced by the low-level NCAR Graphics example "epltch".

Note: In the "epltch" example (see previous paragraph), you can write two of the characters in a row by using either the string ':GIU:**' or the string "`:1047,1047:`". This is one example of the need for a comma between two function codes.

An octal number may be used as a function code in part of the input character string during the processing of which characters are being selected from fonts other than the pwritx_database font. Such an octal function code will still have the effect of selecting a character from the pwritx_database font.

Resource database

Overview

A program often needs a variety of options in the HLU environment (for example, fonts, colors, and dash patterns). Specifying all of these options on the command line is awkward because users may want to customize many aspects of the program and need a convenient way to establish these customizations as the default setting. The **resource database** is provided for this purpose. Resource specifications are stored in human-readable files.

The **resource database** is different than most database systems. In most database systems, you perform a query using an imprecise specification, and you get back a set of records. The **resource database**, however, allows you to specify a large set of values with an imprecise specification, to query the database with a precise specification, and to get back only a single value. This should be used by applications that need to know what the user prefers for colors, fonts, and other resources.

As an example of how the **resource database** works, consider the ap01c example program that displays a **TextItem** object. The HLU library is designed so that programs use a hierarchy of objects all the way down to each individual object. This is called the instance hierarchy. Each of these objects can be assigned a name, and is a member of a class, and therefore has a class name. The *instance hierarchy* along with the name and class of each object can be used to generate a fully qualified name and class for each object. Fully qualified names or classes can have arbitrary numbers of component names, but a fully qualified name always has the same number of component names as a fully qualified class.

For example, the ap01c program has a top-level **App** object named "ap01", which has a class name of "appClass". Each name and class finally has an attribute (for example "txFont" or "Font"). By convention, the first character of name attributes are in lowercase, and the first character of class attributes are capitalized. If each object is properly assigned a name and class, it is easy for the user to specify attributes of any portion of the application.

Example ap01c creates an **XWorkstation** object child of the **App** object with a name of "x" and a class name of "xWorkstationClass". ap01c creates a **TextItem** object child of "x" with a name of "tx1" and a class name of "textItemClass". This object has a fully qualified name, "ap01.x.tx1", and a fully qualified class, "appClass.xWorkstationClass.textItemClass". Its fully qualified name is the fully qualified name of its parent, "ap01.x", followed by its name, "tx1". Its fully qualified class name, is the fully qualified class name of its parent, "appClass.xWorkstationClass", followed by its particular class name, "textItemClass". The fully qualified name of a resource is the attribute’s name appended to the object’s fully qualified name, and the fully qualified class is its class appended to the object’s class.

The **TextItem** object needs the following resources: String, Font, and Color. Each resource is an attribute of the **TextItem** object and, as such, has a name and a class. For example, the String attribute is name *txString*, and its class is *TxString*.

When an HLU object looks up a resource (for example, a color), it passes the complete name and complete class of the resource to a lookup routine. The **resource database** compares this complete specification against the incomplete specification of entries in the database, finds the best match, and returns the corresponding value for that entry.

Resource file syntax

The syntax of a resource file is a sequence of resource lines terminated by newline characters or end-of-file. The syntax of an individual resource line is:

```
ResourceLine     = Comment|IncludeFile|ResourceSpec|<empty line>
Comment          = "!" {<any character except null or newline>}
IncludeFile      = "#" WhiteSpace "include" WhiteSpace FileName WhiteSpace
FileName         = <valid filename for operating system>
ResourceSpec     = WhiteSpace ResourceName WhiteSpace ":" WhiteSpace Value
ResourceName     = [Binding] {Component Binding} ComponentName
Binding          = "."|"*"
WhiteSpace       = {<space>|<horizontal tab>}
Component        = "?"|ComponentName
ComponentName    = NameChar {NameChar}
NameChar         = "a"-"z"|"A"-"Z"|"0"-"9"|"_"|"-"
Value            = {<any character except null or unescaped newline>}
```

Elements separated by vertical bar (|) are alternatives. Curly braces ({...}) indicate zero or more repetitions of the enclosed elements. Square brackets ([...]) indicate that the enclosed element is optional. Quotes ("...") are used around literal characters.

IncludeFile lines are interpreted by replacing the line with the contents of the specified file. The word "include" must be in lowercase. The filename is interpreted relative to the directory of the file in which the line occurs (for example, if the filename contains no directory or contains a relative directory specification).

If a **ResourceName** contains a contiguous sequence of two or more **Binding** characters, the sequence will be replaced with a single "." character if the sequence contains only "." characters, otherwise the sequence will be replaced with a single "*" character.

A **resource database** never contains more than one entry for a given **ResourceName**. If a resource file contains multiple lines with the same **ResourceName**, the last line in the file is used.

Any whitespace characters before or after the name or colon in a **ResourceSpec** are ignored. To allow a **Value** to begin with whitespace, the two-character sequence "\space" (backslash followed by space) is recognized and replaced by a space character, and the two-character sequence "\tab" (backslash followed by horizontal tab) is recognized and replaced by a horizontal tab character. To allow a **Value** to contain embedded newline characters, the two-character sequence "\n" is recognized and replaced by a newline character. To allow a **Value** to contain arbitrary character codes, the four-character sequence "\xxx", where each *x* is a digit character in the range of "0" to "7", is recognized and replaced with a single byte that contains the octal value specified by the sequence. Finally, the two-character sequence "\\" is recognized and replaced with a single backslash.

As an example of these sequences, the following resource line contains a value consisting of four characters: a backslash, a null, a "z", and a newline:

```
magic.value:    \\\000\
z\n
```

Resource matching rules

The algorithm for determining which resource database entry matches a given query is the heart of the **resource database**. All queries must fully specify the name and class of the desired resource (use of "*" and "?" are not permitted). The library supports up to 100 components in a full name or class. Resources are stored in the database with only partially specified names and classes, using pattern matching constructs. An asterisk (*) is a loose binding and is used to represent any number of intervening components, including none. A period (.) is a tight binding and is used to separate immediately adjacent components. A question mark (?) is used to match any single component name or class. A database entry cannot end in a loose binding; the final component (which cannot be "?") must be specified. The lookup algorithm searches the database for the entry that most closely matches (is most specific for) the full name and class being queried. When more than one database entry matches the full name and class, precedence rules are used to select just one.

The full name and class are scanned from left to right (from highest level in the hierarchy to lowest), one component at a time. At each level, the corresponding component and/or binding of each matching entry is determined, and these matching components and bindings are compared according to precedence rules. Each of the rules is applied at each level, before moving to the next level, until a rule selects a single entry over all others. The rules (in order of precedence) are:

1. An entry that contains a matching component (whether name, class, or "?") takes precedence over entries that elide the level (that is, entries that match the level in a loose binding).
2. An entry with a matching name takes precedence over both entries with a matching class and entries that match using "?". An entry with a matching class takes precedence over entries that match using "?".
3. An entry preceded by a tight binding takes precedence over entries preceded by a loose binding.

To illustrate these rules, consider following the resource database entries for example **ap01**:

```
ap01*xWorkstationClass*txFont:  default         (entry A)
*tx1.Font:                      helvetica       (entry B)
ap01.x*textItemClass*txFont:    times-bold      (entry C)
ap01.x*?.Font:                  courier         (entry D)
ap01.x*textItemClass.txFont:    courier-bold    (entry E)
```

When the HLU library tries to determine a default for the *Font* to use for "tx1", it will query for the resource:

```
ap01.x.tx1.txFont                                   (name)
appClass.xWorkstationClass.textItemClass.Font       (class)
```

At the first level (ap01, appClass), rule 1 eliminates entry **B**. At the second level (x, xWorkstationClass), rule 2 eliminates entry **A**. At the third level (tx1, textItemClass), rule 2 eliminates entry **D**. At the fourth level (txFont, Font), rule 3 eliminates entry **C**. Therefore, "tx1" should display its string using a Font of courier-bold.

Copyright

Copyright 1987-1999 University Corporation for Atmospheric Research
The use of this Software is governed by a License Agreement.

NCAR Graphics is a registered trademark of the University Corporation for Atmospheric Research.

Note: Most of the **resource database** software was originally taken from the MIT X11R5 distribution. Also, most of the documentation in this file is taken from the MIT X11R5 distribution. These were modified as needed for use in the HLU library. This is with the permission granted from the following:

Documentation

```
The X Window System is a trademark of MIT.
TekHVC is a trademark of Tektronix, Inc.
Copyright 1985,1986,1987,1988,1989,1990,1991 by Massachusetts Institute of
Technology, Cambridge, Massachusetts, and Digital Equipment Corporation,
Maynard, Massachusetts.
Portions Copyright 1990,1991 by Tektronix, Inc.
Permission to use, copy, modify and distribute this documentation for any
purpose and without fee is hereby granted, provided that the above copyright
notice appears in all copies and that both that copyright notice and this
permission notice appear in all copies, and that the names of MIT, Digital, and
Tektronix not be used in advertising or publicity pertaining to this
documentation without specific, written prior permission. MIT, Digital, and
Tektronix makes no representations about the suitability of this documentation
for any purpose. It is provided "as is" without express or implied warranty.
```

Code

```
Copyright 1987, 1988, 1990 by Digital Equipment Corporation, Maynard,
Massachusetts, and the Massachusetts Institute of Technology, Cambridge,
Massachusetts.

                        All Rights Reserved

Permission to use, copy, modify, and distribute this software and its
documentation for any purpose and without fee is hereby granted, provided that
the above copyright notice appears in all copies and that both that copyright
notice and this permission notice appear in supporting documentation, and that
the names of Digital or MIT not be used in advertising or publicity pertaining
to distribution of the software without specific, written prior permission.

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

MapPlot Ncarg4_0 database table

This table lists the contents of the **MapPlot** Ncarg4_0 database. The last column contains the area names, accessible through the **MapPlot** class resource *mpAreaNames*. These names are considered the primary key to the data. Their order in the database is given by the first column of the table, labelled "ELEMENT NUMBER."

The second column contains an integer representation of the area type. You can retrieve an array of these values by getting the read-only array resource, *mpAreaTypes*.

The third column contains the fixed area group assigned to each area. This value is used to determine the "geophysical" fill coloring of each area. Since it is based only on whether an area is ocean, land, or inland water, it cannot be modified by the user. You can access these values by getting the array resource *mpFixedAreaGroups*.

The fourth column contains the dynamic area group assigned to each area. This group is used to determine the "political" fill coloring assigned to each area in the database. By default, it is set up such that distinct adjoining political areas are guaranteed to be a different color, but the user can reset these values for other purposes, such as to represent statistical data. You can access these values through the array resource, *mpDynamicAreaGroups*.

ELEMENT NUMBER	TYPE	FIXED GROUP	DYNAM. GROUP	AREA NAME
1	0	1	1	Oceans
2	1	2	4	Africa-Europe-and-Asia
3	1	2	9	Antarctica
4	1	2	5	Australia
5	1	2	4	North-and-South-America
6	2	2	4	Borneo
7	2	2	8	Canada-Baffin-Island
8	2	2	8	Canada-Ellesmere-Island
9	2	2	8	Canada-Victoria-Island
10	2	2	6	Cuba
11	2	2	4	Greenland
12	2	2	7	Indonesia-Sulawesi-(Celebes)
13	2	2	7	Indonesia-Sumatra
14	2	2	4	Ireland
15	2	2	5	Japan-Hokkaido
16	2	2	5	Japan-Honshu
17	2	2	5	Japan-Kyushu
18	2	2	5	Japan-Shikoku
19	2	2	5	Madagascar
20	2	2	4	New-Guinea
21	2	2	8	New-Zealand-North-Island_1
22	2	2	8	New-Zealand-North-Island_2
23	2	2	8	New-Zealand-South-Island
24	2	2	4	England-Scotland-and-Wales
25	2	2	4	Tierra-del-Fuego
26	3	2	9	Alaska-Atka-Island
27	3	2	9	Alaska-Attu-Island
28	3	2	9	Alaska-Kodiak-Island

29	3	2	9	Alaska-Nunivak-Island
30	3	2	9	Alaska-Saint-Lawrence-Island
31	3	2	9	Alaska-Umnak-Island
32	3	2	9	Alaska-Unalaska-Island
33	3	2	9	Antarctica-Alexander-Island
34	3	2	9	Antarctica-Anvers-Island
35	3	2	9	Antarctica-Bear-Island
36	3	2	9	Antarctica-Brabant-Island
37	3	2	9	Antarctica-Charcot-Island
38	3	2	9	Antarctica-Coronation-Island
39	3	2	9	Antarctica-Guest-Island
40	3	2	9	Antarctica-Joinvile-Island
41	3	2	9	Antarctica-King-George-Island
42	3	2	9	Antarctica-Livingston-Island
43	3	2	9	Antarctica-Roosevelt-Island
44	3	2	9	Antarctica-Ross-Island
45	3	2	5	Australia-Kangaroo-Island
46	3	2	5	Australia-Melville-and-Bathurst-Island
47	3	2	5	Australia-Tasmania
48	3	2	8	Bahamas_1
49	3	2	8	Bahamas_2
50	3	2	8	Bahamas_3
51	3	2	5	Bismarck-Archipelago-New-Britain
52	3	2	5	Bismarck-Archipelago-New-Hanover
53	3	2	5	Bismarck-Archipelago-New-Ireland
54	3	2	8	Canada-Akimiski-Island
55	3	2	8	Canada-Anticosti-Island
56	3	2	8	Canada-Axel-Heiberg-Island
57	3	2	8	Canada-Banks-Island
58	3	2	8	Canada-Bathurst-Island
59	3	2	8	Canada-Belcher-Island
60	3	2	8	Canada-Borden-Island
61	3	2	8	Canada-Brock-Island
62	3	2	8	Canada-Byam-Martin-Island
63	3	2	8	Canada-Bylot-Island
64	3	2	8	Canada-Coats-Island
65	3	2	8	Canada-Cornwall-Island
66	3	2	8	Canada-Cornwallis-Island
67	3	2	8	Canada-Devon-Island
68	3	2	8	Canada-Eglinton-Island
69	3	2	8	Canada-Ellef-Ringnes-and-Amund-Ringnes-Island
70	3	2	8	Canada-Lougheed-Island
71	3	2	8	Canada-Mackenzie-King-Island
72	3	2	8	Canada-Manitoulin-Island
73	3	2	8	Canada-Mansel-Island
74	3	2	8	Canada-Meighen-Island
75	3	2	8	Canada-Melville-Island
76	3	2	8	Canada-Newfoundland-Island
77	3	2	8	Canada-Prince-Charles-Island
78	3	2	8	Canada-Prince-Edward-Island
79	3	2	8	Canada-Prince-Patrick-Island
80	3	2	8	Canada-Prince-of-Wales-Island
81	3	2	8	Canada-Queen-Charlotte-Island
82	3	2	8	Canada-Southampton-Island
83	3	2	8	Canada-Vancouver-Island
84	3	2	4	Canary-Island_1
85	3	2	4	Canary-Island_2
86	3	2	4	Canary-Island_3
87	3	2	4	Canary-Island_4
88	3	2	4	Chile-Isla-de-Chiloe
89	3	2	7	China-Hainan-Dao-Island

90	3	2	5	Corsica
91	3	2	4	Crete
92	3	2	6	Cuba-Isla-de-Pinos
93	3	2	5	Curacao
94	3	2	5	Cyprus
95	3	2	4	Denmark-Fyn
96	3	2	4	Denmark-Lolland
97	3	2	4	Denmark-Sjaelland
98	3	2	4	Faeroe-Island
99	3	2	7	Faulkland-Island
100	3	2	5	Fiji-Vanua-Levu
101	3	2	5	Fiji-Viti-Levu
102	3	2	5	Galapagos-Island
103	3	2	4	Gotland
104	3	2	4	Greece_1
105	3	2	4	Greenland-off-shore-island
106	3	2	5	Guadeloupe-east
107	3	2	5	Guadeloupe-west
108	3	2	4	Haiti-and-the-Dominican-Republic
109	3	2	9	Hawaiian-Island-Hawaii
110	3	2	9	Hawaiian-Island-Kauai
111	3	2	9	Hawaiian-Island-Maui
112	3	2	9	Hawaiian-Island-Niihau
113	3	2	9	Hawaiian-Island-Oahu
114	3	2	9	Iceland
115	3	2	5	Ile-Kerguelen
116	3	2	7	Indonesia-Alor
117	3	2	7	Indonesia-Bali
118	3	2	7	Indonesia-Bangka
119	3	2	7	Indonesia-Bawean
120	3	2	7	Indonesia-Belitung
121	3	2	7	Indonesia-Biak-and-Kepulauan-Schouten
122	3	2	7	Indonesia-Buru
123	3	2	7	Indonesia-Ceram
124	3	2	7	Indonesia-Flores
125	3	2	7	Indonesia-Halmahera
126	3	2	7	Indonesia-Java
127	3	2	7	Indonesia-Kepulauan-Aru
128	3	2	7	Indonesia-Kepulauan-Tanimbar
129	3	2	7	Indonesia-Nias
130	3	2	7	Indonesia-Obi
131	3	2	7	Indonesia-Siberut
132	3	2	7	Indonesia-Sumba
133	3	2	7	Indonesia-Sumbawa-and-Lombok
134	3	2	7	Indonesia-Timor
135	3	2	7	Indonesia-Waigeo
136	3	2	7	Indonesia-Wetar
137	3	2	7	Indonesia-island
138	3	2	7	Jamaica
139	3	2	5	Long-Island
140	3	2	5	Macias-Nguema-Biyogo-(Fernando-Po)
141	3	2	4	Mallorca
142	3	2	5	Martinique
143	3	2	5	Mauritius
144	3	2	5	New-Caledonia-(island)
145	3	2	5	New-Hebrides-Espiritu-Santo
146	3	2	5	New-Hebrides-Malekula
147	3	2	8	New-Zealand-Chatham-Island
148	3	2	8	New-Zealand-Stewart-Island
149	3	2	4	Outer-Hebrides
150	3	2	6	Philippines-Luzon

151	3	2	6	Philippines-Mindanao
152	3	2	6	Philippines-Mindoro
153	3	2	6	Philippines-Palawan
154	3	2	6	Philippines-Panay-Negros-and-Bohol
155	3	2	6	Philippines-Samar
156	3	2	5	Puerto-Rico
157	3	2	5	Reunion-Island
158	3	2	4	Rhodes
159	3	2	5	Samoa-Savaii
160	3	2	5	Samoa-Upolu
161	3	2	8	Sardinia
162	3	2	8	Sicily
163	3	2	5	Solomon-Island-Bougainville
164	3	2	5	Solomon-Island-Guadalcanal
165	3	2	5	Solomon-Island-San-Cristobal
166	3	2	7	South-Georgia-Island
167	3	2	7	Sri-Lanka-(Ceylon)
168	3	2	6	Svalbard
169	3	2	5	Tahiti
170	3	2	4	Taiwan
171	3	2	8	Trinidad-and-Tobago
172	3	2	6	Turks-and-Caicos-Island
173	3	2	4	USSR-Franz-Josef-Land_1
174	3	2	4	USSR-Franz-Josef-Land_2
175	3	2	4	USSR-Franz-Josef-Land_3
176	3	2	4	USSR-Franz-Josef-Land_4
177	3	2	4	USSR-Franz-Josef-Land_5
178	3	2	4	USSR-Franz-Josef-Land_6
179	3	2	4	USSR-Novaya-Zemlya
180	3	2	4	USSR-Ostrov-Bol'shevik
181	3	2	4	USSR-Ostrov-Bol'shoy-Begichev
182	3	2	4	USSR-Ostrov-Bol'shoy-Lyakhovskiy
183	3	2	4	USSR-Ostrov-Hiiumaa
184	3	2	4	USSR-Ostrov-Iturup
185	3	2	4	USSR-Ostrov-Karaginskiy
186	3	2	4	USSR-Ostrov-Kolguyev
187	3	2	4	USSR-Ostrov-Kotel'nyy-and-Ostrov-Faddeyevskiy
188	3	2	4	USSR-Ostrov-Malyy-Lyakhovskiy
189	3	2	4	USSR-Ostrov-Novaya-Sibir'
190	3	2	4	USSR-Ostrov-Oktyabr'skoy-Revolyutsii-Komsomolets-Pio
191	3	2	4	USSR-Ostrov-Paramushir
192	3	2	4	USSR-Ostrov-Saaremaa
193	3	2	4	USSR-Ostrov-Sakhalin
194	3	2	4	USSR-Ostrov-Urup
195	3	2	4	USSR-Shantarskiye-Ostrova
196	3	2	4	USSR-Wrangel-Island
197	3	2	8	Yemen-(Aden)-Socotra-Island
198	4	3	3	Black-Sea
199	4	3	3	Canada-Great-Bear-Lake
200	4	3	3	Canada-Great-Slave-Lake
201	4	3	3	Canada-Lake-Athabasca
202	4	3	3	Canada-Lake-Manitoba
203	4	3	3	Canada-Lake-Winnipeg
204	4	3	3	Canada-Lake-Winnipegosis
205	4	3	3	Canada-Reindeer-Lake
206	4	3	3	Caspian-Sea
207	4	3	3	Lake-Chad
208	4	3	3	Lake-Erie
209	4	3	3	Lake-Nyasa
210	4	3	3	Lake-Ontario
211	4	3	3	Lake-Superior

212	4	3	3	Lake-Tanganyika
213	4	3	3	Lake-Victoria
214	4	3	3	Lakes-Michigan-and-Huron
215	4	3	3	USSR-Aral-Sea
216	4	3	3	USSR-Azov-Sea
217	4	3	3	USSR-Lake-Baikal
218	4	3	3	USSR-Lake-Balkhash-(east)
219	4	3	3	USSR-Lake-Balkhash-(west)
220	4	3	3	USSR-Onazhskoye-Ozero
221	4	3	3	USSR-Ozero-Ladozhskoye-(Lake-Ladoga)
222	5	2	6	Afghanistan
223	5	2	9	Alaska
224	5	2	6	Albania
225	5	2	5	Algeria
226	5	2	7	Andorra
227	5	2	7	Angola
228	5	2	7	Angola-exclave-called-Cabinda
229	5	2	6	Argentina
230	5	2	6	Argentina-Tierra-del-Fuego
231	5	2	6	Austria
232	5	2	7	Bangladesh
233	5	2	9	Belgium
234	5	2	5	Belize
235	5	2	6	Benin
236	5	2	6	Bhutan
237	5	2	7	Bolivia
238	5	2	6	Borneo-Brunei
239	5	2	9	Botswana
240	5	2	9	Brazil
241	5	2	8	Bulgaria
242	5	2	6	Burma
243	5	2	7	Burundi
244	5	2	5	Cambodia
245	5	2	6	Cameroon
246	5	2	8	Canada
247	5	2	8	Central-African-Republic
248	5	2	5	Chad
249	5	2	5	Chile
250	5	2	5	Chile-Tierra-del-Fuego
251	5	2	8	China
252	5	2	5	Colombia
253	5	2	5	Congo
254	5	2	5	Costa-Rica
255	5	2	5	Czechoslovakia
256	5	2	5	Denmark-Jutland
257	5	2	7	Disputed-area-between-India-and-China
258	5	2	6	Djibouti
259	5	2	6	Dominican-Republic
260	5	2	6	East-Germany
261	5	2	6	Ecuador
262	5	2	9	Egypt
263	5	2	7	El-Salvador
264	5	2	8	England
265	5	2	9	Equatorial-Guinea
266	5	2	7	Ethiopia
267	5	2	6	Finland
268	5	2	6	France
269	5	2	6	French-Guiana
270	5	2	8	Gabon
271	5	2	5	Gambia
272	5	2	8	Ghana

273	5	2	5	Greece_2
274	5	2	6	Guatemala
275	5	2	8	Guinea
276	5	2	7	Guinea-Bissau
277	5	2	7	Guyana
278	5	2	9	Haiti
279	5	2	8	Honduras
280	5	2	8	Hungary
281	5	2	5	India
282	5	2	8	Indonesia-Borneo-Kalimantan
283	5	2	8	Indonesia-New-Guinea-Irian-Jaya
284	5	2	8	Iran
285	5	2	9	Iraq
286	5	2	8	Ireland-Northern
287	5	2	7	Ireland-Republic
288	5	2	5	Israel
289	5	2	9	Italy
290	5	2	7	Ivory-Coast
291	5	2	8	Jordan
292	5	2	8	Kenya
293	5	2	5	Kuwait
294	5	2	9	Laos
295	5	2	6	Lebanon
296	5	2	9	Lesotho
297	5	2	6	Liberia
298	5	2	7	Libya
299	5	2	7	Liechtenstein
300	5	2	8	Luxembourg
301	5	2	5	Malawi
302	5	2	9	Malaysia-Borneo-Malaysia-Timor
303	5	2	9	Malaysia-Malay-Peninsula
304	5	2	9	Mali
305	5	2	7	Mauritania
306	5	2	8	Mexico
307	5	2	6	Mongolia
308	5	2	8	Morocco
309	5	2	7	Mozambique
310	5	2	5	Namibia
311	5	2	6	Nepal
312	5	2	6	Netherlands
313	5	2	6	New-Guinea-Papua
314	5	2	9	Nicaragua
315	5	2	8	Niger
316	5	2	7	Nigeria
317	5	2	5	North-Korea
318	5	2	7	Norway
319	5	2	7	Oman-northern-section
320	5	2	7	Oman-southern-section
321	5	2	7	Pakistan
322	5	2	8	Panama
323	5	2	8	Paraguay
324	5	2	8	Peru
325	5	2	8	Poland
326	5	2	7	Portugal
327	5	2	7	Qatar
328	5	2	6	Romania
329	5	2	8	Rwanda
330	5	2	6	Saudi-Arabia
331	5	2	5	Scotland
332	5	2	6	Senegal
333	5	2	9	Sierra-Leone

334	5	2	5	Somalia
335	5	2	8	South-Africa
336	5	2	7	South-Korea
337	5	2	5	Spain
338	5	2	6	Sudan
339	5	2	8	Surinam
340	5	2	9	Swaziland
341	5	2	8	Sweden
342	5	2	8	Switzerland
343	5	2	7	Syria
344	5	2	6	Tanzania
345	5	2	8	Thailand
346	5	2	9	Togo
347	5	2	8	Tunisia
348	5	2	6	Turkey
349	5	2	9	US
350	5	2	4	USSR
351	5	2	5	Uganda
352	5	2	5	United-Arab-Emirates
353	5	2	7	Unnamed-neutral-zone-between-Iraq-and-Saudi-Arabia
354	5	2	5	Upper-Volta
355	5	2	5	Uruguay
356	5	2	6	Venezuela
357	5	2	7	Vietnam
358	5	2	6	Wales
359	5	2	7	West-Germany
360	5	2	9	Western-Sahara
361	5	2	9	Yemen-(Aden)
362	5	2	8	Yemen-(Sana)
363	5	2	7	Yugoslavia
364	5	2	9	Zaire
365	5	2	8	Zambia
366	5	2	5	Zimbabwe
367	6	2	7	US-Alabama
368	6	2	4	US-Arizona
369	6	2	7	US-Arkansas
370	6	2	6	US-Bahamas-Andros_1
371	6	2	6	US-Bahamas-Andros_2
372	6	2	6	US-Bahamas-Andros_3
373	6	2	6	US-Bahamas-Andros_4
374	6	2	6	US-Bahamas-Andros_5
375	6	2	6	US-Bahamas-Cat-Island
376	6	2	6	US-Bahamas-Eleuthera-Island_1
377	6	2	6	US-Bahamas-Eleuthera-Island_2
378	6	2	6	US-Bahamas-Eleuthera-Island_3
379	6	2	6	US-Bahamas-Grand-Bahama_1
380	6	2	6	US-Bahamas-Grand-Bahama_2
381	6	2	6	US-Bahamas-Grand-Bahama_3
382	6	2	6	US-Bahamas-Great-Abaco-and-Little-Abaco
383	6	2	6	US-Bahamas-Mores-Island
384	6	2	6	US-Bahamas-New-Providence-Island
385	6	2	2	US-Border
386	6	2	6	US-California
387	6	2	6	US-California-San-Clemente-Island
388	6	2	6	US-California-San-Miguel-Island
389	6	2	6	US-California-San-Nicholas-Island
390	6	2	6	US-California-Santa-Catalina-Island
391	6	2	6	US-California-Santa-Cruz-Island
392	6	2	6	US-California-Santa-Rosa-Island
393	6	2	8	US-Colorado
394	6	2	7	US-Connecticut

395	6	2	7	US-Delaware
396	6	2	6	US-Florida
397	6	2	6	US-Florida-Keys_1
398	6	2	6	US-Florida-Keys_2
399	6	2	6	US-Florida-Keys_3
400	6	2	6	US-Florida-Keys_4
401	6	2	6	US-Florida-Pine-Island
402	6	2	6	US-Florida-Sanibel-Island
403	6	2	6	US-Florida-island_1
404	6	2	6	US-Florida-island_2
405	6	2	6	US-Florida-island_3
406	6	2	6	US-Florida-island_4
407	6	2	6	US-Florida-island_5
408	6	2	6	US-Florida-island_6
409	6	2	8	US-Georgia
410	6	2	8	US-Idaho
411	6	2	6	US-Illinois
412	6	2	7	US-Indiana
413	6	2	8	US-Iowa
414	6	2	6	US-Kansas
415	6	2	8	US-Kentucky
416	6	2	4	US-Lake-Champlain-Grand-Isle
417	6	2	4	US-Lake-Erie-Long-Point-(land)
418	6	2	4	US-Lake-Huron-Barrie-Island
419	6	2	4	US-Lake-Huron-Bois-Blanc-Island
420	6	2	4	US-Lake-Huron-Christian-Island
421	6	2	4	US-Lake-Huron-Cockburn-Island
422	6	2	4	US-Lake-Huron-Drummond-Island
423	6	2	4	US-Lake-Huron-Fitzwilliam-Island
424	6	2	4	US-Lake-Huron-Manitoulin-Island
425	6	2	4	US-Lake-Michigan-Beaver-Island
426	6	2	4	US-Lake-Michigan-Manitou-Island
427	6	2	4	US-Lake-Michigan-Washington-Island
428	6	2	4	US-Lake-Ontario-island_1
429	6	2	4	US-Lake-Ontario-island_2
430	6	2	4	US-Lake-Ontario-island_3
431	6	2	4	US-Lake-Superior-Grand-Island
432	6	2	4	US-Lake-Superior-Isle-Royale
433	6	2	4	US-Lake-Superior-Madeline-Island
434	6	2	4	US-Lake-Superior-Michipicoten-Island
435	6	2	4	US-Lake-Superior-Oak-Island
436	6	2	4	US-Lake-Superior-Outer-Island
437	6	2	4	US-Lake-Superior-Pie-Island
438	6	2	4	US-Lake-Superior-Stockton-Island
439	6	2	4	US-Lake-Superior-island
440	6	2	6	US-Louisiana
441	6	2	6	US-Louisiana-Cat-Island
442	6	2	6	US-Louisiana-Marsh-Island
443	6	2	6	US-Louisiana-Timbalier-Island
444	6	2	6	US-Louisiana-island_1
445	6	2	6	US-Louisiana-island_10
446	6	2	6	US-Louisiana-island_11
447	6	2	6	US-Louisiana-island_12
448	6	2	6	US-Louisiana-island_2
449	6	2	6	US-Louisiana-island_3
450	6	2	6	US-Louisiana-island_4
451	6	2	6	US-Louisiana-island_5
452	6	2	6	US-Louisiana-island_6
453	6	2	6	US-Louisiana-island_7
454	6	2	6	US-Louisiana-island_8
455	6	2	6	US-Louisiana-island_9

456	6	2	7	US-Maine
457	6	2	7	US-Maine-island_1
458	6	2	7	US-Maine-island_2
459	6	2	7	US-Maine-island_3
460	6	2	7	US-Maine-island_4
461	6	2	7	US-Maine-island_5
462	6	2	7	US-Maine-island_6
463	6	2	7	US-Maine-island_7
464	6	2	7	US-Maine-island_8
465	6	2	7	US-Maine-island_9
466	6	2	6	US-Maryland-island_1
467	6	2	6	US-Maryland-island_2
468	6	2	6	US-Maryland-island_3
469	6	2	6	US-Maryland-island_4
470	6	2	6	US-Maryland_1
471	6	2	6	US-Maryland_2
472	6	2	6	US-Maryland_3
473	6	2	6	US-Massachusetts
474	6	2	6	US-Massachusetts-Marthas-Vineyard
475	6	2	6	US-Massachusetts-Nantucket-Island
476	6	2	4	US-Michigan-(lower)
477	6	2	4	US-Michigan-(upper)
478	6	2	6	US-Minnesota
479	6	2	6	US-Minnesota-Lake-of-the-Woods-island_1
480	6	2	6	US-Minnesota-Lake-of-the-Woods-island_2
481	6	2	6	US-Minnesota-Lake-of-the-Woods-island_3
482	6	2	6	US-Minnesota-Lake-of-the-Woods-land-west-of-lake
483	6	2	6	US-Minnesota-little-piece
484	6	2	4	US-Mississippi
485	6	2	4	US-Missouri
486	6	2	7	US-Montana
487	6	2	7	US-Nebraska
488	6	2	7	US-Nevada
489	6	2	5	US-New-Hampshire
490	6	2	4	US-New-Jersey
491	6	2	4	US-New-Jersey-island
492	6	2	7	US-New-Mexico
493	6	2	5	US-New-York
494	6	2	5	US-New-York-Fishers-Island
495	6	2	5	US-New-York-Long-Island
496	6	2	5	US-New-York-Shelter-Island
497	6	2	6	US-North-Carolina
498	6	2	6	US-North-Carolina-Cape-Fear
499	6	2	6	US-North-Carolina-island_1
500	6	2	6	US-North-Carolina-island_10
501	6	2	6	US-North-Carolina-island_2
502	6	2	6	US-North-Carolina-island_3
503	6	2	6	US-North-Carolina-island_4
504	6	2	6	US-North-Carolina-island_5
505	6	2	6	US-North-Carolina-island_6
506	6	2	6	US-North-Carolina-island_7
507	6	2	6	US-North-Carolina-island_8
508	6	2	6	US-North-Carolina-island_9
509	6	2	5	US-North-Dakota
510	6	2	6	US-Ohio
511	6	2	5	US-Oklahoma
512	6	2	5	US-Oregon
513	6	2	8	US-Pennsylvania
514	6	2	8	US-Rhode-Island
515	6	2	8	US-Rhode-Island-Block-Island
516	6	2	8	US-Rhode-Island-island_1

517	6	2	8	US-Rhode-Island-island_2
518	6	2	8	US-Rhode-Island-island_3
519	6	2	4	US-South-Carolina
520	6	2	4	US-South-Dakota
521	6	2	5	US-Staten-Island
522	6	2	5	US-Tennessee
523	6	2	4	US-Texas
524	6	2	4	US-Texas-Padre-Island
525	6	2	4	US-Texas-island_1
526	6	2	4	US-Texas-island_2
527	6	2	5	US-Utah
528	6	2	4	US-Vermont
529	6	2	4	US-Virginia-island_1
530	6	2	4	US-Virginia-island_2
531	6	2	4	US-Virginia-island_3
532	6	2	4	US-Virginia-island_4
533	6	2	4	US-Virginia_1
534	6	2	4	US-Virginia_2
535	6	2	4	US-Virginia_3
536	6	2	4	US-Virginia_4
537	6	2	4	US-Washington
538	6	2	4	US-Washington-island_1
539	6	2	4	US-Washington-island_2
540	6	2	4	US-Washington-island_3
541	6	2	4	US-Washington-island_4
542	6	2	4	US-Washington-island_5
543	6	2	4	US-Washington-island_6
544	6	2	4	US-Washington-island_7
545	6	2	5	US-West-Virginia
546	6	2	5	US-Wisconsin
547	6	2	6	US-Wyoming
548	7	3	3	US-Canada-lake
549	7	3	3	US-Clark-Hill-Reservoir
550	7	3	3	US-Delaware-bay
551	7	3	3	US-Florida-Lake-Okeechobee
552	7	3	3	US-Florida-lake
553	7	3	3	US-Great-Salt-Lake
554	7	3	3	US-Lake-Champlain
555	7	3	3	US-Lake-Erie
556	7	3	3	US-Lake-George
557	7	3	3	US-Lake-Mead-(east)
558	7	3	3	US-Lake-Mead-(west)
559	7	3	3	US-Lake-Ontario
560	7	3	3	US-Lake-Seminole
561	7	3	3	US-Lake-Saint-Clair
562	7	3	3	US-Lake-Superior
563	7	3	3	US-Lake-Tahoe
564	7	3	3	US-Lake-Texoma
565	7	3	3	US-Lakes-Michigan-and-Huron
566	7	3	3	US-Long-Island-Great-South-Bay
567	7	3	3	US-Louisiana-Calcasieu-Lake
568	7	3	3	US-Louisiana-Grand-Lake
569	7	3	3	US-Louisiana-Lake-Maurepas
570	7	3	3	US-Louisiana-Lake-Pontchartrain
571	7	3	3	US-Louisiana-Sabine-Lake
572	7	3	3	US-Louisiana-White-Lake
573	7	3	3	US-Maryland-inlet
574	7	3	3	US-Massachusetts-bay
575	7	3	3	US-Massachusetts-inlet
576	7	3	3	US-Minnesota-Lake-of-the-Woods
577	7	3	3	US-Minnesota-Namakan-Lake

578	7	3	3	US-Minnesota-Rainy-Lake
579	7	3	3	US-Minnesota-Shoal-Lake
580	7	3	3	US-New-Hampshire-lake_1
581	7	3	3	US-New-Hampshire-lake_2
582	7	3	3	US-New-Jersey-Newark-Bay
583	7	3	3	US-Sault-Sainte-Marie

MapPlot Ncarg4_1 database table

This table lists the contents of the **MapPlot** `Ncarg4_1` database. Each named area in the `Ncarg4_1` database has a position in a hierarchical tree. At the top level there are only two areas: "Land" and "Water." All other areas are descendents of these areas. Each area has a basic name and also has a fully qualified name derived by determining its ancestor chain up to the top level. An area may have several parents at the same level. For instance, all the islands off the coast of California, a level 4 entity, are children of California and are also level 4 entities.

Generally **MapPlot** returns just the basic name as the value of the *mpAreaNames* array resource. However, the simple names are not necessarily unique. There are, for example, three islands in the database called "Long Island." In order to provide a name that will uniquely identify the area in these cases, **MapPlot** prefixes the name with the immediate parent. If the parent is at a higher level (lower value of *mpAreaType*), then a colon character (':') is placed between the names. If the parent is at the same level, a period character ('.') is used instead. The name strings can be used as returned in *mpAreaNames* to provide unique names to the specifier resources (*mpFillAreaSpecifiers*, *mpOutlineSpecifiers*, or *mpMaskAreaSpecifiers*). If a non-unique basic name (such as "Long Island") is supplied to a specifier resource, all matching areas will be added to the specifier list.

One other point: names that are parenthesized represent an abstract intermediate level that implies ownership of the children (without full inclusion) by the parenthesized entity, as, for example, territories of the United States. These relationships are much easier to grasp by perusing the table listing the full names of the areas that follows the basic table.

The last column contains the area names, as returned by the **MapPlot** class resource *mpAreaNames*. These names are considered the primary key to the data. Their order in the database is given by the first column of the table, labelled "ELEMENT NUMBER."

The second column contains an integer representation of the area type. You can retrieve an array of these values by getting the read-only array resource, *mpAreaTypes*.

The third column contains the fixed area group assigned to each area. This value is used to determine the "geophysical" fill coloring of each area. Since it is based only on whether an area is ocean, land, or inland water, it cannot be modified by the user. You can access these values by getting the array resource *mpFixedAreaGroups*.

The fourth column contains the dynamic area group assigned to each area. This group is used to determine the "political" fill coloring assigned to each area in the database. In general, it is set up such that distinct adjoining political areas will be different colors. Note, however, that type 2 (mpContinental) areas do not always have a distinct color from type 3 (mpNational) or type 4 (mpState) areas they contain or that are adjacent to them. The user can reset these values for other purposes, such as to represent statistical data. You can access these values through the array resource *mpDynamicAreaGroups*.

ELEMENT NUMBER	TYPE	FIXED GROUP	DYNAM. GROUP	AREA NAME
1	1	2	2	Land
2	2	2	5	Africa
3	3	2	8	(Morocco)
4	3	2	8	Western Sahara
5	3	2	4	Algeria
6	3	2	6	Angola
7	3	2	6	exclave called Cabinda
8	3	2	5	Benin
9	3	2	8	Botswana
10	3	2	4	Burkina Faso
11	3	2	6	Burundi
12	3	2	5	Cameroon
13	3	2	7	Central African Republic
14	3	2	4	Chad
15	3	2	4	Congo
16	3	2	5	Djibouti
17	3	2	8	Egypt
18	3	2	8	Equatorial Guinea
19	3	2	8	Bioko
20	3	2	8	Rio Muni
21	3	2	6	Ethiopia
22	3	2	6	Dahlak Archipelago
23	3	2	6	Dehalak
24	3	2	6	Nora Island
25	3	2	7	Gabon
26	3	2	7	Ghana
27	3	2	7	Guinea
28	3	2	6	Guinea-Bissau
29	3	2	6	Arquipelago dos Bijagos
30	3	2	6	Isla de Orango
31	3	2	6	Ivory Coast
32	3	2	7	Kenya
33	3	2	8	Lesotho
34	3	2	5	Liberia
35	3	2	6	Libya
36	3	2	4	Malawi
37	3	2	8	Mali
38	3	2	6	Mauritania
39	3	2	7	Morocco
40	3	2	6	Mozambique
41	3	2	4	Namibia
42	3	2	7	Niger
43	3	2	6	Nigeria
44	3	2	7	Rwanda
45	3	2	5	Senegal
46	3	2	8	Sierra Leone
47	3	2	8	Sherbro Island
48	3	2	4	Somalia
49	3	2	7	South Africa
50	3	2	5	Sudan

51	3	2	8	Swaziland
52	3	2	5	Tanzania
53	3	2	5	Pemba Island
54	3	2	5	Zanzibar Island
55	3	2	4	The Gambia
56	3	2	8	Togo
57	3	2	7	Tunisia
58	3	2	7	Jerba Island
59	3	2	4	Uganda
60	3	2	8	Zaire
61	3	2	7	Zambia
62	3	2	5	Zimbabwe
63	2	2	9	Antarctica
64	2	2	9	Adelaide Island
65	2	2	9	Anvers Island
66	2	2	9	Bear Island
67	2	2	9	Berkner Island
68	2	2	9	Brabant Island
69	2	2	9	Burke Island
70	2	2	9	Charcot Island
71	2	2	9	Coronation Island
72	2	2	9	Coulman Island
73	2	2	9	Elephant Island
74	2	2	9	James Ross Island
75	2	2	9	Joinville Island
76	2	2	9	King George Island
77	2	2	9	Livingston Island
78	2	2	9	Newman Island
79	2	2	9	Peter I Island
80	2	2	9	Renaud Island
81	2	2	9	Snow Hill Island
82	2	2	9	Antarctica . unidentified islands
83	2	2	7	Atlantic islands
84	3	2	4	Atlantic islands : (Denmark)
85	3	2	4	Faeroe Islands
86	3	2	4	Eysturoy
87	3	2	4	Nordoyar
88	3	2	4	Sandoy
89	3	2	4	Streymoy
90	3	2	4	Suduroy
91	3	2	4	Vagar
92	3	2	8	(Equatorial Guinea)
93	3	2	8	Annobon
94	3	2	6	(Norway)
95	3	2	6	Byornoya
96	3	2	6	Jan Mayen
97	3	2	6	Svalbard
98	3	2	6	Barentsoya
99	3	2	6	Edgeoya
100	3	2	6	Kong Karls Land
101	3	2	6	Kongsoya
102	3	2	6	Svenskoya
103	3	2	6	Kvitoya
104	3	2	6	Nordauslandet
105	3	2	6	Prins Karls Forland
106	3	2	6	Spitsbergen
107	3	2	6	(Portugal)
108	3	2	6	Azores
109	3	2	6	Faial
110	3	2	6	Azores . Flores
111	3	2	6	Pico

112	3	2	6	Santa Maria
113	3	2	6	Sao Jorge
114	3	2	6	Sao Miguel
115	3	2	6	Terceira
116	3	2	6	Madeira Islands
117	3	2	6	Madeira
118	3	2	6	Porto Santo
119	3	2	4	(Spain)
120	3	2	4	Canary Islands
121	3	2	4	Fuerteventura
122	3	2	4	Gran Canaria
123	3	2	4	Lanzarote
124	3	2	4	Tenerife
125	3	2	7	Atlantic islands : (United Kingdom)
126	3	2	7	Bermuda Islands
127	3	2	7	Falkland Islands
128	3	2	7	East Falkland
129	3	2	7	Weddell Island
130	3	2	7	West Falkland
131	3	2	7	South Georgia Island
132	3	2	7	South Sandwich Island
133	3	2	6	Cape Verde
134	3	2	6	Boa Vista
135	3	2	6	Fogo
136	3	2	6	Sal
137	3	2	6	Santo Antao
138	3	2	6	Sao Nicolau
139	3	2	6	Sao Tiago
140	3	2	4	Sao Tome and Principe
141	3	2	4	Principe
142	3	2	4	Sao Tome
143	2	2	7	Land : Australia
144	3	2	5	Australia : Australia
145	3	2	5	Australia . Bathurst Island
146	3	2	5	Cape Barren Island
147	3	2	5	Flinders Island
148	3	2	5	Fraser Island
149	3	2	5	Groote Eylandt
150	3	2	5	Kangaroo Island
151	3	2	5	King Island
152	3	2	5	Marchinbar Island
153	3	2	5	Australia . Melville Island
154	3	2	5	Mornington Island
155	3	2	5	North Stradbroke Island
156	3	2	5	Tasmania
157	2	2	6	Caribbean islands
158	2	2	6	Hispaniola
159	3	2	7	Dominican Republic
160	3	2	8	Haiti
161	3	2	8	Ile de la Gonave
162	3	2	8	Ile de la Tortue
163	3	2	5	Caribbean islands : (France)
164	3	2	5	Guadeloupe
165	3	2	5	Martinique
166	3	2	5	(Netherlands)
167	3	2	5	Bonaire
168	3	2	5	Curacao
169	3	2	7	Caribbean islands : (United Kingdom)
170	3	2	7	Turks and Caicos Islands
171	3	2	5	Caribbean islands : (United States)
172	3	2	5	Puerto Rico

173	3	2	5	Saint Croix
174	3	2	5	Saint Thomas
175	3	2	7	Antigua and Barbuda
176	3	2	7	Antigua
177	3	2	5	Bahamas
178	3	2	5	Abaco Island
179	3	2	5	Acklins and Crooked Islands
180	3	2	5	Andros Island
181	3	2	5	Bahamas . Cat Island
182	3	2	5	Eleuthera
183	3	2	5	Grand Bahama Island
184	3	2	5	Grand Inagua Island
185	3	2	5	Bahamas . Long Island
186	3	2	5	Mayaguana Island
187	3	2	5	Moore’s Island
188	3	2	5	New Providence
189	3	2	5	San Salvador
190	3	2	6	Cuba
191	3	2	6	Archipielago de Camaguey
192	3	2	6	Isla de Pinos
193	3	2	7	Dominica
194	3	2	8	Grenada
195	3	2	7	Jamaica
196	3	2	7	Saint Kitts and Nevis
197	3	2	7	Saint Kitts
198	3	2	7	Saint Lucia
199	3	2	7	Saint Vincent and the Grenadines
200	3	2	7	Saint Vincent
201	3	2	8	Trinidad and Tobago
202	3	2	8	Tobago
203	3	2	8	Trinidad
204	2	2	8	Central America
205	3	2	4	Belize
206	3	2	4	Costa Rica
207	3	2	6	El Salvador
208	3	2	5	Guatemala
209	3	2	7	Honduras
210	3	2	8	Nicaragua
211	3	2	7	Panama
212	3	2	7	Isla de Coiba
213	2	2	8	Eurasia
214	3	2	5	Afghanistan
215	3	2	5	Albania
216	3	2	6	Andorra
217	3	2	5	Austria
218	3	2	7	Bahrain
219	3	2	6	Bangladesh
220	3	2	6	Shabbaz Island
221	3	2	4	Belarus
222	3	2	8	Belgium
223	3	2	5	Bhutan
224	3	2	7	Bosnia and Herzegovina
225	3	2	7	Bulgaria
226	3	2	4	Cambodia
227	3	2	7	China
228	3	2	7	Hainan Dao
229	3	2	7	Hong Kong Island
230	3	2	7	Lantau Island
231	3	2	5	Croatia
232	3	2	5	Brac
233	3	2	5	Cres

234	3	2	5	Korcula
235	3	2	5	Krk
236	3	2	5	Cyprus
237	3	2	8	Czech Republic
238	3	2	4	Denmark
239	3	2	4	Bornholm
240	3	2	4	Falster
241	3	2	4	Fyn
242	3	2	4	Jutland
243	3	2	4	Langeland
244	3	2	4	Lolland
245	3	2	4	Sjaelland
246	3	2	6	Disputed area between India and China
247	3	2	4	Estonia
248	3	2	5	Finland
249	3	2	5	Aland
250	3	2	5	Finland . unidentified islands
251	3	2	5	France
252	3	2	5	Corsica
253	3	2	6	Germany
254	3	2	6	Rugen
255	3	2	6	Greece
256	3	2	6	Andros
257	3	2	6	Crete
258	3	2	6	Evvoia
259	3	2	6	Kefallinia
260	3	2	6	Kerkira
261	3	2	6	Khios
262	3	2	6	Lesvos
263	3	2	6	Limnos
264	3	2	6	Naxos
265	3	2	6	Rhodes
266	3	2	6	Thasos
267	3	2	7	Hungary
268	3	2	4	India
269	3	2	7	Iran
270	3	2	7	Qeshm
271	3	2	8	Iraq
272	3	2	6	Ireland
273	3	2	4	Israel
274	3	2	8	Italy
275	3	2	8	Sardinia
276	3	2	8	Sicily
277	3	2	5	Japan
278	3	2	5	Amakusa Shoto
279	3	2	5	Amami O Shima
280	3	2	5	Awaji-shima
281	3	2	5	Goto Retto
282	3	2	5	Fukue Jima
283	3	2	5	Nakadori Jima
284	3	2	5	Hokkaido
285	3	2	5	Honshu
286	3	2	5	Iki
287	3	2	5	Iriomote Jima
288	3	2	5	Ishigaki Shima
289	3	2	5	Kamino Shima
290	3	2	5	Kyushu
291	3	2	5	Miyako Jima
292	3	2	5	Okinawa
293	3	2	5	Osumi Shoto
294	3	2	5	Tanega Shima

295	3	2	5	Yaku Shima
296	3	2	5	Rishiri-to
297	3	2	5	Sado
298	3	2	5	Shikoku
299	3	2	5	Shimono Shima
300	3	2	5	Tokuno Shima
301	3	2	7	Jordan
302	3	2	4	Kuwait
303	3	2	8	Laos
304	3	2	6	Latvia
305	3	2	5	Lebanon
306	3	2	6	Liechtenstein
307	3	2	5	Lithuania
308	3	2	7	Luxembourg
309	3	2	4	Macedonia
310	3	2	8	Eurasia : Malaysia
311	3	2	8	Malay Peninsula
312	3	2	7	Malta
313	3	2	7	Moldova
314	3	2	5	Mongolia
315	3	2	5	Myanmar
316	3	2	5	Kadan Kyun
317	3	2	5	Kanmaw Kyun
318	3	2	5	Letsok-aw Kyun
319	3	2	5	Nepal
320	3	2	5	Netherlands
321	3	2	5	Texel
322	3	2	5	Netherlands . unidentified islands
323	3	2	4	North Korea
324	3	2	6	Norway
325	3	2	6	Arnoy
326	3	2	6	Hamaroy
327	3	2	6	Hitra
328	3	2	6	Kvaloy - 1
329	3	2	6	Kvaloy - 2
330	3	2	6	Lofoten Vesteralen
331	3	2	6	Andoya
332	3	2	6	Hinnoya
333	3	2	6	Langoya
334	3	2	6	Mageroya
335	3	2	6	Ringvassoy
336	3	2	6	Seiland
337	3	2	6	Senja
338	3	2	6	Soroya
339	3	2	6	Stjernoy
340	3	2	6	Vannoy
341	3	2	6	Oman
342	3	2	6	Jazirat Masirah
343	3	2	6	Pakistan
344	3	2	7	Poland
345	3	2	6	Portugal
346	3	2	6	Qatar
347	3	2	5	Romania
348	3	2	8	Russia
349	3	2	8	Franz Josef Land
350	3	2	8	Ostrov Champ
351	3	2	8	Ostrov Dzheksona
352	3	2	8	Ostrov Gallya
353	3	2	8	Ostrov Greem Bell
354	3	2	8	Ostrov Gukera
355	3	2	8	Ostrov Karla-Aleksandra

356	3	2	8	Ostrov La-Rons'yer
357	3	2	8	Ostrov Luidzhi
358	3	2	8	Ostrov Mak-Klintoka
359	3	2	8	Ostrov Rudol'fa
360	3	2	8	Ostrov Sal'm
361	3	2	8	Ostrov Solsberi
362	3	2	8	Ostrov Tsiglera
363	3	2	8	Ostrov Viner-Neyshtadt
364	3	2	8	Ostrova Belaya Zemlya
365	3	2	8	Zemlya Alexandra
366	3	2	8	Zemlya George
367	3	2	8	Zemlya Vil'cheka
368	3	2	8	Kaliningrad Oblast
369	3	2	8	Novaya Zemlya
370	3	2	8	Ostrov Ayon
371	3	2	8	Ostrov Belkov'skiy
372	3	2	8	Ostrov Belyy
373	3	2	8	Ostrov Bering
374	3	2	8	Ostrov Bol'shoy Begichev
375	3	2	8	Ostrov Bol'shoy Lyakhovskiy
376	3	2	8	Ostrov Bol'shoy Shantar
377	3	2	8	Ostrov Faddeyevskiy
378	3	2	8	Ostrov Hiiumaa
379	3	2	8	Ostrov Isachenko
380	3	2	8	Ostrov Iturup
381	3	2	8	Ostrov Karaginskiy
382	3	2	8	Ostrov Kolguyev
383	3	2	8	Ostrov Kotel'nyy
384	3	2	8	Ostrov Kunashir
385	3	2	8	Ostrov Malyy Lyakhovskiy
386	3	2	8	Ostrov Mednyy
387	3	2	8	Ostrov Mezhdusharskiy
388	3	2	8	Ostrov Novaya Sibir'
389	3	2	8	Ostrov Ol'khon (Lake Baikal)
390	3	2	8	Ostrov Oleniy
391	3	2	8	Ostrov Onekotan
392	3	2	8	Ostrov Paramushir
393	3	2	8	Ostrov Russkiy
394	3	2	8	Ostrov Saaremaa
395	3	2	8	Ostrov Sakhalin
396	3	2	8	Ostrov Shiashkotan
397	3	2	8	Ostrov Shmidta
398	3	2	8	Ostrov Shokal'skogo
399	3	2	8	Ostrov Shumshu
400	3	2	8	Ostrov Sibiryakova
401	3	2	8	Ostrov Simushir
402	3	2	8	Ostrov Stolbovoy
403	3	2	8	Ostrov Taymyr
404	3	2	8	Ostrov Urup
405	3	2	8	Ostrov Ushakova
406	3	2	8	Ostrov Vaygach
407	3	2	8	Ostrov Yarok
408	3	2	8	Ostrova Arkticheskogo Instituta
409	3	2	8	Severnaya Zemlya
410	3	2	8	Ostrov Bol'shevik
411	3	2	8	Ostrov Komsomolets
412	3	2	8	Ostrov Malyy Taymyr
413	3	2	8	Ostrov Oktyabr'skoy Revolyutsii
414	3	2	8	Ostrov Pioner
415	3	2	8	Shikotan
416	3	2	8	Wrangel Island

417	3	2	8	Russia . unidentified islands
418	3	2	5	Saudi Arabia
419	3	2	5	Jaza'ir Farasan
420	3	2	6	Singapore
421	3	2	4	Slovakia
422	3	2	6	Slovenia
423	3	2	6	South Korea
424	3	2	6	Cheju Do
425	3	2	6	Ullung Do
426	3	2	4	Spain
427	3	2	4	Ibiza
428	3	2	4	Mallorca
429	3	2	4	Menorca
430	3	2	7	Sri Lanka (Ceylon)
431	3	2	7	Sweden
432	3	2	7	Gotland
433	3	2	7	Oland
434	3	2	7	Switzerland
435	3	2	6	Syria
436	3	2	8	Taiwan
437	3	2	7	Thailand
438	3	2	7	Ko Phuket
439	3	2	5	Turkey
440	3	2	6	Ukraine
441	3	2	4	United Arab Emirates
442	3	2	4	Abu al Abyad
443	3	2	7	United Kingdom
444	3	2	7	England
445	3	2	7	Isle of Wight
446	3	2	7	Isle of Man
447	3	2	7	Northern Ireland
448	3	2	5	Scotland
449	3	2	5	Inner Hebrides
450	3	2	5	Island of Mull
451	3	2	5	Island of Skye
452	3	2	5	Islay
453	3	2	5	Jura
454	3	2	5	Island of Arran
455	3	2	5	Orkney Islands
456	3	2	5	Dwarfie Stane
457	3	2	5	Mainland
458	3	2	5	Outer Hebrides
459	3	2	5	Isle of Lewis
460	3	2	5	North Uist
461	3	2	5	Shetland Islands
462	3	2	8	Wales
463	3	2	8	Anglesey
464	3	2	6	Vietnam
465	3	2	6	Dao Phu Quoc
466	3	2	7	Yemen
467	4	2	8	Aden
468	4	2	8	Abd-Al-Kuri
469	4	2	8	Socotra
470	4	2	7	Sana
471	4	2	7	Az Zukur
472	4	2	7	Hanish Al Kabir
473	4	2	7	Kamaran
474	3	2	6	Yugoslavia
475	2	2	4	Land : Greenland
476	3	2	4	Greenland : (Denmark)
477	3	2	4	(Denmark) . Greenland

478	3	2	4	Clavering Oy
479	3	2	4	Geographical Society Oy
480	3	2	4	Hovgaard Oy
481	3	2	4	Ile de France
482	3	2	4	Milne Land
483	3	2	4	Norske Oer
484	3	2	4	Qeqertarsuaq
485	3	2	4	Shannon
486	3	2	4	Store Koldewey
487	3	2	4	Traill Oy
488	3	2	4	Ymer Oy
489	3	2	4	Greenland . unidentified islands
490	2	2	5	Land : Iceland
491	3	2	5	Iceland : Iceland
492	2	2	7	Indian Ocean islands
493	3	2	5	Indian Ocean islands : (France)
494	3	2	5	Ile Kerguelen
495	3	2	5	Mayotte (Dzaoudzi)
496	3	2	5	Reunion Island
497	3	2	4	(India)
498	3	2	4	Andaman Islands
499	3	2	4	Nicobar Islands
500	3	2	4	Great Nicobar
501	3	2	7	(South Africa)
502	3	2	7	Prince Edward Islands
503	3	2	7	Marion Island
504	3	2	6	Comoros
505	3	2	6	Mwali
506	3	2	6	Njazidja
507	3	2	6	Nzwani
508	3	2	8	Madagascar
509	3	2	6	Maldives
510	3	2	8	Mauritius
511	3	2	7	Seychelles
512	3	2	7	Aldabra Island
513	3	2	7	Mahe Island
514	2	2	7	North America
515	3	2	5	North America : (France)
516	3	2	5	Saint Pierre and Miquelon
517	3	2	5	Miquelon
518	3	2	8	Canada
519	3	2	8	Air Force Island
520	3	2	8	Akimiski Island
521	3	2	8	Akpatok Island
522	3	2	8	Amherst Island (Lake Ontario)
523	3	2	8	Amund Ringnes Island
524	3	2	8	Anticosti Island
525	3	2	8	Axel Heiberg Island
526	3	2	8	Baffin Island
527	3	2	8	Banks Island - 1
528	3	2	8	Banks Island - 2
529	3	2	8	Barrie Island (Lake Huron)
530	3	2	8	Batchawana Island (Lake Superior)
531	3	2	8	Canada . Bathurst Island
532	3	2	8	Belcher Islands
533	3	2	8	Big Island
534	3	2	8	Big Island (Great Slave Lake)
535	3	2	8	Borden Island
536	3	2	8	Brock Island
537	3	2	8	Byam Martin Island
538	3	2	8	Bylot Island

539	3	2	8	Cape Breton Island
540	3	2	8	Charles Island
541	3	2	8	Christian Island (Lake Huron)
542	3	2	8	Coats Island
543	3	2	8	Coburg Island
544	3	2	8	Cockburn Island (Lake Huron)
545	3	2	8	Cornwall Island
546	3	2	8	Cornwallis Island
547	3	2	8	Devon Island
548	3	2	8	Eglinton Island
549	3	2	8	Ellef Ringnes Island
550	3	2	8	Ellesmere Island
551	3	2	8	Emerald Isle
552	3	2	8	Fitzwilliam Island (Lake Huron)
553	3	2	8	Island of Newfoundland
554	3	2	8	Isle Royale (Lake Superior)
555	3	2	8	Jens Munk Island
556	3	2	8	King Christian Island
557	3	2	8	King William Island
558	3	2	8	Lougheed Island
559	3	2	8	Mackenzie King Island
560	3	2	8	Manitoulin Island (Lake Huron)
561	3	2	8	Mansel Island
562	3	2	8	Meighen Island
563	3	2	8	Canada . Melville Island
564	3	2	8	Michipicoten Island (Lake Superior)
565	3	2	8	Nottingham Island
566	3	2	8	Philpots Island
567	3	2	8	Pie Island (Lake Superior)
568	3	2	8	Pitt Island
569	3	2	8	Porcher Island
570	3	2	8	Prescott Island
571	3	2	8	Prince Charles Island
572	3	2	8	Prince Edward Island
573	3	2	8	Prince Patrick Island
574	3	2	8	Canada . Prince of Wales Island
575	3	2	8	Queen Charlotte Islands
576	3	2	8	Graham Island
577	3	2	8	Moresby Island
578	3	2	8	Resolution Island
579	3	2	8	Richards Island
580	3	2	8	Rowley Island
581	3	2	8	Russell Island
582	3	2	8	Salisbury Island
583	3	2	8	Simpson Island (Great Slave Lake)
584	3	2	8	Somerset Island
585	3	2	8	Southampton Island
586	3	2	8	Stefansson Island
587	3	2	8	Vancouver Island
588	3	2	8	Vansittart Island
589	3	2	8	Victoria Island
590	3	2	8	Wales Island
591	3	2	8	Wolfe Island (Lake Ontario)
592	3	2	8	unidentified island (Great Slave Lake)
593	3	2	8	Canada . unidentified islands
594	3	2	8	Mexico
595	3	2	8	Angel de la Guarda
596	3	2	8	Isla Cedros
597	3	2	8	Isla de Cozumel
598	3	2	8	Isla de Guadalupe
599	3	2	8	Islas Revillagigedo

600	3	2	8	Tiburon
601	3	2	5	North America : United States
602	3	2	5	Conterminous US
603	4	2	8	Alabama
604	4	2	7	Arizona
605	4	2	4	Arkansas
606	4	2	5	California
607	4	2	5	Channel Islands
608	4	2	5	San Clemente Island
609	4	2	5	San Miguel Island
610	4	2	5	San Nicholas Island
611	4	2	5	Santa Catalina Island
612	4	2	5	Channel Islands . Santa Cruz Island
613	4	2	5	Santa Rosa Island
614	4	2	4	Colorado
615	4	2	8	Connecticut
616	4	2	5	Delaware
617	4	2	6	Florida
618	4	2	6	Keys
619	4	2	6	Pine Island
620	4	2	6	Saint Vincent Island
621	4	2	6	Sanibel Island
622	4	2	5	Georgia
623	4	2	4	Idaho
624	4	2	4	Illinois
625	4	2	5	Indiana
626	4	2	6	Iowa
627	4	2	6	Kansas
628	4	2	8	Kentucky
629	4	2	5	Louisiana
630	4	2	5	Louisiana . Cat Island
631	4	2	5	Grand Terre Island
632	4	2	5	Isle au Pitre and others
633	4	2	5	Isles Dernieres
634	4	2	5	Marsh Island
635	4	2	5	Timbalier Island
636	4	2	5	Louisiana . unidentified islands
637	4	2	6	Maine
638	4	2	6	Deer Isle
639	4	2	6	Great Wass Island
640	4	2	6	Head Harbor Island
641	4	2	6	Isle au Haut
642	4	2	6	Islesboro Island
643	4	2	6	Mount Desert Island
644	4	2	6	Swans Island
645	4	2	6	Vinalhaven Island
646	4	2	8	Maryland
647	4	2	8	Assateague Island
648	4	2	8	Bloodsworth Island
649	4	2	8	Deal Island
650	4	2	8	Smith Island
651	4	2	8	Maryland . unidentified islands
652	4	2	4	Massachusetts
653	4	2	4	Martha’s Vineyard
654	4	2	4	Nantucket Island
655	4	2	6	Michigan
656	4	2	6	Beaver Island (Lake Michigan)
657	4	2	6	Bois Blanc Island (Lake Huron)
658	4	2	6	Drummond Island (Lake Huron)
659	4	2	6	Grand Island (Lake Superior)
660	4	2	6	Manitou Island (Lake Michigan)

661	4	2	7	Minnesota
662	4	2	7	Big Island (Lake of the Woods)
663	4	2	7	Bigsby Island (Lake of the Woods)
664	4	2	6	Mississippi
665	4	2	5	Missouri
666	4	2	5	Montana
667	4	2	7	Nebraska
668	4	2	6	Nevada
669	4	2	5	New Hampshire
670	4	2	6	New Jersey
671	4	2	6	New Mexico
672	4	2	5	New York
673	4	2	5	Fishers Island
674	4	2	5	Grand Isle (Lake Champlain)
675	4	2	5	New York . Long Island
676	4	2	5	Shelter Island
677	4	2	5	Staten Island
678	4	2	5	Thousand Island (Lake Ontario)
679	4	2	4	North Carolina
680	4	2	4	Cape Fear
681	4	2	4	Hatteras Island
682	4	2	4	Outer Banks
683	4	2	4	Roanoke Island
684	4	2	4	North Dakota
685	4	2	4	Ohio
686	4	2	8	Oklahoma
687	4	2	8	Oregon
688	4	2	7	Pennsylvania
689	4	2	6	Rhode Island
690	4	2	6	Block Island
691	4	2	6	Rhode Island . unidentified islands
692	4	2	8	South Carolina
693	4	2	8	South Dakota
694	4	2	7	Tennessee
695	4	2	7	Texas
696	4	2	7	Galveston Island
697	4	2	7	Padre and Matagorda Islands
698	4	2	8	Utah
699	4	2	6	Vermont
700	4	2	6	Virginia
701	4	2	6	Washington
702	4	2	6	San Juan Islands
703	4	2	5	West Virginia
704	4	2	8	Wisconsin
705	4	2	8	Madeline Island (Lake Superior)
706	4	2	8	Oak Island (Lake Superior)
707	4	2	8	Outer Island (Lake Superior)
708	4	2	8	Stockton Island (Lake Superior)
709	4	2	8	Washington Island (Lake Michigan)
710	4	2	6	Wyoming
711	4	2	5	Alaska
712	4	2	5	Adak Island
713	4	2	5	Admiralty Island
714	4	2	5	Afognak Island
715	4	2	5	Akutan Island
716	4	2	5	Amchitka Island
717	4	2	5	Amlia Island
718	4	2	5	Annette Island
719	4	2	5	Atka Island
720	4	2	5	Attu Island
721	4	2	5	Baranof Island

722	4	2	5	Chichagof Island
723	4	2	5	Dall Island
724	4	2	5	Hagemeister Island
725	4	2	5	Hinchinbrook Island
726	4	2	5	Kanaga Island
727	4	2	5	Kiska Island
728	4	2	5	Kodiak Island
729	4	2	5	Kuiu Island
730	4	2	5	Kupreanof Island
731	4	2	5	Mitkof Island
732	4	2	5	Mitrofania Island
733	4	2	5	Montague Island
734	4	2	5	Nelson Island
735	4	2	5	Nunivak Island
736	4	2	5	Pribilof Islands
737	4	2	5	Saint George Island
738	4	2	5	Saint Paul Island
739	4	2	5	Alaska . Prince of Wales Island
740	4	2	5	Raspberry Island
741	4	2	5	Revillagigedo Island
742	4	2	5	Saint Lawrence Island
743	4	2	5	Saint Matthew Island
744	4	2	5	Sanak Island
745	4	2	5	Seguam Island
746	4	2	5	Semisopochnoi Island
747	4	2	5	Shuyak Island
748	4	2	5	Sitkalidak Island
749	4	2	5	Sitkinak Island
750	4	2	5	Swindle Island
751	4	2	5	Tanaga Island
752	4	2	5	Tugidak Island
753	4	2	5	Umnak Island
754	4	2	5	Unalaska Island
755	4	2	5	Unga Island
756	4	2	5	Unimak Island
757	4	2	5	Wrangell Island
758	4	2	5	Zarembo Island
759	2	2	6	Pacific islands
760	2	2	6	Borneo
761	3	2	5	Brunei
762	3	2	7	Borneo : Indonesia
763	3	2	7	Kalimantan
764	3	2	8	Borneo : Malaysia
765	3	2	8	Malaysia Timor
766	2	2	6	New Guinea
767	3	2	7	New Guinea : Indonesia
768	3	2	7	Irian Jaya
769	3	2	5	New Guinea : Papua New Guinea
770	3	2	6	(Australia)
771	3	2	6	Christmas Island
772	3	2	4	(Chile)
773	3	2	4	Easter Island
774	3	2	5	(Ecuador)
775	3	2	5	Galapagos Islands
776	3	2	5	Isla Fernandina
777	3	2	5	Isla Isabela
778	3	2	5	Isla San Cristobal
779	3	2	5	Isla San Salvador
780	3	2	5	Isla Santa Cruz
781	3	2	5	Pacific islands : (France)
782	3	2	5	French Polynesia

783	3	2	5	Marquesas Islands
784	3	2	5	Moorea
785	3	2	5	Raiatea and Tahaa
786	3	2	5	Tahiti
787	3	2	5	Tuamotu Archipelago
788	3	2	5	Anaa
789	3	2	5	Fangatau
790	3	2	5	Hereheretue
791	3	2	5	Manihi
792	3	2	5	Marutea
793	3	2	5	Morane
794	3	2	5	Takaroa
795	3	2	5	Tematagi
796	3	2	5	unidentified
797	3	2	5	New Caledonia
798	3	2	5	Iles Loyaute (Loyalty Islands)
799	3	2	5	Lifu
800	3	2	5	Mare
801	3	2	5	(Japan)
802	3	2	5	Minami Tori Shima
803	3	2	8	(New Zealand)
804	3	2	8	Manihiki
805	3	2	8	Niue
806	3	2	8	Penrhyn
807	3	2	8	Tokelau
808	3	2	7	Pacific islands : (United Kingdom)
809	3	2	7	Ducie Island
810	3	2	7	Pitcairn Island
811	3	2	5	Pacific islands : (United States)
812	3	2	5	Baker Island
813	3	2	5	Guam
814	3	2	5	Northern Mariana Islands
815	3	2	5	Asuncion
816	3	2	5	Guguan
817	3	2	5	Saipan
818	3	2	5	Palmyra Atoll
819	3	2	5	Wake Island
820	3	2	6	Fiji
821	3	2	6	Vanua Levu
822	3	2	6	Viti Levu
823	3	2	7	Pacific islands : Indonesia
824	3	2	7	Alor
825	3	2	7	Ambon
826	3	2	7	Babar
827	3	2	7	Bali
828	3	2	7	Bangka
829	3	2	7	Bawean
830	3	2	7	Belitung
831	3	2	7	Bengkalis
832	3	2	7	Biak and Supiori
833	3	2	7	Bintan
834	3	2	7	Buru
835	3	2	7	Buton
836	3	2	7	Ceram
837	3	2	7	Damar
838	3	2	7	Indonesia . Flores
839	3	2	7	Halmahera
840	3	2	7	Java
841	3	2	7	Kabaena
842	3	2	7	Kai Kecil
843	3	2	7	Karakelong

844	3	2	7	Kasiruta
845	3	2	7	Kepulauan Aru
846	3	2	7	Maikoor
847	3	2	7	Trangan
848	3	2	7	Wokam
849	3	2	7	Kepulauan Kangean
850	3	2	7	Kepulauan Tanimbar
851	3	2	7	Laut
852	3	2	7	Lomblen
853	3	2	7	Lombok
854	3	2	7	Mangole
855	3	2	7	Misool
856	3	2	7	Morotai
857	3	2	7	Muna
858	3	2	7	Nias
859	3	2	7	Obi
860	3	2	7	Pantar
861	3	2	7	Peleng
862	3	2	7	Roti
863	3	2	7	Rupat
864	3	2	7	Salawati
865	3	2	7	Sanana
866	3	2	7	Sangihe
867	3	2	7	Siberut
868	3	2	7	Simuelue
869	3	2	7	Singkep
870	3	2	7	Sulawesi (Celebes)
871	3	2	7	Sumatra
872	3	2	7	Sumba
873	3	2	7	Sumbawa and Lombok
874	3	2	7	Taliabu
875	3	2	7	Tanjung Pemali
876	3	2	7	Timor
877	3	2	7	Waigeo
878	3	2	7	Wetar
879	3	2	7	Wowoni
880	3	2	7	Yapen
881	3	2	6	Kiribati
882	3	2	6	Banaba (Ocean Island)
883	3	2	6	Nonouti
884	3	2	6	Phoenix Islands
885	3	2	6	Kanton
886	3	2	6	Rawaki
887	3	2	6	Marshall Islands
888	3	2	6	Jaluit
889	3	2	6	Kwajalein Atoll
890	3	2	6	Rongelap
891	3	2	6	Wotje
892	3	2	6	Micronesia
893	3	2	6	Kosrae
894	3	2	6	Mortlock Island
895	3	2	6	Senyavin Island
896	3	2	6	Truk
897	3	2	6	Yap
898	3	2	8	New Zealand
899	3	2	8	Auckland Islands
900	3	2	8	Chatham Islands
901	3	2	8	North Island
902	3	2	8	South Island
903	3	2	8	Stewart Island
904	3	2	5	Palau

905	3	2	5	Pacific islands : Papua New Guinea
906	3	2	5	Bismarck Archipelago
907	3	2	5	Bismarck Archipelago . Long Island
908	3	2	5	Manus
909	3	2	5	Mussau
910	3	2	5	New Britain
911	3	2	5	New Hanover
912	3	2	5	New Ireland
913	3	2	5	Umboi
914	3	2	5	D'Entre Casteaux Islands
915	3	2	5	Fergusson Island
916	3	2	5	Normanby Island
917	3	2	5	Papua New Guinea . Solomon Islands
918	3	2	5	Bougainville
919	3	2	5	Buka
920	3	2	5	Woodlark Island
921	3	2	6	Philippines
922	3	2	6	Basilan
923	3	2	6	Biliran
924	3	2	6	Bohol
925	3	2	6	Burias
926	3	2	6	Busuanga
927	3	2	6	Cebu
928	3	2	6	Culion
929	3	2	6	Dinagat
930	3	2	6	Guimaras
931	3	2	6	Jolo
932	3	2	6	Leyte
933	3	2	6	Luzon
934	3	2	6	Marinduque
935	3	2	6	Masbate
936	3	2	6	Mindanao
937	3	2	6	Mindoro
938	3	2	6	Negros
939	3	2	6	Palawan
940	3	2	6	Panay
941	3	2	6	Polillo Islands
942	3	2	6	Samar
943	3	2	6	Siargao
944	3	2	6	Sibuyan
945	3	2	6	Tablas
946	3	2	6	Tawi Tawi
947	3	2	6	Ticao
948	3	2	6	Pacific islands : Solomon Islands
949	3	2	6	Choiseul
950	3	2	6	Guadalcanal
951	3	2	6	Kolombangara
952	3	2	6	Malaita
953	3	2	6	New Georgia
954	3	2	6	Nggela Sule
955	3	2	6	Rendova
956	3	2	6	Rennell (Mu Nggava)
957	3	2	6	San Cristobal
958	3	2	6	Solomon Islands . Santa Cruz Island
959	3	2	6	Santa Isabel
960	3	2	6	Vangunu
961	3	2	6	Vella Lavella
962	3	2	6	Tonga
963	3	2	6	Tuvalu
964	3	2	5	Pacific islands : United States
965	4	2	5	Hawaii (state)

966	4	2	5	Hawaii (island)
967	4	2	5	Kahoolawe
968	4	2	5	Kauai
969	4	2	5	Lanai
970	4	2	5	Maui
971	4	2	5	Molokai
972	4	2	5	Niihau
973	4	2	5	Oahu
974	4	2	5	islet (Pearl Harbor)
975	4	2	5	islet off south coast
976	3	2	7	Vanuatu
977	3	2	7	Ambrym
978	3	2	7	Efate
979	3	2	7	Erromango
980	3	2	7	Espiritu Santo
981	3	2	7	Malekula
982	3	2	7	Pentecost Island
983	3	2	7	Tanna
984	3	2	5	Western Samoa
985	3	2	5	Savai'i
986	3	2	5	Upolu
987	2	2	5	South America
988	3	2	5	South America : (France)
989	3	2	5	French Guiana
990	3	2	5	Argentina
991	3	2	5	Isla de Los Estados
992	3	2	5	Argentina . Tierra del Fuego
993	3	2	6	Bolivia
994	3	2	8	Brazil
995	3	2	8	Ilha Caviana
996	3	2	8	Ilha Janaucu
997	3	2	8	Ilha Maranjo
998	3	2	8	Ilha Mexiana
999	3	2	8	Ilha Queimada
1000	3	2	8	Brazil . unidentified islands
1001	3	2	4	Chile
1002	3	2	4	Isla Capitan Arecena
1003	3	2	4	Isla Dawson
1004	3	2	4	Isla Desolacion
1005	3	2	4	Isla Gordon
1006	3	2	4	Isla Hoste
1007	3	2	4	Isla Lennox
1008	3	2	4	Isla Londonderry
1009	3	2	4	Isla Magdalena
1010	3	2	4	Isla Navarino
1011	3	2	4	Isla Nueva
1012	3	2	4	Isla Picton
1013	3	2	4	Isla Riesco
1014	3	2	4	Isla Santa Ines
1015	3	2	4	Isla Stewart
1016	3	2	4	Isla Wellington
1017	3	2	4	Isla de Chiloe
1018	3	2	4	Chile . Tierra del Fuego
1019	3	2	4	Chile . unidentified islands
1020	3	2	4	Colombia
1021	3	2	5	Ecuador
1022	3	2	6	Guyana
1023	3	2	7	Paraguay
1024	3	2	7	Peru
1025	3	2	7	Suriname
1026	3	2	4	Uruguay

1027	3	2	5	Venezuela
1028	3	2	5	Isla de Margarita
1029	1	1	1	Water
1030	2	3	3	Aral Sea (Eurasia)
1031	2	3	3	Azov Sea (Eurasia)
1032	2	3	3	Black Sea (Eurasia)
1033	2	3	3	Buheirat Murrat el Kubra (Africa)
1034	2	3	3	Buheirat el Manzala (Africa)
1035	2	3	3	Calcasieu Lake (North America)
1036	2	3	3	Caspian Sea (Eurasia)
1037	2	3	3	Clark Hill Reservoir (North America)
1038	2	3	3	Grand Lake (North America)
1039	2	3	3	Great Bear Lake (North America)
1040	2	3	3	Great Salt Lake (North America)
1041	2	3	3	Great Slave Lake (North America)
1042	2	3	3	Great South Bay (North America)
1043	2	3	3	Lago de Nicaragua (Central America)
1044	2	3	3	Lagoa dos Patos (South America)
1045	2	3	3	Lake Albert (Africa)
1046	2	3	3	Lake Athabasca (North America)
1047	2	3	3	Lake Baikal (Eurasia)
1048	2	3	3	Lake Balkhash (Eurasia)
1049	2	3	3	Lake Chad (Africa)
1050	2	3	3	Lake Champlain (North America)
1051	2	3	3	Lake Erie (North America)
1052	2	3	3	Lake George (North America)
1053	2	3	3	Lake Kariba (Africa)
1054	2	3	3	Lake Malawi (Africa)
1055	2	3	3	Lake Manitoba (North America)
1056	2	3	3	Lake Maurepas (North America)
1057	2	3	3	Lake Mead (North America)
1058	2	3	3	Lake Okeechobee (North America)
1059	2	3	3	Lake Ontario (North America)
1060	2	3	3	Lake Pontchartrain (North America)
1061	2	3	3	Lake Saint Clair (North America)
1062	2	3	3	Lake Seminole (North America)
1063	2	3	3	Lake Superior (North America)
1064	2	3	3	Lake Tahoe (North America)
1065	2	3	3	Lake Tanganyika (Africa)
1066	2	3	3	Lake Texoma (North America)
1067	2	3	3	Lake Titicaca (South America)
1068	2	3	3	Lake Turkana (Africa)
1069	2	3	3	Lake Victoria (Africa)
1070	2	3	3	Lake Volta (Africa)
1071	2	3	3	Lake Winnipeg (North America)
1072	2	3	3	Lake Winnipegosis (North America)
1073	2	3	3	Lake of the Woods (North America)
1074	2	3	3	Lakes Michigan and Huron (North America)
1075	2	3	3	Namakan Lake (North America)
1076	2	1	1	Ocean
1077	2	3	3	Onazhskoye Ozero (Eurasia)
1078	2	3	3	Ozero Ladozhskoye or Lake Ladoga (Eurasia)
1079	2	3	3	Rainy Lake (North America)
1080	2	3	3	Reindeer Lake (North America)
1081	2	3	3	Sabine Lake (North America)
1082	2	3	3	Sault Sainte Marie (North America)
1083	2	3	3	Sea of Marmara (Eurasia)
1084	2	3	3	Shoal Lake (North America)
1085	2	3	3	White Lake (North America)

Table of Full Names

The following table lists the fully qualified name of each entity in the **MapPlot** Ncarg4_1 database. This listing makes much more apparent the logic in the ordering of the names in the database. The colon character (':') separates a parent from a child at the next level (i.e. the child's area type is equal to the parent's area type plus one). The period character ('.') separates a parent from a child at the same level.

ELEMENT NUMBER	FULLY QUALIFIED NAME
1	Land
2	Land : Africa
3	Land : Africa : (Morocco)
4	Land : Africa : (Morocco) . Western Sahara
5	Land : Africa : Algeria
6	Land : Africa : Angola
7	Land : Africa : Angola . exclave called Cabinda
8	Land : Africa : Benin
9	Land : Africa : Botswana
10	Land : Africa : Burkina Faso
11	Land : Africa : Burundi
12	Land : Africa : Cameroon
13	Land : Africa : Central African Republic
14	Land : Africa : Chad
15	Land : Africa : Congo
16	Land : Africa : Djibouti
17	Land : Africa : Egypt
18	Land : Africa : Equatorial Guinea
19	Land : Africa : Equatorial Guinea . Bioko
20	Land : Africa : Equatorial Guinea . Rio Muni
21	Land : Africa : Ethiopia
22	Land : Africa : Ethiopia . Dahlak Archipelago
23	Land : Africa : Ethiopia . Dahlak Archipelago . Dehalak
24	Land : Africa : Ethiopia . Dahlak Archipelago . Nora Island
25	Land : Africa : Gabon
26	Land : Africa : Ghana
27	Land : Africa : Guinea
28	Land : Africa : Guinea-Bissau
29	Land : Africa : Guinea-Bissau . Arquipelago dos Bijagos
30	Land : Africa : Guinea-Bissau . Arquipelago dos Bijagos . Isla de Orango
31	Land : Africa : Ivory Coast
32	Land : Africa : Kenya
33	Land : Africa : Lesotho
34	Land : Africa : Liberia
35	Land : Africa : Libya
36	Land : Africa : Malawi
37	Land : Africa : Mali
38	Land : Africa : Mauritania
39	Land : Africa : Morocco
40	Land : Africa : Mozambique
41	Land : Africa : Namibia
42	Land : Africa : Niger
43	Land : Africa : Nigeria
44	Land : Africa : Rwanda
45	Land : Africa : Senegal
46	Land : Africa : Sierra Leone
47	Land : Africa : Sierra Leone . Sherbro Island
48	Land : Africa : Somalia
49	Land : Africa : South Africa
50	Land : Africa : Sudan

51 Land : Africa : Swaziland
52 Land : Africa : Tanzania
53 Land : Africa : Tanzania . Pemba Island
54 Land : Africa : Tanzania . Zanzibar Island
55 Land : Africa : The Gambia
56 Land : Africa : Togo
57 Land : Africa : Tunisia
58 Land : Africa : Tunisia . Jerba Island
59 Land : Africa : Uganda
60 Land : Africa : Zaire
61 Land : Africa : Zambia
62 Land : Africa : Zimbabwe
63 Land : Antarctica
64 Land : Antarctica . Adelaide Island
65 Land : Antarctica . Anvers Island
66 Land : Antarctica . Bear Island
67 Land : Antarctica . Berkner Island
68 Land : Antarctica . Brabant Island
69 Land : Antarctica . Burke Island
70 Land : Antarctica . Charcot Island
71 Land : Antarctica . Coronation Island
72 Land : Antarctica . Coulman Island
73 Land : Antarctica . Elephant Island
74 Land : Antarctica . James Ross Island
75 Land : Antarctica . Joinville Island
76 Land : Antarctica . King George Island
77 Land : Antarctica . Livingston Island
78 Land : Antarctica . Newman Island
79 Land : Antarctica . Peter I Island
80 Land : Antarctica . Renaud Island
81 Land : Antarctica . Snow Hill Island
82 Land : Antarctica . unidentified islands
83 Land : Atlantic islands
84 Land : Atlantic islands : (Denmark)
85 Land : Atlantic islands : (Denmark) . Faeroe Islands
86 Land : Atlantic islands : (Denmark) . Faeroe Islands . Eysturoy
87 Land : Atlantic islands : (Denmark) . Faeroe Islands . Nordoyar
88 Land : Atlantic islands : (Denmark) . Faeroe Islands . Sandoy
89 Land : Atlantic islands : (Denmark) . Faeroe Islands . Streymoy
90 Land : Atlantic islands : (Denmark) . Faeroe Islands . Suduroy
91 Land : Atlantic islands : (Denmark) . Faeroe Islands . Vagar
92 Land : Atlantic islands : (Equatorial Guinea)
93 Land : Atlantic islands : (Equatorial Guinea) . Annobon
94 Land : Atlantic islands : (Norway)
95 Land : Atlantic islands : (Norway) . Byornoya
96 Land : Atlantic islands : (Norway) . Jan Mayen
97 Land : Atlantic islands : (Norway) . Svalbard
98 Land : Atlantic islands : (Norway) . Svalbard . Barentsoya
99 Land : Atlantic islands : (Norway) . Svalbard . Edgeoya
100 Land : Atlantic islands : (Norway) . Svalbard . Kong Karls Land
101 Land : Atlantic islands : (Norway) . Svalbard . Kong Karls Land . Kongsoya
102 Land : Atlantic islands : (Norway) . Svalbard . Kong Karls Land . Svenskoya
103 Land : Atlantic islands : (Norway) . Svalbard . Kvitoya
104 Land : Atlantic islands : (Norway) . Svalbard . Nordauslandet
105 Land : Atlantic islands : (Norway) . Svalbard . Prins Karls Forland
106 Land : Atlantic islands : (Norway) . Svalbard . Spitsbergen
107 Land : Atlantic islands : (Portugal)
108 Land : Atlantic islands : (Portugal) . Azores
109 Land : Atlantic islands : (Portugal) . Azores . Faial
110 Land : Atlantic islands : (Portugal) . Azores . Flores
111 Land : Atlantic islands : (Portugal) . Azores . Pico

112 Land : Atlantic islands : (Portugal) . Azores . Santa Maria
113 Land : Atlantic islands : (Portugal) . Azores . Sao Jorge
114 Land : Atlantic islands : (Portugal) . Azores . Sao Miguel
115 Land : Atlantic islands : (Portugal) . Azores . Terceira
116 Land : Atlantic islands : (Portugal) . Madeira Islands
117 Land : Atlantic islands : (Portugal) . Madeira Islands . Madeira
118 Land : Atlantic islands : (Portugal) . Madeira Islands . Porto Santo
119 Land : Atlantic islands : (Spain)
120 Land : Atlantic islands : (Spain) . Canary Islands
121 Land : Atlantic islands : (Spain) . Canary Islands . Fuerteventura
122 Land : Atlantic islands : (Spain) . Canary Islands . Gran Canaria
123 Land : Atlantic islands : (Spain) . Canary Islands . Lanzarote
124 Land : Atlantic islands : (Spain) . Canary Islands . Tenerife
125 Land : Atlantic islands : (United Kingdom)
126 Land : Atlantic islands : (United Kingdom) . Bermuda Islands
127 Land : Atlantic islands : (United Kingdom) . Falkland Islands
128 Land : Atlantic islands : (United Kingdom) . Falkland Islands . East Falkla
129 Land : Atlantic islands : (United Kingdom) . Falkland Islands . Weddell Isl
130 Land : Atlantic islands : (United Kingdom) . Falkland Islands . West Falkla
131 Land : Atlantic islands : (United Kingdom) . South Georgia Island
132 Land : Atlantic islands : (United Kingdom) . South Sandwich Island
133 Land : Atlantic islands : Cape Verde
134 Land : Atlantic islands : Cape Verde . Boa Vista
135 Land : Atlantic islands : Cape Verde . Fogo
136 Land : Atlantic islands : Cape Verde . Sal
137 Land : Atlantic islands : Cape Verde . Santo Antao
138 Land : Atlantic islands : Cape Verde . Sao Nicolau
139 Land : Atlantic islands : Cape Verde . Sao Tiago
140 Land : Atlantic islands : Sao Tome and Principe
141 Land : Atlantic islands : Sao Tome and Principe . Principe
142 Land : Atlantic islands : Sao Tome and Principe . Sao Tome
143 Land : Australia
144 Land : Australia : Australia
145 Land : Australia : Australia . Bathurst Island
146 Land : Australia : Australia . Cape Barren Island
147 Land : Australia : Australia . Flinders Island
148 Land : Australia : Australia . Fraser Island
149 Land : Australia : Australia . Groote Eylandt
150 Land : Australia : Australia . Kangaroo Island
151 Land : Australia : Australia . King Island
152 Land : Australia : Australia . Marchinbar Island
153 Land : Australia : Australia . Melville Island
154 Land : Australia : Australia . Mornington Island
155 Land : Australia : Australia . North Stradbroke Island
156 Land : Australia : Australia . Tasmania
157 Land : Caribbean islands
158 Land : Caribbean islands . Hispaniola
159 Land : Caribbean islands . Hispaniola : Dominican Republic
160 Land : Caribbean islands . Hispaniola : Haiti
161 Land : Caribbean islands . Hispaniola : Haiti . Ile de la Gonave
162 Land : Caribbean islands . Hispaniola : Haiti . Ile de la Tortue
163 Land : Caribbean islands : (France)
164 Land : Caribbean islands : (France) . Guadeloupe
165 Land : Caribbean islands : (France) . Martinique
166 Land : Caribbean islands : (Netherlands)
167 Land : Caribbean islands : (Netherlands) . Bonaire
168 Land : Caribbean islands : (Netherlands) . Curacao
169 Land : Caribbean islands : (United Kingdom)
170 Land : Caribbean islands : (United Kingdom) . Turks and Caicos Islands
171 Land : Caribbean islands : (United States)
172 Land : Caribbean islands : (United States) . Puerto Rico

173 Land : Caribbean islands : (United States) . Saint Croix
174 Land : Caribbean islands : (United States) . Saint Thomas
175 Land : Caribbean islands : Antigua and Barbuda
176 Land : Caribbean islands : Antigua and Barbuda . Antigua
177 Land : Caribbean islands : Bahamas
178 Land : Caribbean islands : Bahamas . Abaco Island
179 Land : Caribbean islands : Bahamas . Acklins and Crooked Islands
180 Land : Caribbean islands : Bahamas . Andros Island
181 Land : Caribbean islands : Bahamas . Cat Island
182 Land : Caribbean islands : Bahamas . Eleuthera
183 Land : Caribbean islands : Bahamas . Grand Bahama Island
184 Land : Caribbean islands : Bahamas . Grand Inagua Island
185 Land : Caribbean islands : Bahamas . Long Island
186 Land : Caribbean islands : Bahamas . Mayaguana Island
187 Land : Caribbean islands : Bahamas . Moore’s Island
188 Land : Caribbean islands : Bahamas . New Providence
189 Land : Caribbean islands : Bahamas . San Salvador
190 Land : Caribbean islands : Cuba
191 Land : Caribbean islands : Cuba . Archipielago de Camaguey
192 Land : Caribbean islands : Cuba . Isla de Pinos
193 Land : Caribbean islands : Dominica
194 Land : Caribbean islands : Grenada
195 Land : Caribbean islands : Jamaica
196 Land : Caribbean islands : Saint Kitts and Nevis
197 Land : Caribbean islands : Saint Kitts and Nevis . Saint Kitts
198 Land : Caribbean islands : Saint Lucia
199 Land : Caribbean islands : Saint Vincent and the Grenadines
200 Land : Caribbean islands : Saint Vincent and the Grenadines . Saint Vincent
201 Land : Caribbean islands : Trinidad and Tobago
202 Land : Caribbean islands : Trinidad and Tobago . Tobago
203 Land : Caribbean islands : Trinidad and Tobago . Trinidad
204 Land : Central America
205 Land : Central America : Belize
206 Land : Central America : Costa Rica
207 Land : Central America : El Salvador
208 Land : Central America : Guatemala
209 Land : Central America : Honduras
210 Land : Central America : Nicaragua
211 Land : Central America : Panama
212 Land : Central America : Panama . Isla de Coiba
213 Land : Eurasia
214 Land : Eurasia : Afghanistan
215 Land : Eurasia : Albania
216 Land : Eurasia : Andorra
217 Land : Eurasia : Austria
218 Land : Eurasia : Bahrain
219 Land : Eurasia : Bangladesh
220 Land : Eurasia : Bangladesh . Shabbaz Island
221 Land : Eurasia : Belarus
222 Land : Eurasia : Belgium
223 Land : Eurasia : Bhutan
224 Land : Eurasia : Bosnia and Herzegovina
225 Land : Eurasia : Bulgaria
226 Land : Eurasia : Cambodia
227 Land : Eurasia : China
228 Land : Eurasia : China . Hainan Dao
229 Land : Eurasia : China . Hong Kong Island
230 Land : Eurasia : China . Lantau Island
231 Land : Eurasia : Croatia
232 Land : Eurasia : Croatia . Brac
233 Land : Eurasia : Croatia . Cres

234	Land : Eurasia : Croatia . Korcula
235	Land : Eurasia : Croatia . Krk
236	Land : Eurasia : Cyprus
237	Land : Eurasia : Czech Republic
238	Land : Eurasia : Denmark
239	Land : Eurasia : Denmark . Bornholm
240	Land : Eurasia : Denmark . Falster
241	Land : Eurasia : Denmark . Fyn
242	Land : Eurasia : Denmark . Jutland
243	Land : Eurasia : Denmark . Langeland
244	Land : Eurasia : Denmark . Lolland
245	Land : Eurasia : Denmark . Sjaelland
246	Land : Eurasia : Disputed area between India and China
247	Land : Eurasia : Estonia
248	Land : Eurasia : Finland
249	Land : Eurasia : Finland . Aland
250	Land : Eurasia : Finland . unidentified islands
251	Land : Eurasia : France
252	Land : Eurasia : France . Corsica
253	Land : Eurasia : Germany
254	Land : Eurasia : Germany . Rugen
255	Land : Eurasia : Greece
256	Land : Eurasia : Greece . Andros
257	Land : Eurasia : Greece . Crete
258	Land : Eurasia : Greece . Evvoia
259	Land : Eurasia : Greece . Kefallinia
260	Land : Eurasia : Greece . Kerkira
261	Land : Eurasia : Greece . Khios
262	Land : Eurasia : Greece . Lesvos
263	Land : Eurasia : Greece . Limnos
264	Land : Eurasia : Greece . Naxos
265	Land : Eurasia : Greece . Rhodes
266	Land : Eurasia : Greece . Thasos
267	Land : Eurasia : Hungary
268	Land : Eurasia : India
269	Land : Eurasia : Iran
270	Land : Eurasia : Iran . Qeshm
271	Land : Eurasia : Iraq
272	Land : Eurasia : Ireland
273	Land : Eurasia : Israel
274	Land : Eurasia : Italy
275	Land : Eurasia : Italy . Sardinia
276	Land : Eurasia : Italy . Sicily
277	Land : Eurasia : Japan
278	Land : Eurasia : Japan . Amakusa Shoto
279	Land : Eurasia : Japan . Amami O Shima
280	Land : Eurasia : Japan . Awaji-shima
281	Land : Eurasia : Japan . Goto Retto
282	Land : Eurasia : Japan . Goto Retto . Fukue Jima
283	Land : Eurasia : Japan . Goto Retto . Nakadori Jima
284	Land : Eurasia : Japan . Hokkaido
285	Land : Eurasia : Japan . Honshu
286	Land : Eurasia : Japan . Iki
287	Land : Eurasia : Japan . Iriomote Jima
288	Land : Eurasia : Japan . Ishigaki Shima
289	Land : Eurasia : Japan . Kamino Shima
290	Land : Eurasia : Japan . Kyushu
291	Land : Eurasia : Japan . Miyako Jima
292	Land : Eurasia : Japan . Okinawa
293	Land : Eurasia : Japan . Osumi Shoto
294	Land : Eurasia : Japan . Osumi Shoto . Tanega Shima

295 Land : Eurasia : Japan . Osumi Shoto . Yaku Shima
296 Land : Eurasia : Japan . Rishiri-to
297 Land : Eurasia : Japan . Sado
298 Land : Eurasia : Japan . Shikoku
299 Land : Eurasia : Japan . Shimono Shima
300 Land : Eurasia : Japan . Tokuno Shima
301 Land : Eurasia : Jordan
302 Land : Eurasia : Kuwait
303 Land : Eurasia : Laos
304 Land : Eurasia : Latvia
305 Land : Eurasia : Lebanon
306 Land : Eurasia : Liechtenstein
307 Land : Eurasia : Lithuania
308 Land : Eurasia : Luxembourg
309 Land : Eurasia : Macedonia
310 Land : Eurasia : Malaysia
311 Land : Eurasia : Malaysia . Malay Peninsula
312 Land : Eurasia : Malta
313 Land : Eurasia : Moldova
314 Land : Eurasia : Mongolia
315 Land : Eurasia : Myanmar
316 Land : Eurasia : Myanmar . Kadan Kyun
317 Land : Eurasia : Myanmar . Kanmaw Kyun
318 Land : Eurasia : Myanmar . Letsok-aw Kyun
319 Land : Eurasia : Nepal
320 Land : Eurasia : Netherlands
321 Land : Eurasia : Netherlands . Texel
322 Land : Eurasia : Netherlands . unidentified islands
323 Land : Eurasia : North Korea
324 Land : Eurasia : Norway
325 Land : Eurasia : Norway . Arnoy
326 Land : Eurasia : Norway . Hamaroy
327 Land : Eurasia : Norway . Hitra
328 Land : Eurasia : Norway . Kvaloy - 1
329 Land : Eurasia : Norway . Kvaloy - 2
330 Land : Eurasia : Norway . Lofoten Vesteralen
331 Land : Eurasia : Norway . Lofoten Vesteralen . Andoya
332 Land : Eurasia : Norway . Lofoten Vesteralen . Hinnoya
333 Land : Eurasia : Norway . Lofoten Vesteralen . Langoya
334 Land : Eurasia : Norway . Mageroya
335 Land : Eurasia : Norway . Ringvassoy
336 Land : Eurasia : Norway . Seiland
337 Land : Eurasia : Norway . Senja
338 Land : Eurasia : Norway . Soroya
339 Land : Eurasia : Norway . Stjernoy
340 Land : Eurasia : Norway . Vannoy
341 Land : Eurasia : Oman
342 Land : Eurasia : Oman . Jazirat Masirah
343 Land : Eurasia : Pakistan
344 Land : Eurasia : Poland
345 Land : Eurasia : Portugal
346 Land : Eurasia : Qatar
347 Land : Eurasia : Romania
348 Land : Eurasia : Russia
349 Land : Eurasia : Russia . Franz Josef Land
350 Land : Eurasia : Russia . Franz Josef Land . Ostrov Champ
351 Land : Eurasia : Russia . Franz Josef Land . Ostrov Dzheksona
352 Land : Eurasia : Russia . Franz Josef Land . Ostrov Gallya
353 Land : Eurasia : Russia . Franz Josef Land . Ostrov Greem Bell
354 Land : Eurasia : Russia . Franz Josef Land . Ostrov Gukera
355 Land : Eurasia : Russia . Franz Josef Land . Ostrov Karla-Aleksandra

356 Land : Eurasia : Russia . Franz Josef Land . Ostrov La-Rons’yer
357 Land : Eurasia : Russia . Franz Josef Land . Ostrov Luidzhi
358 Land : Eurasia : Russia . Franz Josef Land . Ostrov Mak-Klintoka
359 Land : Eurasia : Russia . Franz Josef Land . Ostrov Rudol’fa
360 Land : Eurasia : Russia . Franz Josef Land . Ostrov Sal’m
361 Land : Eurasia : Russia . Franz Josef Land . Ostrov Solsberi
362 Land : Eurasia : Russia . Franz Josef Land . Ostrov Tsiglera
363 Land : Eurasia : Russia . Franz Josef Land . Ostrov Viner-Neyshtadt
364 Land : Eurasia : Russia . Franz Josef Land . Ostrova Belaya Zemlya
365 Land : Eurasia : Russia . Franz Josef Land . Zemlya Alexandra
366 Land : Eurasia : Russia . Franz Josef Land . Zemlya George
367 Land : Eurasia : Russia . Franz Josef Land . Zemlya Vil’cheka
368 Land : Eurasia : Russia . Kaliningrad Oblast
369 Land : Eurasia : Russia . Novaya Zemlya
370 Land : Eurasia : Russia . Ostrov Ayon
371 Land : Eurasia : Russia . Ostrov Belkov’skiy
372 Land : Eurasia : Russia . Ostrov Belyy
373 Land : Eurasia : Russia . Ostrov Bering
374 Land : Eurasia : Russia . Ostrov Bol’shoy Begichev
375 Land : Eurasia : Russia . Ostrov Bol’shoy Lyakhovskiy
376 Land : Eurasia : Russia . Ostrov Bol’shoy Shantar
377 Land : Eurasia : Russia . Ostrov Faddeyevskiy
378 Land : Eurasia : Russia . Ostrov Hiiumaa
379 Land : Eurasia : Russia . Ostrov Isachenko
380 Land : Eurasia : Russia . Ostrov Iturup
381 Land : Eurasia : Russia . Ostrov Karaginskiy
382 Land : Eurasia : Russia . Ostrov Kolguyev
383 Land : Eurasia : Russia . Ostrov Kotel’nyy
384 Land : Eurasia : Russia . Ostrov Kunashir
385 Land : Eurasia : Russia . Ostrov Malyy Lyakhovskiy
386 Land : Eurasia : Russia . Ostrov Mednyy
387 Land : Eurasia : Russia . Ostrov Mezhdusharskiy
388 Land : Eurasia : Russia . Ostrov Novaya Sibir’
389 Land : Eurasia : Russia . Ostrov Ol’khon (Lake Baikal)
390 Land : Eurasia : Russia . Ostrov Oleniy
391 Land : Eurasia : Russia . Ostrov Onekotan
392 Land : Eurasia : Russia . Ostrov Paramushir
393 Land : Eurasia : Russia . Ostrov Russkiy
394 Land : Eurasia : Russia . Ostrov Saaremaa
395 Land : Eurasia : Russia . Ostrov Sakhalin
396 Land : Eurasia : Russia . Ostrov Shiashkotan
397 Land : Eurasia : Russia . Ostrov Shmidta
398 Land : Eurasia : Russia . Ostrov Shokal’skogo
399 Land : Eurasia : Russia . Ostrov Shumshu
400 Land : Eurasia : Russia . Ostrov Sibiryakova
401 Land : Eurasia : Russia . Ostrov Simushir
402 Land : Eurasia : Russia . Ostrov Stolbovoy
403 Land : Eurasia : Russia . Ostrov Taymyr
404 Land : Eurasia : Russia . Ostrov Urup
405 Land : Eurasia : Russia . Ostrov Ushakova
406 Land : Eurasia : Russia . Ostrov Vaygach
407 Land : Eurasia : Russia . Ostrov Yarok
408 Land : Eurasia : Russia . Ostrova Arkticheskogo Instituta
409 Land : Eurasia : Russia . Severnaya Zemlya
410 Land : Eurasia : Russia . Severnaya Zemlya . Ostrov Bol’shevik
411 Land : Eurasia : Russia . Severnaya Zemlya . Ostrov Komsomolets
412 Land : Eurasia : Russia . Severnaya Zemlya . Ostrov Malyy Taymyr
413 Land : Eurasia : Russia . Severnaya Zemlya . Ostrov Oktyabr’skoy Revolyutsi
414 Land : Eurasia : Russia . Severnaya Zemlya . Ostrov Pioner
415 Land : Eurasia : Russia . Shikotan
416 Land : Eurasia : Russia . Wrangel Island

417 Land : Eurasia : Russia . unidentified islands
418 Land : Eurasia : Saudi Arabia
419 Land : Eurasia : Saudi Arabia . Jaza’ir Farasan
420 Land : Eurasia : Singapore
421 Land : Eurasia : Slovakia
422 Land : Eurasia : Slovenia
423 Land : Eurasia : South Korea
424 Land : Eurasia : South Korea . Cheju Do
425 Land : Eurasia : South Korea . Ullung Do
426 Land : Eurasia : Spain
427 Land : Eurasia : Spain . Ibiza
428 Land : Eurasia : Spain . Mallorca
429 Land : Eurasia : Spain . Menorca
430 Land : Eurasia : Sri Lanka (Ceylon)
431 Land : Eurasia : Sweden
432 Land : Eurasia : Sweden . Gotland
433 Land : Eurasia : Sweden . Oland
434 Land : Eurasia : Switzerland
435 Land : Eurasia : Syria
436 Land : Eurasia : Taiwan
437 Land : Eurasia : Thailand
438 Land : Eurasia : Thailand . Ko Phuket
439 Land : Eurasia : Turkey
440 Land : Eurasia : Ukraine
441 Land : Eurasia : United Arab Emirates
442 Land : Eurasia : United Arab Emirates . Abu al Abyad
443 Land : Eurasia : United Kingdom
444 Land : Eurasia : United Kingdom . England
445 Land : Eurasia : United Kingdom . England . Isle of Wight
446 Land : Eurasia : United Kingdom . Isle of Man
447 Land : Eurasia : United Kingdom . Northern Ireland
448 Land : Eurasia : United Kingdom . Scotland
449 Land : Eurasia : United Kingdom . Scotland . Inner Hebrides
450 Land : Eurasia : United Kingdom . Scotland . Inner Hebrides . Island of Mul
451 Land : Eurasia : United Kingdom . Scotland . Inner Hebrides . Island of Sky
452 Land : Eurasia : United Kingdom . Scotland . Inner Hebrides . Islay
453 Land : Eurasia : United Kingdom . Scotland . Inner Hebrides . Jura
454 Land : Eurasia : United Kingdom . Scotland . Island of Arran
455 Land : Eurasia : United Kingdom . Scotland . Orkney Islands
456 Land : Eurasia : United Kingdom . Scotland . Orkney Islands . Dwarfie Stane
457 Land : Eurasia : United Kingdom . Scotland . Orkney Islands . Mainland
458 Land : Eurasia : United Kingdom . Scotland . Outer Hebrides
459 Land : Eurasia : United Kingdom . Scotland . Outer Hebrides . Isle of Lewis
460 Land : Eurasia : United Kingdom . Scotland . Outer Hebrides . North Uist
461 Land : Eurasia : United Kingdom . Scotland . Shetland Islands
462 Land : Eurasia : United Kingdom . Wales
463 Land : Eurasia : United Kingdom . Wales . Anglesey
464 Land : Eurasia : Vietnam
465 Land : Eurasia : Vietnam . Dao Phu Quoc
466 Land : Eurasia : Yemen
467 Land : Eurasia : Yemen : Aden
468 Land : Eurasia : Yemen : Aden . Abd-Al-Kuri
469 Land : Eurasia : Yemen : Aden . Socotra
470 Land : Eurasia : Yemen : Sana
471 Land : Eurasia : Yemen : Sana . Az Zukur
472 Land : Eurasia : Yemen : Sana . Hanish Al Kabir
473 Land : Eurasia : Yemen : Sana . Kamaran
474 Land : Eurasia : Yugoslavia
475 Land : Greenland
476 Land : Greenland : (Denmark)
477 Land : Greenland : (Denmark) . Greenland

478 Land : Greenland : (Denmark) . Greenland . Clavering Oy
479 Land : Greenland : (Denmark) . Greenland . Geographical Society Oy
480 Land : Greenland : (Denmark) . Greenland . Hovgaard Oy
481 Land : Greenland : (Denmark) . Greenland . Ile de France
482 Land : Greenland : (Denmark) . Greenland . Milne Land
483 Land : Greenland : (Denmark) . Greenland . Norske Oer
484 Land : Greenland : (Denmark) . Greenland . Qeqertarsuaq
485 Land : Greenland : (Denmark) . Greenland . Shannon
486 Land : Greenland : (Denmark) . Greenland . Store Koldewey
487 Land : Greenland : (Denmark) . Greenland . Traill Oy
488 Land : Greenland : (Denmark) . Greenland . Ymer Oy
489 Land : Greenland : (Denmark) . Greenland . unidentified islands
490 Land : Iceland
491 Land : Iceland : Iceland
492 Land : Indian Ocean islands
493 Land : Indian Ocean islands : (France)
494 Land : Indian Ocean islands : (France) . Ile Kerguelen
495 Land : Indian Ocean islands : (France) . Mayotte (Dzaoudzi)
496 Land : Indian Ocean islands : (France) . Reunion Island
497 Land : Indian Ocean islands : (India)
498 Land : Indian Ocean islands : (India) . Andaman Islands
499 Land : Indian Ocean islands : (India) . Nicobar Islands
500 Land : Indian Ocean islands : (India) . Nicobar Islands . Great Nicobar
501 Land : Indian Ocean islands : (South Africa)
502 Land : Indian Ocean islands : (South Africa) . Prince Edward Islands
503 Land : Indian Ocean islands : (South Africa) . Prince Edward Islands . Mari
504 Land : Indian Ocean islands : Comoros
505 Land : Indian Ocean islands : Comoros . Mwali
506 Land : Indian Ocean islands : Comoros . Njazidja
507 Land : Indian Ocean islands : Comoros . Nzwani
508 Land : Indian Ocean islands : Madagascar
509 Land : Indian Ocean islands : Maldives
510 Land : Indian Ocean islands : Mauritius
511 Land : Indian Ocean islands : Seychelles
512 Land : Indian Ocean islands : Seychelles . Aldabra Island
513 Land : Indian Ocean islands : Seychelles . Mahe Island
514 Land : North America
515 Land : North America : (France)
516 Land : North America : (France) . Saint Pierre and Miquelon
517 Land : North America : (France) . Saint Pierre and Miquelon . Miquelon
518 Land : North America : Canada
519 Land : North America : Canada . Air Force Island
520 Land : North America : Canada . Akimiski Island
521 Land : North America : Canada . Akpatok Island
522 Land : North America : Canada . Amherst Island (Lake Ontario)
523 Land : North America : Canada . Amund Ringnes Island
524 Land : North America : Canada . Anticosti Island
525 Land : North America : Canada . Axel Heiberg Island
526 Land : North America : Canada . Baffin Island
527 Land : North America : Canada . Banks Island - 1
528 Land : North America : Canada . Banks Island - 2
529 Land : North America : Canada . Barrie Island (Lake Huron)
530 Land : North America : Canada . Batchawana Island (Lake Superior)
531 Land : North America : Canada . Bathurst Island
532 Land : North America : Canada . Belcher Islands
533 Land : North America : Canada . Big Island
534 Land : North America : Canada . Big Island (Great Slave Lake)
535 Land : North America : Canada . Borden Island
536 Land : North America : Canada . Brock Island
537 Land : North America : Canada . Byam Martin Island
538 Land : North America : Canada . Bylot Island

539 Land : North America : Canada . Cape Breton Island
540 Land : North America : Canada . Charles Island
541 Land : North America : Canada . Christian Island (Lake Huron)
542 Land : North America : Canada . Coats Island
543 Land : North America : Canada . Coburg Island
544 Land : North America : Canada . Cockburn Island (Lake Huron)
545 Land : North America : Canada . Cornwall Island
546 Land : North America : Canada . Cornwallis Island
547 Land : North America : Canada . Devon Island
548 Land : North America : Canada . Eglinton Island
549 Land : North America : Canada . Ellef Ringnes Island
550 Land : North America : Canada . Ellesmere Island
551 Land : North America : Canada . Emerald Isle
552 Land : North America : Canada . Fitzwilliam Island (Lake Huron)
553 Land : North America : Canada . Island of Newfoundland
554 Land : North America : Canada . Isle Royale (Lake Superior)
555 Land : North America : Canada . Jens Munk Island
556 Land : North America : Canada . King Christian Island
557 Land : North America : Canada . King William Island
558 Land : North America : Canada . Lougheed Island
559 Land : North America : Canada . Mackenzie King Island
560 Land : North America : Canada . Manitoulin Island (Lake Huron)
561 Land : North America : Canada . Mansel Island
562 Land : North America : Canada . Meighen Island
563 Land : North America : Canada . Melville Island
564 Land : North America : Canada . Michipicoten Island (Lake Superior)
565 Land : North America : Canada . Nottingham Island
566 Land : North America : Canada . Philpots Island
567 Land : North America : Canada . Pie Island (Lake Superior)
568 Land : North America : Canada . Pitt Island
569 Land : North America : Canada . Porcher Island
570 Land : North America : Canada . Prescott Island
571 Land : North America : Canada . Prince Charles Island
572 Land : North America : Canada . Prince Edward Island
573 Land : North America : Canada . Prince Patrick Island
574 Land : North America : Canada . Prince of Wales Island
575 Land : North America : Canada . Queen Charlotte Islands
576 Land : North America : Canada . Queen Charlotte Islands . Graham Island
577 Land : North America : Canada . Queen Charlotte Islands . Moresby Island
578 Land : North America : Canada . Resolution Island
579 Land : North America : Canada . Richards Island
580 Land : North America : Canada . Rowley Island
581 Land : North America : Canada . Russell Island
582 Land : North America : Canada . Salisbury Island
583 Land : North America : Canada . Simpson Island (Great Slave Lake)
584 Land : North America : Canada . Somerset Island
585 Land : North America : Canada . Southampton Island
586 Land : North America : Canada . Stefansson Island
587 Land : North America : Canada . Vancouver Island
588 Land : North America : Canada . Vansittart Island
589 Land : North America : Canada . Victoria Island
590 Land : North America : Canada . Wales Island
591 Land : North America : Canada . Wolfe Island (Lake Ontario)
592 Land : North America : Canada . unidentified island (Great Slave Lake)
593 Land : North America : Canada . unidentified islands
594 Land : North America : Mexico
595 Land : North America : Mexico . Angel de la Guarda
596 Land : North America : Mexico . Isla Cedros
597 Land : North America : Mexico . Isla de Cozumel
598 Land : North America : Mexico . Isla de Guadalupe
599 Land : North America : Mexico . Islas Revillagigedo

600 Land : North America : Mexico . Tiburon
601 Land : North America : United States
602 Land : North America : United States . Conterminous US
603 Land : North America : United States . Conterminous US : Alabama
604 Land : North America : United States . Conterminous US : Arizona
605 Land : North America : United States . Conterminous US : Arkansas
606 Land : North America : United States . Conterminous US : California
607 Land : North America : United States . Conterminous US : California . Chann
608 Land : North America : United States . Conterminous US : California . Chann
609 Land : North America : United States . Conterminous US : California . Chann
610 Land : North America : United States . Conterminous US : California . Chann
611 Land : North America : United States . Conterminous US : California . Chann
612 Land : North America : United States . Conterminous US : California . Chann
613 Land : North America : United States . Conterminous US : California . Chann
614 Land : North America : United States . Conterminous US : Colorado
615 Land : North America : United States . Conterminous US : Connecticut
616 Land : North America : United States . Conterminous US : Delaware
617 Land : North America : United States . Conterminous US : Florida
618 Land : North America : United States . Conterminous US : Florida . Keys
619 Land : North America : United States . Conterminous US : Florida . Pine Isl
620 Land : North America : United States . Conterminous US : Florida . Saint Vi
621 Land : North America : United States . Conterminous US : Florida . Sanibel
622 Land : North America : United States . Conterminous US : Georgia
623 Land : North America : United States . Conterminous US : Idaho
624 Land : North America : United States . Conterminous US : Illinois
625 Land : North America : United States . Conterminous US : Indiana
626 Land : North America : United States . Conterminous US : Iowa
627 Land : North America : United States . Conterminous US : Kansas
628 Land : North America : United States . Conterminous US : Kentucky
629 Land : North America : United States . Conterminous US : Louisiana
630 Land : North America : United States . Conterminous US : Louisiana . Cat Is
631 Land : North America : United States . Conterminous US : Louisiana . Grand
632 Land : North America : United States . Conterminous US : Louisiana . Isle a
633 Land : North America : United States . Conterminous US : Louisiana . Isles
634 Land : North America : United States . Conterminous US : Louisiana . Marsh
635 Land : North America : United States . Conterminous US : Louisiana . Timbal
636 Land : North America : United States . Conterminous US : Louisiana . uniden
637 Land : North America : United States . Conterminous US : Maine
638 Land : North America : United States . Conterminous US : Maine . Deer Isle
639 Land : North America : United States . Conterminous US : Maine . Great Wass
640 Land : North America : United States . Conterminous US : Maine . Head Harbo
641 Land : North America : United States . Conterminous US : Maine . Isle au Ha
642 Land : North America : United States . Conterminous US : Maine . Islesboro
643 Land : North America : United States . Conterminous US : Maine . Mount Dese
644 Land : North America : United States . Conterminous US : Maine . Swans Isla
645 Land : North America : United States . Conterminous US : Maine . Vinalhaven
646 Land : North America : United States . Conterminous US : Maryland
647 Land : North America : United States . Conterminous US : Maryland . Assatea
648 Land : North America : United States . Conterminous US : Maryland . Bloodsw
649 Land : North America : United States . Conterminous US : Maryland . Deal Is
650 Land : North America : United States . Conterminous US : Maryland . Smith I
651 Land : North America : United States . Conterminous US : Maryland . unident
652 Land : North America : United States . Conterminous US : Massachusetts
653 Land : North America : United States . Conterminous US : Massachusetts . Ma
654 Land : North America : United States . Conterminous US : Massachusetts . Na
655 Land : North America : United States . Conterminous US : Michigan
656 Land : North America : United States . Conterminous US : Michigan . Beaver
657 Land : North America : United States . Conterminous US : Michigan . Bois Bl
658 Land : North America : United States . Conterminous US : Michigan . Drummon
659 Land : North America : United States . Conterminous US : Michigan . Grand I
660 Land : North America : United States . Conterminous US : Michigan . Manitou

661	Land : North America : United States . Conterminous US : Minnesota
662	Land : North America : United States . Conterminous US : Minnesota . Big Is
663	Land : North America : United States . Conterminous US : Minnesota . Bigsby
664	Land : North America : United States . Conterminous US : Mississippi
665	Land : North America : United States . Conterminous US : Missouri
666	Land : North America : United States . Conterminous US : Montana
667	Land : North America : United States . Conterminous US : Nebraska
668	Land : North America : United States . Conterminous US : Nevada
669	Land : North America : United States . Conterminous US : New Hampshire
670	Land : North America : United States . Conterminous US : New Jersey
671	Land : North America : United States . Conterminous US : New Mexico
672	Land : North America : United States . Conterminous US : New York
673	Land : North America : United States . Conterminous US : New York . Fishers
674	Land : North America : United States . Conterminous US : New York . Grand I
675	Land : North America : United States . Conterminous US : New York . Long Is
676	Land : North America : United States . Conterminous US : New York . Shelter
677	Land : North America : United States . Conterminous US : New York . Staten
678	Land : North America : United States . Conterminous US : New York . Thousan
679	Land : North America : United States . Conterminous US : North Carolina
680	Land : North America : United States . Conterminous US : North Carolina . C
681	Land : North America : United States . Conterminous US : North Carolina . H
682	Land : North America : United States . Conterminous US : North Carolina . O
683	Land : North America : United States . Conterminous US : North Carolina . R
684	Land : North America : United States . Conterminous US : North Dakota
685	Land : North America : United States . Conterminous US : Ohio
686	Land : North America : United States . Conterminous US : Oklahoma
687	Land : North America : United States . Conterminous US : Oregon
688	Land : North America : United States . Conterminous US : Pennsylvania
689	Land : North America : United States . Conterminous US : Rhode Island
690	Land : North America : United States . Conterminous US : Rhode Island . Blo
691	Land : North America : United States . Conterminous US : Rhode Island . uni
692	Land : North America : United States . Conterminous US : South Carolina
693	Land : North America : United States . Conterminous US : South Dakota
694	Land : North America : United States . Conterminous US : Tennessee
695	Land : North America : United States . Conterminous US : Texas
696	Land : North America : United States . Conterminous US : Texas . Galveston
697	Land : North America : United States . Conterminous US : Texas . Padre and
698	Land : North America : United States . Conterminous US : Utah
699	Land : North America : United States . Conterminous US : Vermont
700	Land : North America : United States . Conterminous US : Virginia
701	Land : North America : United States . Conterminous US : Washington
702	Land : North America : United States . Conterminous US : Washington . San J
703	Land : North America : United States . Conterminous US : West Virginia
704	Land : North America : United States . Conterminous US : Wisconsin
705	Land : North America : United States . Conterminous US : Wisconsin . Madeli
706	Land : North America : United States . Conterminous US : Wisconsin . Oak Is
707	Land : North America : United States . Conterminous US : Wisconsin . Outer
708	Land : North America : United States . Conterminous US : Wisconsin . Stockt
709	Land : North America : United States . Conterminous US : Wisconsin . Washin
710	Land : North America : United States . Conterminous US : Wyoming
711	Land : North America : United States : Alaska
712	Land : North America : United States : Alaska . Adak Island
713	Land : North America : United States : Alaska . Admiralty Island
714	Land : North America : United States : Alaska . Afognak Island
715	Land : North America : United States : Alaska . Akutan Island
716	Land : North America : United States : Alaska . Amchitka Island
717	Land : North America : United States : Alaska . Amlia Island
718	Land : North America : United States : Alaska . Annette Island
719	Land : North America : United States : Alaska . Atka Island
720	Land : North America : United States : Alaska . Attu Island
721	Land : North America : United States : Alaska . Baranof Island

722 Land : North America : United States : Alaska . Chichagof Island
723 Land : North America : United States : Alaska . Dall Island
724 Land : North America : United States : Alaska . Hagemeister Island
725 Land : North America : United States : Alaska . Hinchinbrook Island
726 Land : North America : United States : Alaska . Kanaga Island
727 Land : North America : United States : Alaska . Kiska Island
728 Land : North America : United States : Alaska . Kodiak Island
729 Land : North America : United States : Alaska . Kuiu Island
730 Land : North America : United States : Alaska . Kupreanof Island
731 Land : North America : United States : Alaska . Mitkof Island
732 Land : North America : United States : Alaska . Mitrofania Island
733 Land : North America : United States : Alaska . Montague Island
734 Land : North America : United States : Alaska . Nelson Island
735 Land : North America : United States : Alaska . Nunivak Island
736 Land : North America : United States : Alaska . Pribilof Islands
737 Land : North America : United States : Alaska . Pribilof Islands . Saint Ge
738 Land : North America : United States : Alaska . Pribilof Islands . Saint Pa
739 Land : North America : United States : Alaska . Prince of Wales Island
740 Land : North America : United States : Alaska . Raspberry Island
741 Land : North America : United States : Alaska . Revillagigedo Island
742 Land : North America : United States : Alaska . Saint Lawrence Island
743 Land : North America : United States : Alaska . Saint Matthew Island
744 Land : North America : United States : Alaska . Sanak Island
745 Land : North America : United States : Alaska . Seguam Island
746 Land : North America : United States : Alaska . Semisopochnoi Island
747 Land : North America : United States : Alaska . Shuyak Island
748 Land : North America : United States : Alaska . Sitkalidak Island
749 Land : North America : United States : Alaska . Sitkinak Island
750 Land : North America : United States : Alaska . Swindle Island
751 Land : North America : United States : Alaska . Tanaga Island
752 Land : North America : United States : Alaska . Tugidak Island
753 Land : North America : United States : Alaska . Umnak Island
754 Land : North America : United States : Alaska . Unalaska Island
755 Land : North America : United States : Alaska . Unga Island
756 Land : North America : United States : Alaska . Unimak Island
757 Land : North America : United States : Alaska . Wrangell Island
758 Land : North America : United States : Alaska . Zarembo Island
759 Land : Pacific islands
760 Land : Pacific islands . Borneo
761 Land : Pacific islands . Borneo : Brunei
762 Land : Pacific islands . Borneo : Indonesia
763 Land : Pacific islands . Borneo : Indonesia . Kalimantan
764 Land : Pacific islands . Borneo : Malaysia
765 Land : Pacific islands . Borneo : Malaysia . Malaysia Timor
766 Land : Pacific islands . New Guinea
767 Land : Pacific islands . New Guinea : Indonesia
768 Land : Pacific islands . New Guinea : Indonesia . Irian Jaya
769 Land : Pacific islands . New Guinea : Papua New Guinea
770 Land : Pacific islands : (Australia)
771 Land : Pacific islands : (Australia) . Christmas Island
772 Land : Pacific islands : (Chile)
773 Land : Pacific islands : (Chile) . Easter Island
774 Land : Pacific islands : (Ecuador)
775 Land : Pacific islands : (Ecuador) . Galapagos Islands
776 Land : Pacific islands : (Ecuador) . Galapagos Islands . Isla Fernandina
777 Land : Pacific islands : (Ecuador) . Galapagos Islands . Isla Isabela
778 Land : Pacific islands : (Ecuador) . Galapagos Islands . Isla San Cristobal
779 Land : Pacific islands : (Ecuador) . Galapagos Islands . Isla San Salvador
780 Land : Pacific islands : (Ecuador) . Galapagos Islands . Isla Santa Cruz
781 Land : Pacific islands : (France)
782 Land : Pacific islands : (France) . French Polynesia

783 Land : Pacific islands : (France) . French Polynesia . Marquesas Islands
784 Land : Pacific islands : (France) . French Polynesia . Moorea
785 Land : Pacific islands : (France) . French Polynesia . Raiatea and Tahaa
786 Land : Pacific islands : (France) . French Polynesia . Tahiti
787 Land : Pacific islands : (France) . French Polynesia . Tuamotu Archipelago
788 Land : Pacific islands : (France) . French Polynesia . Tuamotu Archipelago
789 Land : Pacific islands : (France) . French Polynesia . Tuamotu Archipelago
790 Land : Pacific islands : (France) . French Polynesia . Tuamotu Archipelago
791 Land : Pacific islands : (France) . French Polynesia . Tuamotu Archipelago
792 Land : Pacific islands : (France) . French Polynesia . Tuamotu Archipelago
793 Land : Pacific islands : (France) . French Polynesia . Tuamotu Archipelago
794 Land : Pacific islands : (France) . French Polynesia . Tuamotu Archipelago
795 Land : Pacific islands : (France) . French Polynesia . Tuamotu Archipelago
796 Land : Pacific islands : (France) . French Polynesia . Tuamotu Archipelago
797 Land : Pacific islands : (France) . New Caledonia
798 Land : Pacific islands : (France) . New Caledonia . Iles Loyaute (Loyalty I
799 Land : Pacific islands : (France) . New Caledonia . Iles Loyaute (Loyalty I
800 Land : Pacific islands : (France) . New Caledonia . Iles Loyaute (Loyalty I
801 Land : Pacific islands : (Japan)
802 Land : Pacific islands : (Japan) . Minami Tori Shima
803 Land : Pacific islands : (New Zealand)
804 Land : Pacific islands : (New Zealand) . Manihiki
805 Land : Pacific islands : (New Zealand) . Niue
806 Land : Pacific islands : (New Zealand) . Penrhyn
807 Land : Pacific islands : (New Zealand) . Tokelau
808 Land : Pacific islands : (United Kingdom)
809 Land : Pacific islands : (United Kingdom) . Ducie Island
810 Land : Pacific islands : (United Kingdom) . Pitcairn Island
811 Land : Pacific islands : (United States)
812 Land : Pacific islands : (United States) . Baker Island
813 Land : Pacific islands : (United States) . Guam
814 Land : Pacific islands : (United States) . Northern Mariana Islands
815 Land : Pacific islands : (United States) . Northern Mariana Islands . Asunc
816 Land : Pacific islands : (United States) . Northern Mariana Islands . Gugua
817 Land : Pacific islands : (United States) . Northern Mariana Islands . Saipa
818 Land : Pacific islands : (United States) . Palmyra Atoll
819 Land : Pacific islands : (United States) . Wake Island
820 Land : Pacific islands : Fiji
821 Land : Pacific islands : Fiji . Vanua Levu
822 Land : Pacific islands : Fiji . Viti Levu
823 Land : Pacific islands : Indonesia
824 Land : Pacific islands : Indonesia . Alor
825 Land : Pacific islands : Indonesia . Ambon
826 Land : Pacific islands : Indonesia . Babar
827 Land : Pacific islands : Indonesia . Bali
828 Land : Pacific islands : Indonesia . Bangka
829 Land : Pacific islands : Indonesia . Bawean
830 Land : Pacific islands : Indonesia . Belitung
831 Land : Pacific islands : Indonesia . Bengkalis
832 Land : Pacific islands : Indonesia . Biak and Supiori
833 Land : Pacific islands : Indonesia . Bintan
834 Land : Pacific islands : Indonesia . Buru
835 Land : Pacific islands : Indonesia . Buton
836 Land : Pacific islands : Indonesia . Ceram
837 Land : Pacific islands : Indonesia . Damar
838 Land : Pacific islands : Indonesia . Flores
839 Land : Pacific islands : Indonesia . Halmahera
840 Land : Pacific islands : Indonesia . Java
841 Land : Pacific islands : Indonesia . Kabaena
842 Land : Pacific islands : Indonesia . Kai Kecil
843 Land : Pacific islands : Indonesia . Karakelong

844 Land : Pacific islands : Indonesia . Kasiruta
845 Land : Pacific islands : Indonesia . Kepulauan Aru
846 Land : Pacific islands : Indonesia . Kepulauan Aru . Maikoor
847 Land : Pacific islands : Indonesia . Kepulauan Aru . Trangan
848 Land : Pacific islands : Indonesia . Kepulauan Aru . Wokam
849 Land : Pacific islands : Indonesia . Kepulauan Kangean
850 Land : Pacific islands : Indonesia . Kepulauan Tanimbar
851 Land : Pacific islands : Indonesia . Laut
852 Land : Pacific islands : Indonesia . Lomblen
853 Land : Pacific islands : Indonesia . Lombok
854 Land : Pacific islands : Indonesia . Mangole
855 Land : Pacific islands : Indonesia . Misool
856 Land : Pacific islands : Indonesia . Morotai
857 Land : Pacific islands : Indonesia . Muna
858 Land : Pacific islands : Indonesia . Nias
859 Land : Pacific islands : Indonesia . Obi
860 Land : Pacific islands : Indonesia . Pantar
861 Land : Pacific islands : Indonesia . Peleng
862 Land : Pacific islands : Indonesia . Roti
863 Land : Pacific islands : Indonesia . Rupat
864 Land : Pacific islands : Indonesia . Salawati
865 Land : Pacific islands : Indonesia . Sanana
866 Land : Pacific islands : Indonesia . Sangihe
867 Land : Pacific islands : Indonesia . Siberut
868 Land : Pacific islands : Indonesia . Simuelue
869 Land : Pacific islands : Indonesia . Singkep
870 Land : Pacific islands : Indonesia . Sulawesi (Celebes)
871 Land : Pacific islands : Indonesia . Sumatra
872 Land : Pacific islands : Indonesia . Sumba
873 Land : Pacific islands : Indonesia . Sumbawa and Lombok
874 Land : Pacific islands : Indonesia . Taliabu
875 Land : Pacific islands : Indonesia . Tanjung Pemali
876 Land : Pacific islands : Indonesia . Timor
877 Land : Pacific islands : Indonesia . Waigeo
878 Land : Pacific islands : Indonesia . Wetar
879 Land : Pacific islands : Indonesia . Wowoni
880 Land : Pacific islands : Indonesia . Yapen
881 Land : Pacific islands : Kiribati
882 Land : Pacific islands : Kiribati . Banaba (Ocean Island)
883 Land : Pacific islands : Kiribati . Nonouti
884 Land : Pacific islands : Kiribati . Phoenix Islands
885 Land : Pacific islands : Kiribati . Phoenix Islands . Kanton
886 Land : Pacific islands : Kiribati . Phoenix Islands . Rawaki
887 Land : Pacific islands : Marshall Islands
888 Land : Pacific islands : Marshall Islands . Jaluit
889 Land : Pacific islands : Marshall Islands . Kwajalein Atoll
890 Land : Pacific islands : Marshall Islands . Rongelap
891 Land : Pacific islands : Marshall Islands . Wotje
892 Land : Pacific islands : Micronesia
893 Land : Pacific islands : Micronesia . Kosrae
894 Land : Pacific islands : Micronesia . Mortlock Island
895 Land : Pacific islands : Micronesia . Senyavin Island
896 Land : Pacific islands : Micronesia . Truk
897 Land : Pacific islands : Micronesia . Yap
898 Land : Pacific islands : New Zealand
899 Land : Pacific islands : New Zealand . Auckland Islands
900 Land : Pacific islands : New Zealand . Chatham Islands
901 Land : Pacific islands : New Zealand . North Island
902 Land : Pacific islands : New Zealand . South Island
903 Land : Pacific islands : New Zealand . Stewart Island
904 Land : Pacific islands : Palau

905 Land : Pacific islands : Papua New Guinea

906 Land : Pacific islands : Papua New Guinea . Bismarck Archipelago

907 Land : Pacific islands : Papua New Guinea . Bismarck Archipelago . Long Isl

908 Land : Pacific islands : Papua New Guinea . Bismarck Archipelago . Manus

909 Land : Pacific islands : Papua New Guinea . Bismarck Archipelago . Mussau

910 Land : Pacific islands : Papua New Guinea . Bismarck Archipelago . New Brit

911 Land : Pacific islands : Papua New Guinea . Bismarck Archipelago . New Hano

912 Land : Pacific islands : Papua New Guinea . Bismarck Archipelago . New Irel

913 Land : Pacific islands : Papua New Guinea . Bismarck Archipelago . Umboi

914 Land : Pacific islands : Papua New Guinea . D’Entre Casteaux Islands

915 Land : Pacific islands : Papua New Guinea . D’Entre Casteaux Islands . Ferg

916 Land : Pacific islands : Papua New Guinea . D’Entre Casteaux Islands . Norm

917 Land : Pacific islands : Papua New Guinea . Solomon Islands

918 Land : Pacific islands : Papua New Guinea . Solomon Islands . Bougainville

919 Land : Pacific islands : Papua New Guinea . Solomon Islands . Buka

920 Land : Pacific islands : Papua New Guinea . Woodlark Island

921 Land : Pacific islands : Philippines

922 Land : Pacific islands : Philippines . Basilan

923 Land : Pacific islands : Philippines . Biliran

924 Land : Pacific islands : Philippines . Bohol

925 Land : Pacific islands : Philippines . Burias

926 Land : Pacific islands : Philippines . Busuanga

927 Land : Pacific islands : Philippines . Cebu

928 Land : Pacific islands : Philippines . Culion

929 Land : Pacific islands : Philippines . Dinagat

930 Land : Pacific islands : Philippines . Guimaras

931 Land : Pacific islands : Philippines . Jolo

932 Land : Pacific islands : Philippines . Leyte

933 Land : Pacific islands : Philippines . Luzon

934 Land : Pacific islands : Philippines . Marinduque

935 Land : Pacific islands : Philippines . Masbate

936 Land : Pacific islands : Philippines . Mindanao

937 Land : Pacific islands : Philippines . Mindoro

938 Land : Pacific islands : Philippines . Negros

939 Land : Pacific islands : Philippines . Palawan

940 Land : Pacific islands : Philippines . Panay

941 Land : Pacific islands : Philippines . Polillo Islands

942 Land : Pacific islands : Philippines . Samar

943 Land : Pacific islands : Philippines . Siargao

944 Land : Pacific islands : Philippines . Sibuyan

945 Land : Pacific islands : Philippines . Tablas

946 Land : Pacific islands : Philippines . Tawi Tawi

947 Land : Pacific islands : Philippines . Ticao

948 Land : Pacific islands : Solomon Islands

949 Land : Pacific islands : Solomon Islands . Choiseul

950 Land : Pacific islands : Solomon Islands . Guadalcanal

951 Land : Pacific islands : Solomon Islands . Kolombangara

952 Land : Pacific islands : Solomon Islands . Malaita

953 Land : Pacific islands : Solomon Islands . New Georgia

954 Land : Pacific islands : Solomon Islands . Nggela Sule

955 Land : Pacific islands : Solomon Islands . Rendova

956 Land : Pacific islands : Solomon Islands . Rennell (Mu Nggava)

957 Land : Pacific islands : Solomon Islands . San Cristobal

958 Land : Pacific islands : Solomon Islands . Santa Cruz Island

959 Land : Pacific islands : Solomon Islands . Santa Isabel

960 Land : Pacific islands : Solomon Islands . Vangunu

961 Land : Pacific islands : Solomon Islands . Vella Lavella

962 Land : Pacific islands : Tonga

963 Land : Pacific islands : Tuvalu

964 Land : Pacific islands : United States

965 Land : Pacific islands : United States : Hawaii (state)

966 Land : Pacific islands : United States : Hawaii (state) . Hawaii (island)
967 Land : Pacific islands : United States : Hawaii (state) . Kahoolawe
968 Land : Pacific islands : United States : Hawaii (state) . Kauai
969 Land : Pacific islands : United States : Hawaii (state) . Lanai
970 Land : Pacific islands : United States : Hawaii (state) . Maui
971 Land : Pacific islands : United States : Hawaii (state) . Molokai
972 Land : Pacific islands : United States : Hawaii (state) . Niihau
973 Land : Pacific islands : United States : Hawaii (state) . Oahu
974 Land : Pacific islands : United States : Hawaii (state) . Oahu . islet (Pea
975 Land : Pacific islands : United States : Hawaii (state) . Oahu . islet off
976 Land : Pacific islands : Vanuatu
977 Land : Pacific islands : Vanuatu . Ambrym
978 Land : Pacific islands : Vanuatu . Efate
979 Land : Pacific islands : Vanuatu . Erromango
980 Land : Pacific islands : Vanuatu . Espiritu Santo
981 Land : Pacific islands : Vanuatu . Malekula
982 Land : Pacific islands : Vanuatu . Pentecost Island
983 Land : Pacific islands : Vanuatu . Tanna
984 Land : Pacific islands : Western Samoa
985 Land : Pacific islands : Western Samoa . Savai'i
986 Land : Pacific islands : Western Samoa . Upolu
987 Land : South America
988 Land : South America : (France)
989 Land : South America : (France) . French Guiana
990 Land : South America : Argentina
991 Land : South America : Argentina . Isla de Los Estados
992 Land : South America : Argentina . Tierra del Fuego
993 Land : South America : Bolivia
994 Land : South America : Brazil
995 Land : South America : Brazil . Ilha Caviana
996 Land : South America : Brazil . Ilha Janaucu
997 Land : South America : Brazil . Ilha Maranjo
998 Land : South America : Brazil . Ilha Mexiana
999 Land : South America : Brazil . Ilha Queimada
1000 Land : South America : Brazil . unidentified islands
1001 Land : South America : Chile
1002 Land : South America : Chile . Isla Capitan Arecena
1003 Land : South America : Chile . Isla Dawson
1004 Land : South America : Chile . Isla Desolacion
1005 Land : South America : Chile . Isla Gordon
1006 Land : South America : Chile . Isla Hoste
1007 Land : South America : Chile . Isla Lennox
1008 Land : South America : Chile . Isla Londonderry
1009 Land : South America : Chile . Isla Magdalena
1010 Land : South America : Chile . Isla Navarino
1011 Land : South America : Chile . Isla Nueva
1012 Land : South America : Chile . Isla Picton
1013 Land : South America : Chile . Isla Riesco
1014 Land : South America : Chile . Isla Santa Ines
1015 Land : South America : Chile . Isla Stewart
1016 Land : South America : Chile . Isla Wellington
1017 Land : South America : Chile . Isla de Chiloe
1018 Land : South America : Chile . Tierra del Fuego
1019 Land : South America : Chile . unidentified islands
1020 Land : South America : Colombia
1021 Land : South America : Ecuador
1022 Land : South America : Guyana
1023 Land : South America : Paraguay
1024 Land : South America : Peru
1025 Land : South America : Suriname
1026 Land : South America : Uruguay

1027 Land : South America : Venezuela
1028 Land : South America : Venezuela . Isla de Margarita
1029 Water
1030 Water : Aral Sea (Eurasia)
1031 Water : Azov Sea (Eurasia)
1032 Water : Black Sea (Eurasia)
1033 Water : Buheirat Murrat el Kubra (Africa)
1034 Water : Buheirat el Manzala (Africa)
1035 Water : Calcasieu Lake (North America)
1036 Water : Caspian Sea (Eurasia)
1037 Water : Clark Hill Reservoir (North America)
1038 Water : Grand Lake (North America)
1039 Water : Great Bear Lake (North America)
1040 Water : Great Salt Lake (North America)
1041 Water : Great Slave Lake (North America)
1042 Water : Great South Bay (North America)
1043 Water : Lago de Nicaragua (Central America)
1044 Water : Lagoa dos Patos (South America)
1045 Water : Lake Albert (Africa)
1046 Water : Lake Athabasca (North America)
1047 Water : Lake Baikal (Eurasia)
1048 Water : Lake Balkhash (Eurasia)
1049 Water : Lake Chad (Africa)
1050 Water : Lake Champlain (North America)
1051 Water : Lake Erie (North America)
1052 Water : Lake George (North America)
1053 Water : Lake Kariba (Africa)
1054 Water : Lake Malawi (Africa)
1055 Water : Lake Manitoba (North America)
1056 Water : Lake Maurepas (North America)
1057 Water : Lake Mead (North America)
1058 Water : Lake Okeechobee (North America)
1059 Water : Lake Ontario (North America)
1060 Water : Lake Pontchartrain (North America)
1061 Water : Lake Saint Clair (North America)
1062 Water : Lake Seminole (North America)
1063 Water : Lake Superior (North America)
1064 Water : Lake Tahoe (North America)
1065 Water : Lake Tanganyika (Africa)
1066 Water : Lake Texoma (North America)
1067 Water : Lake Titicaca (South America)
1068 Water : Lake Turkana (Africa)
1069 Water : Lake Victoria (Africa)
1070 Water : Lake Volta (Africa)
1071 Water : Lake Winnipeg (North America)
1072 Water : Lake Winnipegosis (North America)
1073 Water : Lake of the Woods (North America)
1074 Water : Lakes Michigan and Huron (North America)
1075 Water : Namakan Lake (North America)
1076 Water : Ocean
1077 Water : Onazhskoye Ozero (Eurasia)
1078 Water : Ozero Ladozhskoye or Lake Ladoga (Eurasia)
1079 Water : Rainy Lake (North America)
1080 Water : Reindeer Lake (North America)
1081 Water : Sabine Lake (North America)
1082 Water : Sault Sainte Marie (North America)
1083 Water : Sea of Marmara (Eurasia)
1084 Water : Shoal Lake (North America)
1085 Water : White Lake (North America)

Introduction to NCL

The NCAR Command Language is a programming language

NCL is a complete programming language. It has many features common to modern programming languages: it has types, variables, operators, expressions, conditional statements, loops, and functions and procedures. Because it is a programming language, users are required to have some level of experience using a programming language. However, many tasks such as converting file formats and creating graphical output do not require advanced programming skills.

In addition to common programming features, NCL also has features that are not found in other programming languages. These additional features handle the manipulation of metadata, the configuration of the output graphics, the import of data from a variety of data formats, and an algebra that supports array operations.

NCL comes with many useful built-in functions and procedures for processing and manipulating data. There are statistical functions as well as advanced mathematical functions like spherical harmonics.

The primary design goals for NCL were to allow convenient access to data in a variety formats, allow for processing of the data, and allow for the creation of output graphics. NCL is not meant to be a replacement for programming in other structured languages; it is meant to provide an integrated environment where data can be selected, manipulated, and visualized interactively without requiring compilation. In fact, NCL supports calling C and Fortran external routines, which makes NCL infinitely configurable.

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.

Learning to use NCL

NCL has three distinct areas. The first is file I/O. NCL has unique syntax that supports directly referencing variables in data files. Not only can the whole variable be read or written to, but portions of data can be accessed as well using NCL's file variable subscripting rules. Unique syntax for accessing additional file variable information, called metadata, is also useful. This additional information is often grid coordinate information, units, missing values, etc.

Second, there are the data processing features of NCL. The difficulty in learning this area is dependent on the complexity of the user's data processing requirements. Differences between fields on identical grids and averages are fairly straightforward, and many examples exist. However, learning to write efficient NCL data processing code requires a fair amount of understanding of the language.

Finally, there is the graphics component. The syntax for the graphics commands are simple, but the use of them can be complex. NCL uses the NCAR Graphics High Level Utilities Library (HLUs) to configure graphical output. Some discussion of how to create plots is given in this document, but the bulk of the information necessary to configure plots is in the HLU documentation.

NCL reference pages

- Introduction to NCL
- NCL usage help
- Built-in NCL functions and procedures
- Supported data format information
- Creating Graphical Output
- Command line editing in interactive mode
- Default script and shared library loading
- NCL language reference
    - NCL keyword listing
    - NCL data types overview
        - Basic numeric types
        - Non-numeric types
        - Coercion of types
        - Creating data
            - Constants and arrays of constants
            - Creating data arrays with *new*
            - Importing data arrays and files
    - Variables
        - Properties of Variables
            - Attributes
            - Missing values
            - Dimensions
            - Coordinates
            - String references (Using '$' to reference Atts., Coords., and Dimensions)
            - Variables used as parameters to functions and procedures
        - Subscripting variables
            - Standard
            - Coordinate
            - Named
        - Variable assignment
            - Value-only assignment (no automatic assignment of atts., coords., or dims.)
            - Variable-to-variable assignment (includes automatic assignment of atts., coords., and dims.)
        - HLU variables
        - Files and file variables
            - Opening data files
            - Referencing variables in files
                - String references (Using '$' to reference file variables)
            - Assignment to File Variables

Built-in NCL functions and procedures

General NCL functions and procedures

Functions

- addfile - creates a variable that references a file
- all - returns True if *all* input is True
- any - returns True if *any* input is True
- asciiread - reads an ASCII data file
- cbinread - reads a C binary data file
- craybinnumrec - returns the number of records in a COS block sequential file
- craybinrecread - works like fbinrecread, reads multiple unformatted sequential records from a COS block file
- day_of_year - returns the day of year, given the year, month, and day of month (Gregorian calendar)
- days_in_month - returns the number of days in a given month (Gregorian calendar)
- dim_max - finds max for dimension n-1 at all other indexes
- dim_min - finds min for dimension n-1 at all other indexes
- dimsizes - returns an array of integer dimension sizes
- fbindirread - reads records from a file written with Fortran's "access=direct"
- fbinnumrec - counts the number of records in a file written with Fortran's "access=sequential"
- fbinread - reads from a file containing a single unformatted record written with Fortran's "access=sequential"
- fbinrecread - reads from a file containing multiple unformatted records written with Fortran's "access=sequential"
- filevardimsizes - returns an array of integer dimension sizes for file variables
- fspan - returns an array of equally spaced floating point numbers in an inclusive specified range
- getenv - returns a *csh* environment variable
- getfilevaratts - returns a list of attribute names for a file variable
- getfilevardims - returns a list of dimension names for a file variable
- getfilevarnames - returns a list of variable names for a file
- getvaratts - returns a list of attribute names for a variable
- ind - returns the integer indexes of all elements that are True
- isatt - returns True if the attribute exists
- isbyte - returns True if input expression is of type byte
- ischar - returns True if input expression is of type char
- iscoord - returns True if the coordinate variable exists
- isdefined - returns True if the input string is a defined identifier
- isdim - returns True if a dimension exists
- isdouble - returns True if input expression is of type double
- isfile - returns True if input expression is of type file
- isfilevar - returns True if a file variables exists
- isfilevaratt - returns True if a file variable attribute exists
- isfilevardim - returns True if a file variable dimension exists
- isfilevarcoord - returns True if a file variable dimension has a coordinate variable
- isfloat - returns True if input expression is of type float
- isfunc - returns True if input string is a defined function
- isgraphic - returns True if input expression is of type graphic
- isinteger - returns True if input expression is of type integer
- isleapyear - returns True if input is a leap year (Gregorian calendar)
- islogical - returns True if input expression is of type logical

- islong - returns True if input expression is of type long
- ismissing - returns an n-dimensional logical array where elements are True if the input is missing
- isnumeric - returns True if input is a numeric value
- ispan - returns an array of equally spaced integer values
- isproc - returns True if input string is a defined procedure
- isshort - returns True if input expression is of type short
- isstring - returns True if input expression is of type string
- isvar - returns True if a variable exists
- mask - returns masked n-dimensional array
- max - returns the maximum value of an array
- maxind - returns the index of the maximum value for a 1-dimensional array
- min - returns the minimum value of an array
- minind - returns the index of the minimum value for a 1-dimensional array
- monthday - returns concatenated month and day, given the year and day of year (Gregorian calendar)
- ndtooned - converts an n-dimensional array to a 1-dimensional array
- num - returns the number of True values from input
- onedtond - converts a 1-dimensional array to an n-dimensional array
- rand - returns a random number
- sizeof - returns the total size in bytes of the input
- sprintf - formats floating point number as strings
- systemfunc - returns the output of a shell command as a string
- typeof - returns the type string name of an expression or variable

Procedures

- asciiwrite - writes a variable to an ASCII file
- attsetvalues - applies a list of resources to a list of HLU objects
- cbinwrite - writes a variable to a C binary file
- delete - deletes an attribute, coordinate variable, or variable
- exit - exits the current script
- fbindirwrite - writes records in a similar manner as using Fortran's "access=direct"
- fbinrecwrite - writes multiple unformatted Fortran records which can be read in Fortran with "access=sequential"
- fbinwrite - writes a variable as a Fortran unformatted array (flat binary write)
- fileattdef - copies attributes from a variable to a file as global attributes
- filedimdef - defines dimensions in files, used to establish an unlimited dimension
- filevarattdef - copies attributes from a variable to a file variable
- filevardef - defines one or more variables in a file without requiring a write
- list_files - lists all open files
- list_filevars - lists all variables for a file
- list_hlus - lists all HLU objects
- list_procfuncs - lists all procedures and functions
- list_vars - lists all variables
- print - prints a variable or expression
- qsort - sorts a 1-dimensional array
- sleep - sleeps for n seconds

- sqsort - string sort
- srand - sets the random number generator seed
- system - calls a system shell command
- undef - undefines variables, functions, and procedures

NCL math functions and procedures

Functions

- abs - computes absolute value for integer input
- acos - computes inverse cosine in radians
- asin - computes inverse sin in radians
- atan - computes inverse tangent in radians
- atan2 - returns the arctangent of y/x in radians
- avg - average an n-dimensional array of numeric data
- ceil - computes the smallest integer larger than input
- chiinv - evaluates the inverse chi=squared distribution function
- cos - computes cosine from radian input
- cosh - computes hyperbolic cosine from radian input
- cz2ccm - computes geopotential height for a CCM2 hybrid coordinates
- dim_avg - averages dimension n-1 at all other indexes
- dim_product - computes the product for dimension n-1 at all other indexes
- dim_sum - computes the sum for dimension n-1 at all other indexes
- dim_stddev - computes standard deviation for dimension n-1 at all other indexes
- dim_variance - computes variance for dimension n-1 at all other indexes
- dtrend - removes the least squares linear trend and estimates of the linear trend at all grid points
- eofcor - computes empirical orthogonal functions via the correlation matrix
- eofcov - computes empirical orthogonal functions via the covariance matrix
- eofcor_ts - computes time series of eof amplitudes for each eigenvalue returned by eofcor
- eofcov_ts - computes time series of eof amplitudes for each eigenvalue returned by eofcov
- esacr - computes sample autocorrelations
- esacv - computes sample auto covariances
- esccr - computes sample cross correlations
- esccv - computes sample cross covariances
- exp - computes **e** to the power of the input
- ezfftb - computes periodic sequences given Fourier coefficients (synthesis)
- ezfftf - computes Fourier coefficients given periodic sequences (analysis)
- f2fsh - interpolates a scalar quantity from one fixed grid to another using spherical harmonics
- f2gsh - interpolates a scalar quantity on a fixed grid to a Gaussian grid using spherical harmonics
- f2fosh - interpolates a scalar quantity on a fixed grid (including pole points) to a fixed-offset grid using spherical harmonics
- fabs - computes absolute value for floating point input
- floor - computes the largest integer smaller than input
- fluxEddy - calculates time averaged eddy flux quantities

- fo2fsh - interpolates a scalar quantity on a fixed-offset grid (including pole points) to a fixed grid using spherical harmonics
- g2fsh - interpolates a scalar quantity on a Gaussian grid to a fixed grid using spherical harmonics
- g2gsh - interpolates a scalar quantity from one Gaussian grid to another using spherical harmonics
- gaus - computes Gaussian latitudes and weights
- hydro - computes the geopotential height using the hydrostatic equation
- idsfft - computes uniform grid from randomly spaced data
- igradsF - computes the scalar function given the gradient (fixed grid)
- igradsG - computes the scalar function given the gradient (Gaussian grid)
- ilapsF - inverts the Laplacian of a scalar function (fixed grid)
- ilapsG - inverts the Laplacian of a scalar function (Gaussian grid)
- int2p - interpolates from one set of pressure levels to another using linear or log interpolation
- lapsF - computes the Laplacian of a scalar (fixed grid)
- lapsG - computes the Laplacian of a scalar (Gaussian grid)
- log - computes natural log
- log10 - computes log base 10
- product - computes the product of every element of the input
- pslec - computes sea level pressure using the ECMWF formulation
- pslhor - computes sea level pressure using the ECMWF formulation and Trenberth's horizontal correction
- pslhyp - computes sea level pressure via the hypsometric equation
- regcoef - calculates the linear regression coefficient between two series
- relhum - calculates relative humidity
- NEW runave - calculates an unweighted running average
- sin - computes sin from radian input
- sindex_yrmo - calculates the Southern Oscillation Index given two series of year-month values
- snindex_yrmo - calculates the Southern Oscillation Index and the noise given two series of year-month values
- sinh - computes hyperbolic sin from radian input
- NEW specx_anal - calculates spectra quantities of a series
- NEW specxy_anal - calculates cross spectra quantities of a series
- sqrt - computes square root of input
- stddev - computes standard deviation
- sum - computes the sum of every element of the input
- svdcov - computes singular value decomposition
- svdstd - standardizes the input data and computes singular value decomposition
- tan - computes tangent from radian input
- tanh - computes hyperbolic tangent from radian input
- uv2dvF - computes the divergence given u and v wind components (fixed grid)
- uv2dvG - computes the divergence given u and v wind components (Gaussian grid)
- uv2vrF - computes the vorticity given u and v wind components (fixed grid)
- uv2vrG - computes the vorticity given u and v wind components (Gaussian grid)
- variance - computes variance
- vibeta - vertical integration using beta factors
- vinth2p - vertical interpolation (optionally, extrapolation) from hybrid coordinates to

pressure levels
- wgt_runave - calculates a weighted running average

Procedures

- dv2uvf - computes the divergent (irrotational) wind components given the divergence (fixed grid)
- dv2uvg - computes the divergent (irrotational) wind components given the divergence (Gaussian grid)
- f2foshv - interpolates a vector pair on a fixed grid (including pole points) to a fixed-offset grid using spherical harmonics
- f2fshv - interpolates a vector pair from one fixed grid to another using spherical harmonics
- f2gshv - interpolates a vector pair on a fixed grid to a Gaussian grid using spherical harmonics
- fo2fshv - interpolates a vector pair on a fixed-offset grid (including pole points) to a fixed grid using spherical harmonics
- g2fshv - interpolates a vector pair on a Gaussian grid to a fixed grid using spherical harmonics
- g2gshv - interpolates a vector pair from one Gaussian grid to another using spherical harmonics
- gradsf - computes the gradient of a scalar (fixed grid)
- gradsg - computes the gradient of a scalar (Gaussian grid)
- igradsf - computes the scalar function given the gradient (fixed grid)
- igradsg - computes the scalar function given the gradient (Gaussian grid)
- ilapsf - inverts the Laplacian of a scalar quantity (fixed grid)
- ilapsg - inverts the Laplacian of a scalar quantity (Gaussian grid)
- ilapvf - inverts the vector Laplacian (fixed grid)
- ilapvg - inverts the vector Laplacian (Gaussian grid)
- lapsf - computes the Laplacian of a scalar (fixed grid)
- lapsg - computes the Laplacian of a scalar (Gaussian grid)
- lapvf - computes the vector Laplacian of a vector quantity (fixed grid)
- lapvg - computes the vector Laplacian of a vector quantity (Gaussian grid)
- lderuvf - computes the latitudinal derivatives given a vector function (fixed grid)
- lderuvg - computes the latitudinal derivatives given a vector function (Gaussian) grid)
- shaec - spherical harmonic analyses of a scalar (fixed grid)
- shagc - spherical harmonic analyses of a scalar (Gaussian grid)
- shsec - spherical harmonic syntheses of a scalar (fixed grid)
- shsgc - spherical harmonic syntheses of a scalar (Gaussian grid)
- stat_medrng - calculates the median, range, and mid-range of the given vectors
- stat_trim - calculates trimmed estimates of the first two moments of the given vectors
- stat2 - calculates estimates of the first two moments of the given vectors
- stat4 - calculates estimates of the first four moments of the given vectors
- uv2dvf - computes the divergence given u and v wind components (fixed grid)
- uv2dvg - computes the divergence given u and v wind components (Gaussian grid)
- uv2sfvpf - computes the stream function and velocity potential given u and v wind components (fixed grid)
- uv2sfvpg - computes the stream function and velocity potential given u and v wind components (Gaussian grid)

- uv2vrf - computes the vorticity given u and v wind components (fixed grid)
- uv2vrg - computes the vorticity given u and v wind components (Gaussian grid)
- uv2vrdvf - computes the vorticity and divergence given u and v wind components (fixed grid)
- uv2vrdvg - computes the vorticity and divergence given u and v wind components (Gaussian grid)
- vhaec - vector harmonic analyses (fixed grid)
- vhagc - vector harmonic analyses (Gaussian grid)
- vhsec - vector spherical harmonic syntheses (fixed grid)
- vhsgc - vector spherical harmonic syntheses (Gaussian grid)
- vr2uvf - computes non-divergent (rotational) wind components given relative vorticity (fixed grid)
- vr2uvg - computes non-divergent (rotational) wind components given relative vorticity (Gaussian grid)
- vrdv2uvf - computes wind components u and v given vorticity and divergence (fixed grid)
- vrdv2uvg - computes wind components u and v given vorticity and divergence (Gaussian grid)

Natgrid Functions

- natgridd - interpolates 2D random data to a rectangular grid using natural neighbor interpolation
- natgrids - interpolates 2D random data to a rectangular grid using natural neighbor interpolation
- nngetp - gets a parameter value
- nngetaspectd - returns an aspect at a specified coordinate value
- nngetaspects - returns an aspect at a specified coordinate value
- nngetsloped - retrieves a slope at a specified coordinate value
- nngetslopes - retrieves a slope at a specified coordinate value

Natgrid Procedures

- nnpntend - terminates interpolation at single points
- nnpntendd - terminates interpolation at single points
- nnpntinitd - calculates all natural neighbor relationships in an input data array
- nnpntinits - calculates all natural neighbor relationships in an input data array
- nnpntd - interpolates at a specified point
- nnpnts - interpolates at a specified point
- nnsetp - sets a parameter value

Dsgrid Functions

- dsgetp - gets a character parameter
- dsgrid2d - interpolates 2D random data to a rectangular grid using simple inverse distance weighted interpolation
- dsgrid2s - interpolates 2D random data to a rectangular grid using simple inverse distance weighted interpolation
- dsgrid3d - interpolates 3D random data to a rectangular grid using simple inverse distance

weighted interpolation
- dsgrid3s - interpolates 3D random data to a rectangular grid using simple inverse distance weighted interpolation

Dsgrid Procedures

- dssetp - sets a parameter value
- dspnt2d - interpolates 2D data at a list of specified points
- dspnt2s - interpolates 2D data at a list of specified points
- dspnt3d - interpolates 3D data at a list of specified points
- dspnt3s - interpolates 3D data at a list of specified points

Fitgrid Functions

- ftgetp - retrieves control parameter values
- ftcurv - calculates an interpolatory tension spline
- ftcurvd - calculates the derivatives of an interpolatory tension spline
- ftcurvi - calculates integrals of an interpolatory tension spline
- ftcurvp - calculates an interpolatory tension spline for periodic functions
- ftcurvpi - calculates the integral of an interpolatory tension spline for periodic functions
- ftcurvs - calculates a smoothing tension spline
- ftcurvps - calculates a smoothing tension spline for periodic functions
- ftsurf - calculates an interpolatory surface passing through a rectangular grid of function values

Fitgrid Procedures

- ftsetp - sets values for control parameters
- ftkurv - calculates an interpolatory spline for parametric curves
- ftkurvp - calculates an interpolatory spline for closed parametric curves
- ftkurvd - calculates an interpolatory spline and derivatives for parametric curves
- ftkurvpd - calculates an interpolatory spline and derivatives for closed parametric curves

Csagrid Functions

- NEW csa1s - one-dimensional approximation, simple interface
- NEW csa1xs - one-dimensional approximation, expanded interface
- NEW csa2s - two-dimensional approximation, simple interface, gridded output
- NEW csa2xs - two-dimensional approximation, expanded interface, gridded output
- NEW csa2ls - two-dimensional approximation, simple interface, list output
- NEW csa2lxs - two-dimensional approximation, expanded interface, list output
- NEW csa3s - three-dimensional approximation, simple interface, gridded output
- NEW csa3xs - three-dimensional approximation, expanded interface, gridded output
- NEW csa3ls - three-dimensional approximation, simple interface, list output
- NEW csa3lxs - three-dimensional approximation, expanded interface, list output

NCL versions of HLU functions and procedures

Functions

- NhlAddAnnotation - adds an annotation to a plot, returns the annomanager
- NhlAddData - adds a data object to a plot and returns a data spec object
- NhlAppGetDefaultParentId - returns the current default parent App Object
- NhlClassName - returns the class name of an object
- NhlGetBB - returns the bounding box of a view object
- NhlGetParentId - returns the parent id(s) of a list of HLU objects
- NhlGetParentWorkstation - returns the parent workstation id(s) of a list of HLU objects
- NhlGetNamedColorIndex - returns an array of color indexes given workstation ids and a list of named colors
- NhlGetWorkspaceObjectId - returns a reference to the workspace object
- NhlIsAllocatedColor - returns True if the input color index is allocated
- NhlIsApp - returns True if the input object is an App object
- NhlIsDataComm - returns True if the input object is a DataComm object
- NhlIsDataItem - returns True if the input object is a DataItem object
- NhlIsDataSpec - returns True if the input object is a DataSpec object
- NhlIsTransform - returns True if the input object is a Transform object
- NhlIsView - returns True if the input object is a View object
- NhlIsWorkstation - returns True if the input object is a Workstation object
- NhlName - returns the HLU name of the input
- NhlNewColor - allocates a new color and returns its index
- ncargpath - returns the path to the NCAR Graphics installation directory

Procedures

Note: Some of these procedures have abbreviated names that reduce the typing needed to enter them. All alternate names are provided in parentheses.

- clear (NhlClearWorkstation) - clears a workstation
- datatondc (NhlDataToNDC) - converts data coordinates to NDC units
- destroy (NhlDestroy) - destroys an HLU object
- draw (NhlDraw) - draws an HLU view object
- frame (NhlFrame) - calls frame on an NCL workstation object
- ndctodata (NhlNDCToData) - converts NDC units into data coordinates
- ncargversion - prints the version of NCAR Graphics
- overlay - overlays multiple plots over each other
- update (NhlUpdateWorkstation) - updates workstation
- NhlAddOverlay - similar to overlay but allows plot to be inserted in the in-between plots
- NhlChangeWorkstation - changes the workstation of a plot
- NhlClearWorkstation (clear) - clears a workstation
- NhlDataPolygon - draws a polygon clipped to a plot's viewport
- NhlDataPolyline - draws a line clipped to a plot's viewport
- NhlDataPolymarker - draws markers in a plot's viewport
- NhlDataToNDC (datatondc) - converts data coordinates to NDC units

- NhlDestroy (destroy) - destroys a plot
- NhlDraw (draw) - draws a plot
- NhlFrame (frame) - calls frame on the workstation
- NhlFreeColor - frees an allocated color
- NhlNDCPolygon - draws a polygon in NDC units clipped to a plot's viewport
- NhlNDCPolyline - draws a polyline in NDC units clipped to a plot's viewport
- NhlNDCPolymarker - draws markers in NDC units clipped to a plot's viewport
- NhlNDCToData (ndctodata) - converts NDC units to data coordinates
- NhlRemoveAnnotation - removes an annotation from a plot manager
- NhlRemoveData - removes DataItems from a plot
- NhlRemoveOverlay - removes a plot from an overlay stack
- NhlSetColor - defines a color
- NhlUpdateData - tells a plot to update its information about its data
- NhlUpdateWorkstation (update) - calls update on a workstation

NCL versions of LLU functions and procedures

Functions

- ncargpath
- wmgetp - get control parameter values for wmap.

Procedures

- ncargversion
- tdez2d - simplified entry to Tdpack for drawing surfaces
- tdez3d - simplified entry to Tdpack for drawing isosurfaces
- wmbarb - draw wind barbs
- wmsetp - set control parameter values for wmap.

NCL data type conversion functions

Note: The reason the functions *floattodouble*, *shorttointeger*, *integertofloat*, etc. don't exist is NCL's implicit coercion rules handle data type conversions that do not potentially truncate the data.

- chartodouble
- chartofloat
- chartointeger
- chartolong
- chartoshort
- chartostring
- doubletobyte
- doubletochar
- doubletofloat
- doubletointeger
- doubletolong

- doubletoshort
- floattobyte
- floattochar
- floattointeger
- floattolong
- floattoshort
- integertobyte
- integertochar
- integertoshort
- longtobyte
- longtochar
- longtointeger
- longtoshort
- shorttobyte
- shorttochar
- stringtochar
- stringtodouble
- stringtofloat
- stringtointeger
- stringtolong
- stringtoshort

NCL data type conversion functions

These functions are used to coerce values from one basic data type to another.

Synopsis

```
function integertobyte(
        int_val : integer
)


function integertochar(
        int_val : integer
)

function integertoshort(
        int_val : integer
)

function shorttobyte(
        short_val : short
)

function shorttochar(
        short_val : short
)
```

```
function longtobyte(
        long_val : long
)

function longtochar(
        long_val : long
)

function longtoshort(
        long_val : long
)

function longtointeger(
        long_val : long
)

function floattobyte(
        float_val : float
)

function floattochar(
        float_val : float
)

function floattoshort(
        float_val : float
)

function floattointeger(
        float_val : float
)

function floattolong(
        float_val : float
)

function doubletobyte(
        double_val : double
)

function doubletochar(
        double_val : double
)

function doubletoshort(
        double_val : double
)

function doubletointeger(
        double_val : double
)

function doubletolong(
        double_val : double
)

function doubletofloat(
        double_val : double
)

function chartodouble(
```

```
        char_val : character
)

function chartofloat(
        char_val : character
)

function chartolong(
        char_val : character
)

function chartoshort(
        char_val : character
)

function chartostring(
        char_val : character
)

function chartointeger(
        char_val : character
)

function stringtochar(
        char_val : string
)

function stringtodouble(
        char_val : string
)

function stringtofloat(
        char_val : string
)

function stringtointeger(
        char_val : string
)

function stringtolong(
        char_val : string
)

function stringtoshort(
        char_val : string
)
```

Description

All of the functions listed here operate similarly. Each takes a parameter of any dimension and size and converts it to a similarly dimensioned value of a different type. These functions perform coercion that is not automatically available through the NCL grammar. (See Coercion of types).

In most of these functions, some information can be lost. For example, converting a floating-point number to an integer truncates the decimal portion of the floating point number. Also when a floating-point number is greater than or less than 2.147483e+09, a 32-bit integer is not able to represent the number. In situations like this, the default missing value for the output type is returned. Checking these limits is only implemented in functions that convert to longs, ints, shorts, bytes, and characters.

The character conversion functions (**charto....**) convert the ASCII value of the character to whatever numeric type is requested. Therefore, conversion of the letter 'a' to an integer yields the value 97.

The **stringtochar** and **chartostring** functions are the only conversion functions where the dimension of the output is different than the input. For **stringtochar**, the output number of dimensions is *n - 1* where *n* is the number of dimensions of the input. The dimension sizes for dimensions 0 through *n-1* remain the same. For **chartostring** the output number of dimensions increases by 1. The new dimension's size is the string length of the longest string in the input parameter plus 1.

NhlAddAnnotation

This function is used to add annotations to a plot object as an external annotation. It returns a reference to one or more AnnoManager instances. (See NhlAddAnnotation.)

Synopsis

```
function NhlAddAnnotation(
        plot_id [1] : graphic,
        anno_view_id : graphic
)
```

Arguments

plot_id
A reference to an HLU plot object to which the annotation is to be added.

anno_view_id
One or more view objects to be added as an annotation.

Description

This function adds one or more views to a plot as annotations. The return value is a multi-dimensional array containing references to **annomanagers** created during the call. Each element in *anno_view_id* is added to the *plot_id*. The output has the same dimensions as the *anno_view_id* parameter.

NhlAddData

This function is used to add one or more additional data items to a plot. (See NhlAddData.)

Synopsis

```
function NhlAddData(
        dcomms[*] : graphic,
        res_name[1] : string,
        data_items[*] : graphic
)
```

Arguments

dcomms
One or more DataComm instances.
res_name
A single-dimension string resource name to assign the data_item values to.
data_items
One or more DataItem instances to be added to each of the objects in the *dcomms* array.

Description

The **NhlAddData** function works like its HLU counterpart, with the following exceptions. One or more DataComm instances can be assigned data during a single call, and one or more DataItem instances can be assigned to each element of the *dcomms* array. Note that only one resource name is used, so all objects in the *dcomms* array must accept the same resource. The return value is a multi-dimensional array of references to data-specific objects created by the add data call. The output is dimensioned m x n, where m is the dimension size of *dcomms*, and n is the dimension size of *data_items*.

NhlAppGetDefaultParentId

Returns a reference to the current default App object.

Synopsis

```
function NhlAppGetDefaultParentId()
```

Description

This function returns a reference to the current default App object. The default App object controls which resource database is used by default when creating HLU objects.

NhlChangeWorkstation

This procedure is used to change the output workstation of one or more HLU View instances. (See NhlChangeWorkstation.)

Synopsis

```
	procedure NhlChangeWorkstation(
		hlu_objs : graphic
		workstation [1] : graphic,
	)
```

Arguments

hlu_objs
: An array of one or more references to HLU View instances.

workstation
: HLU Workstation instance.

Description

This procedure is used to change the parent workstation for one or more HLU View instances. The parent workstation for each View instance in the array *hlu_obj* is changed to the workstation referenced by *workstation*.

NhlClassName

This function is used to retrieve the class name of one or more HLU objects. (See NhlClassName.)

Synopsis

```
function NhlClassName(
        objs[*]: graphic
)
```

Arguments

objs
: An array of one or more instances of HLU objects.

Description

This function returns an array of strings which are the class names of each object of the *objs* array. This function works like its HLU counterpart, but uses one or more objects as input.

NhlClearWorkstation (clear)

This procedure is used to clear one or more workstations. (See NhlClearWorkstation.) Two names are provided for this NCL procedure: the formal name is **NhlClearWorkstation**, and the abbreviated name is **clear**.

Synopsis

```
procedure NhlClearWorkstation(
        workstations:graphic
)

or

procedure clear(
        workstations:graphic
)
```

Arguments

workstations
: An array of one or more references to HLU Workstation instances.

Description

Each element of the *workstations* parameter is cleared. If any of the workstations pause on the clear procedure, they must have a button click or key click in them before the actual clearing occurs. The NCL functions NhlFrame and NhlUpdateWorkstation are related to this call. See their descriptions for more information.

NhlDataPolygon

This procedure is used to draw a polygon using data coordinates. (See NhlDataPolygon.)

Synopsis

```
procedure NhlDataPolygon(
        objs[*]: graphic,
        style[*]:graphic
        x[*]: float,
        y[*]: float
)
```

Arguments

objs
: An array of HLU plot objects.

style
: Either one HLU style object or an array with the same dimensions as the *objs* parameter.

x
: The X data coordinates of the polygon in data coordinates.

y
: The Y data coordinates of the polygon in data coordinates.

Description

NhlDataPolygon draws the polygon defined by the pair of vectors *x* and *y* using either a single style object or the corresponding style object from the *style* parameter. The vectors *x* and *y* contain values in data coordinates. The coordinates are mapped using the HLU object's data transformation and drawn as a polygon clipped to the viewports of each HLU plot object in the *objs* array. The difference between the NCL version and the HLU version is that the NCL version draws the polygon over one or more HLU plot objects using one or more styles, and the NCL version does not need the length of the *x* and *y* parameters because this information is inherent in NCL data. In the HLU version, NULL can be used to use the default graphic style object. In NCL, however, the default style object must be retrieved from the workstation parent using the wkDefGraphicStyleId resource and the getvalues statement. For more information on controlling how the polygon is drawn and filled, see the GraphicStyle class description.

NhlDataPolyline

This procedure is used to draw a polyline using data coordinates. (See NhlDataPolyline.)

Synopsis

```
procedure NhlDataPolyline(
        objs[*]: graphic,
        style[*]:graphic
        x[*]: float,
        y[*]: float
)
```

Arguments

objs

An array of HLU plot objects.

style

Either one HLU style object or an array with the same dimensions as the *objs* parameter.

x

The X data coordinates of the polyline in data coordinates.

y

The Y data coordinates of the polyline in data coordinates.

Description

NhlDataPolyline draws the polyline defined by the pair of vectors *x* and *y* using either a single style object or the corresponding style object from the *style* parameter. The vectors *x* and *y* contain values in data coordinates. The coordinates are mapped using the HLU object's data transformation and drawn as a polyline clipped to the viewports of each HLU plot object in the *objs* array. The difference between the NCL version and the HLU version is that the NCL version draws the polyline over one or more HLU plot objects using one or more styles, and the NCL version does not need the length of the *x* and *y* parameters because this information is inherent in NCL data. In the HLU version, NULL can be used to use the default graphic style object. In NCL, however, the default style object must be retrieved from the workstation parent using the wkDefGraphicStyleId resource and the getvalues statement. For more information on controlling how the polyline is drawn, see the GraphicStyle class description.

NhlDataPolymarker

This procedure is used to draw a polymarker using data coordinates. (See NhlDataPolymarker.)

Synopsis

```
procedure NhlDataPolymarker(
        objs[*]: graphic,
        style[*]:graphic
        x[*]: float,
        y[*]: float
)
```

Arguments

objs
: An array of HLU plot objects.

style
: Either one HLU style object or an array with the same dimensions as the *objs* parameter.

x
: The X data coordinates of the polymarker in data coordinates.

y
: The Y data coordinates of the polymarker in data coordinates.

Description

NhlDataPolymarker draws the polymarker defined by the pair of vectors *x* and *y* using either a single style object or the corresponding style object from the *style* parameter. The vectors *x* and *y* contain values in data coordinates. The coordinates are mapped using the HLU object's data transformation and drawn as a polymarker clipped to the viewports of each HLU plot object in the *objs* array. The difference between the NCL version and the HLU version is that the NCL version draws the polymarker over one or more HLU plot objects using one or more styles, and the NCL version does not need the length of the *x* and *y* parameters because this information is inherent in NCL data. In the HLU version, NULL can be used to use the default graphic style object. In NCL, however, the default style object must be retrieved from the workstation parent using the wkDefGraphicStyleId resource and the getvalues statement. For more information on controlling how the polymarker is drawn, see the GraphicStyle class description.

NhlDataToNDC (datatondc)

Converts data units into normalized device coordinates (NDCs). (See NhlDataToNDC.) Two names are provided for this NCL procedure: the formal name is NhlDataToNDC, and the abbreviated name is datatondc.

Synopsis

```
procedure NhlDataToNDC(
        plot[1] : graphic,
        x_in[*] : float,
        y_in[*] : float,
        x_out[*] : float,
        y_out[*] : float
)

or

procedure datatondc(
        plot[1] : graphic,
        x_in[*] : float,
        y_in[*] : float,
        x_out[*] : float,
        y_out[*] : float
)
```

Arguments

plot
    HLU plot.
x_in
    Single-dimensioned floating-point vector of x data coordinates.
y_in
    Single-dimensioned floating-point vector of y data coordinates; must be the same size as *x_in*.
x_out
    Single-dimensioned floating-point vector the same size as *x_in*; will contain x-coordinate output for **datatondc**.
y_out
    Single-dimensioned floating-point vector the same size as *x_out*; will contain y-coordinate output from **datatondc**.

Description

The **NhlDataToNDC** procedure uses the built-in transformation mapping feature of HLU plot objects to map data coordinates into Normalized Device Coordinates (screen coordinates). All four floating-point parameters must have the same number of elements. **NhlDataToNDC** maps coordinate pairs from *x_in* and *y_in* and places the results in the corresponding elements of *x_out* and *y_out* respectively. Both *x_out* and *y_out* can be the same variables as *x_in* and *y_in*. When the data coordinates are outside of the data transformation specified in the plot, a missing value is placed in both of the corresponding indexes in the output arrays. The missing value chosen is either the *x_in* missing value, *y_in* missing value, or a system default.

NhlDestroy (destroy)

This procedure is used to destroy instances HLU objects from NCL. Two names are provided for this NCL procedure: the formal name is NhlDestroy, and the abbreviated name is destroy. (See NhlDestroy.)

Synopsis

```
procedure NhlDestroy(
        obj:graphic
)

or

procedure destroy(
        obj:graphic
)
```

Arguments

obj

An array of one or more HLU object instances to be destroyed.

Description

Each element of the *obj* parameter is destroyed and replaced with the default missing value for HLU objects. Unlike the delete command, NhlDestroy does not delete the variable referencing the objects.

NhlDraw (draw)

This procedure is used to call the HLU draw function (NhlDraw) on NCL data that are arrays of HLU

View instances. Two names are provided for this NCL procedure: the formal name is NhlDraw, and the abbreviated name is draw. (See NhlDraw.)

Synopsis

```
procedure NhlDraw(
        view : graphic
)

or

procedure draw(
        view : graphic
)
```

Arguments

view

An array of HLU View instances of any dimensionality.

Description

The **NhlDraw** procedure calls the HLU function NhlDraw for each element of the array *view*. If any of the ids referenced by the parameter *hlu_obj* are invalid, an HLU error message will be displayed. HLUs are created using the *create* language construct.

NhlFrame (frame)

This procedure updates and then clears the HLU workstations that are passed to it as parameters. (See NhlFrame.) Two names are provided for this NCL procedure: the formal name is NhlFrame, and the abbreviated name is frame.

Synopsis

```
procedure NhlFrame(
        work : graphic
)

or

procedure frame (
        work : graphic
```

```
)
```

Arguments

work

An array of HLU Workstation instances of any dimensionality.

Description

NhlFrame updates and then clears the Workstation instances passed to it as elements of the array *work*. If the parameter references HLU objects that are not Workstation instances, HLU library errors will be generated. The frame procedure calls the HLU function NhlFrame.

NhlFreeColor

This procedure is used to remove one or more color entries from one or more HLU Workstation instances. (See NhlFreeColor.)

Synopsis

```
function NhlFreeColor(
        work[*]:graphic,
        ci[*]:integer
)
```

Arguments

work

One or more Workstation instances.

ci

One or more workstation color indexes to be freed.

Description

The **NhlFreeColor** procedure works like its HLU counterpart, except it accepts multiple Workstation instances and color indexes as input. This function frees each color index in *ci* from each workstation in the *work* parameter.

NhlGetBB

This function is used retrieve the bounding boxes of a list of HLU input objects. (See NhlGetBB.)

Synopsis

```
function NhlGetBB(
        objs[*]:graphic
)
```

Arguments

objs

List of HLU View instances from which bounding boxes are requested.

Description

The HLU version of this returns a data structure containing the top, bottom, left, and right NDC units of a View instance. In NCL, this function returns an array of the four coordinates. Index 0 contains the top coordinate, 1 contains the bottom, 2 contains the left, and 3 contains the right. If more than one View instance is passed in in the *objs* array. The return array is dimensioned [n]x[4] where n is the number of View instances in the *objs* parameter. If any element of the *objs* parameter is not a valid plot, the four bounding box elements associated with its index will be assigned the integer default missing value.

NhlGetNamedColorIndex

This function takes as input an array of workstation ids (any dimensionality) and an array of color names (any dimensionality) and returns an array of color map indexes that match the color names the closest.

Note: This function is only available in version 4.1.1 of NCL. If your site is licensed for version 4.1, then you should receive version 4.1.1 automatically.

Synopsis

```
function NhlGetNamedColorIndex(
   wks        : graphic,
   color_name : string
)
```

Arguments

wks
An array of any dimensionality of NCL workstation ids
color_name
An array of any dimensionality of named colors from the $NCARG_ROOT/lib/ncarg/database/rgb.txt file

---

Description

NhlGetNamedColorIndex looks up the given color names in the color maps of the given workstations and returns an integer array of color indexes for each workstation id that represent the closest match to each color. If a bogus color name is given, then the color index returned will be negative. The array returned will be dimensioned *n* x *m* where *n* represents the dimensions of *wks* and *m* represents the dimensions of *color_name*. If only one workstation is given, then the dimensions of the output will be equal to the dimensions of *color_name*.

Example

If you have the following NCL script:

```
begin
  wks = create "workstation" ncgmWorkstationClass defaultapp
    "wkColorMap" : (/"white","black","red","green","blue","yellow",\
                     "cyan","magenta"/)
  end create

  i = NhlGetNamedColorIndex(wks,(/"red","yellow","blue","green",\
                                   "cyan","magenta"/))
  j = NhlGetNamedColorIndex(wks,(/"gray","brown"/))
  print(i)
  print(j)
end
```

then `i` will be equal to (/2,5,4,3,6,7/) and `j` will be equal to (/0,2/). There's no gray or brown defined in the above color map, so when the color map is searched for the closest match to gray and brown, it comes up with white and red, which are color indexes 0 and 2.

NhlGetParentId

Return the id(s) of the parent(s) of the object(s) passed in. (See NhlGetParentId.)

Synopsis

```
function NhlGetParentId(
        objs : graphic
)
```

Arguments

obj

An array of one or more HLU objects whose parent ids are requested.

Description

Returns and array of parent object id's. If an object has no parent either the default parent is returned if the object was created with the *defaultapp* as the parent or the parent id that was provided with the objects *create* statement.

NhlGetParentWorkstation

Return the id(s) of the parent workstation(s) of the object(s) passed in.

Synopsis

```
function NhlGetParentWorkstation(
        obj : graphic
)
```

Arguments

obj

List of HLU View instances for which the objects workstation is requested .

Description

This function returns an array the same size as the input array. The output is a list of HLU workstation objects corresponding to the workstations of the input array. If an object is not a View class and hence does not have a parent workstation associated with it a missing value is returned.

NhlGetWorkspaceObjectId

Returns a reference to the current WorkspaceObject.

Synopsis

```
function NhlGetWorkspaceObjectId()
```

Description

The Workspace object manages blocks of memory used by the HLU objects. This function returns a reference to the workspace object so the workspace array sizes may be configured by the user.

NhlIsAllocatedColor

This function queries a workstation to to determine whether or not a given color index has been allocated. (See NhlIsAllocatedColor.)

Synopsis

```
function NhlIsAllocatedColor(
        work[*]:graphic,
        ci[*]:integer
)
```

Arguments

work

One or more HLU Workstation instances.

ci

An array of one or more color indexes. Valid range is 0-255.

Description

The Workstation instances referenced by *work* are queried to see whether each of the elements of *ci* is an allocated color or not. This function returns and array of logical values. If an element of *ci* is a missing value, then the corresponding output value is a missing value. The output is dimension *m x n* where m is the dimension size of the *work* parameter and n is the dimension of the *ci* parameter.

NhlIsApp

This function is used to distinguish HLU view object from other types of HLU Object.

Synopsis

```
function NhlIsApp(
        obj : graphic
)
```

Arguments

obj

An array of one or more HLU objects.

Description

This function returns a logical array with the same dimensions as the input argument *obj*. The output array elements are True when the corresponding input element references an HLU App object and False when the input element is any other class of HLU object.

NhlIsDataComm

Used to distinguish between HLU Workstation objects from other HLU objects. (See also NhlIsDataComm.html)

Synopsis

```
function NhlIsDataComm(
        obj : graphic
)
```

Arguments

obj

An array of one or more HLU objects.

Description

This function returns a logical array with the same dimensions as the input argument *obj*. The output array elements are True when the corresponding input element references an HLU DataComm object and False when the input element is an object of any other HLU class. DataComm objects are objects that accept, manage and display data.

NhlIsDataItem

Used to distinguish between HLU Workstation objects from other HLU objects. (See also NhlIsDataItem.html)

Synopsis

```
function NhlIsDataItem(
        obj : graphic
)
```

Arguments

obj

An array of one or more HLU objects.

Description

This function returns a logical array with the same dimensions as the input argument *obj*. The output array elements are True when the corresponding input element references an HLU DataItem object and False when the input element is an object of any other HLU class.

NhlIsDataSpec

Used to distinguish between HLU Workstation objects from other HLU objects. (See also NhlIsDataSpec.html)

Synopsis

```
function NhlIsDataSpec(
        obj : graphic
)
```

Arguments

obj
: An array of one or more HLU objects.

Description

This function returns a logical array with the same dimensions as the input argument *obj*. The output array elements are True when the corresponding input element references an HLU DataSpec object and False when the input element is an object of any other HLU class.

NhlIsTransform

Used to distinguish between HLU Transform objects and other HLU objects. (See also NhlIsTransform)

Synopsis

```
function NhlIsTransform(
        obj : graphic
)
```

Arguments

obj

An array of one or more HLU objects.

Description

This function returns a logical array with the same dimensions as the input argument *obj*. The output array elements are True when the corresponding input element references an HLU Transform object. and False when the input element is any other HLU object. Transform objects are HLU objects that support the NhlDataToNDC and NhlNDCToData functions as well as many other drawing functions.

NhlIsView

This function is used to distinguish HLU view object from other types of HLU Object.

Synopsis

```
function NhlIsView(
        obj : graphic
)
```

Arguments

obj

An array of one or more HLU objects.

Description

This function returns a logical array with the same dimensions as the input argument *obj*. The output array elements are True when the corresponding input element references an HLU View object and False when the input element is any other class of HLU object.

NhlIsWorkstation

Used to distinguish between HLU Workstation objects and other HLU objects. (See also NhlIsWorkstation)

Synopsis

```
function NhlIsWorkstation(
        wk : graphic
)
```

Arguments

wk

One or more references to HLU objects

Description

This function returns a logical array with the same dimensions as the input argument *wk*. The array elements are True when the corresponding input element references an HLU Workstation object and False when the input element is any other HLU object.

NhlNDCPolygon

This procedure is used to draw a polygon using NDC units. (See NhlNDCPolygon.)

Synopsis

```
procedure NhlNDCPolygon(
        objs[*]: graphic,
        style[*]:graphic
        x[*]: float,
        y[*]: float
)
```

Arguments

objs
: An array of HLU plot objects.

style
: Either one HLU style object or an array with the same dimensions as the *objs* parameter.

x
: The X coordinates of the polygon in NDC units.

y
: The Y coordinates of the polygon in NDC units.

Description

NhlNDCPolygon draws the polygon defined by the pair of vectors *x* and *y* using either a single style object or the corresponding style object from the *style* parameter. The vectors *x* and *y* contain values in NDC units. The coordinates are drawn as a polygon clipped to the viewports of each HLU plot object in the *objs* array. The difference between the NCL version and the HLU version is that the NCL version draws the polygon over one or more HLU plot objects using one or more styles, and the NCL version does not need the length of the *x* and *y* parameters because this information is inherent in NCL data. In the HLU version, NULL can be used to use the default graphic style object. In NCL, however, the default style object must be retrieved from the workstation parent using the wkDefGraphicStyleId resource and the getvalues statement. For more information on controlling how the polygon is drawn and filled, see the GraphicStyle class description.

NhlNDCPolyline

This procedure is used to draw a polyline using NDC units. (See NhlNDCPolyline.)

Synopsis

```
procedure NhlNDCPolyline(
        objs[*]: graphic,
        style[*]:graphic
        x[*]: float,
        y[*]: float
)
```

Arguments

objs
: An array of HLU plot objects.

style
: Either one HLU style object or an array with the same dimensions as the *objs* parameter.

x
: The X coordinates of the polyline in NDC units.

y
: The Y coordinates of the polyline in NDC units.

Description

NhlNDCPolyline draws the polyline defined by the pair of vectors *x* and *y* using either a single style object or the corresponding style object from the *style* parameter. The vectors *x* and *y* contain values in NDC units. The coordinates are drawn as a polyline clipped to the viewports of each HLU plot object in the *objs* array. The difference between the NCL version and the HLU version is that the NCL version draws the polyline over one or more HLU plot objects using one or more styles, and the NCL version does not need the length of the *x* and *y* parameters because this information is inherent in NCL data. In the HLU version, NULL can be used to use the default graphic style object. In NCL, however, the default style object must be retrieved from the workstation parent using the wkDefGraphicStyleId resource and the getvalues statement. For more information on controlling how the polyline is drawn, see the GraphicStyle class description.

NhlNDCPolymarker

This procedure is used to draw a polymarker using NDC units. (See NhlNDCPolymarker.)

Synopsis

```
procedure NhlNDCPolymarker(
        objs[*]: graphic,
        style[*]:graphic
        x[*]: float,
        y[*]: float
)
```

Arguments

objs
: An array of HLU plot objects.

style
: Either one HLU style object or an array with the same dimensions as the *objs* parameter.

x
: The X coordinates of the polymarker in NDC units.

y
: The Y coordinates of the polymarker in NDC units.

Description

NhlNDCPolymarker draws the polymarker defined by the pair of vectors *x* and *y* using either a single style object or the corresponding style object from the *style* parameter. The vectors *x* and *y* contain values in NDC units. The coordinates are drawn as a polymarker clipped to the viewports of each HLU plot object in the *objs* array. The difference between the NCL version and the HLU version is that the NCL version draws the polymarker over one or more HLU plot objects using one or more styles, and the NCL version does not need the length of the *x* and *y* parameters because this information is inherent in NCL data. In the HLU version, NULL can be used to use the default graphic style object. In NCL, however, the default style object must be retrieved from the workstation parent using the wkDefGraphicStyleId resource and the getvalues statement. For more information on controlling how the polymarker is drawn, see the GraphicStyle class description.

NhlNDCToData (ndctodata)

Converts NDC units into data coordinates. (See NhlNDCToData.) Two names are provided for this NCL procedure: the formal name is NhlNDCToData, and the abbreviated name is ndctodata.

Synopsis

```
procedure NhlNDCToData(
        plot[1] : graphic,
        x_in[*] : float,
        y_in[*] : float,
        x_out[*] : float,
        y_out[*] : float
)

or

procedure ndctodata(
        plot[1] : graphic,
        x_in[*] : float,
        y_in[*] : float,
        x_out[*] : float,
        y_out[*] : float
)
```

Arguments

plot

Instance of an HLU plot.

x_in

Single-dimensioned floating-point vector of x NDC units.

y_in

Single-dimensioned floating-point vector of y NDC units; must be the same size as *x_in*.

x_out

Single-dimensioned floating-point vector the same size as *x_in*; will contain x-coordinate output for **NhlNDCToData**.

y_out

Single-dimensioned floating-point vector the same size as *x_out*; will contain y-coordinate output from **NhlNDCToData**.

Description

The **NhlNDCToData** procedure uses the built-in transformation mapping feature of HLU plot objects to map NDC coordinates into data coordinates. All four floating-point parameters must have the same number of elements. **NhlNDCToData** maps coordinate pairs from *x_in* and *y_in* and places the results in the corresponding elements of *x_out* and *y_out*, respectively. Both *x_out* and *y_out* can be the same variables as *x_in* and *y_in*. When the data coordinates are outside the NDC viewport specified in the plot, a missing value is placed in both of the corresponding indexes in the output arrays. The missing value chosen is either the *x_in* missing value, *y_in* missing value, or a system default.

NhlName

This function is used to retrieve the name of one or more HLU object instances. (See NhlName.)

Synopsis

```
function NhlName(
        objs[*]:graphic
)
```

Arguments

objs

Contains one or more HLU object instances.

Description

This function returns the HLU name--assigned during create--of each element of the *objs* array. If an item of *obj* is either missing or is an invalid HLU object, the result is a missing value.

NhlNewColor

The function is used to allocate new workstation color indexes.(See NhlNewColor.)

Synopsis

```
function NhlNewColor(
        workstations [*] : graphic,
        red [*] : float,
        green [*] : float,
        blue [*] : float
)
```

Arguments

workstations
: One or more HLU Workstation instances.

red
: Array of floating point values between 0 and 1 inclusive. Must be dimensioned the same as *green* and *blue*.

green
: Array of floating point values between 0 and 1 inclusive. Must be dimensioned the same as *red* and *blue*.

blue
: Array of floating point values between 0 and 1 inclusive. Must be dimensioned the same as *green* and *red*.

Description

For each Workstation instance, this function allocates color indexes for every red, green, blue triplet. Since the color indexes allocated can be different for each Workstation instance, the return array of color indexes is dimensioned [m]x[n], where m is the number of Workstation instances, and n is the number of rgb triplets.

NhlRemoveAnnotation

This procedure is used to remove annomanagers from the plots they are registered in. (See NhlRemoveAnnotation.)

Synopsis

```
procedure NhlRemoveAnnotation(
        plot_id[1] : graphic,
        anno_manager_id : graphic
)
```

Arguments

plot_id
: One HLU plot object.

anno_manager_id
: References to one or more AnnoManager instances to be removed from each of the *plot_id* elements.

Description

This function is used to remove external annotations from a plot they have previously been added to using the NhlAddAnnotation function.

NhlRemoveData

This procedure is used to remove data items from one or more plots. (See NhlRemoveData)

Synopsis

```
procedure NhlRemoveData(
        plot_objs[*]:graphic,
        resname[1]:string,
        data_objs[*]:graphic
)
```

Arguments

plot_objs

An array of one or more HLU plots from which data are to be removed.

resname

The resource name from which the data should be removed.

data_objs

An array of one or more HLU DataItem instances to be removed from each of the plots.

Description

The **NhlRemoveData** procedure removes one or more DataItem instances from one or more HLU plots given the resource name from which the data should be removed. The data items being removed were either added with the NhlAddData or added using create or setvalues.

NhlRemoveOverlay

This function is used to remove one or more plots from an overlay. (See NhlRemoveOverlay.)

Synopsis

```
function NhlRemoveOverlay(
        base[1]:graphic,
        plot[*]:graphic,
        restore[1]:logical
)
```

Arguments

base

References the HLU plot that is the base of an overlay.

plot

An array of one or more HLU plots to be removed from the base.

restore

Single-dimensioned logical value. If True and the member plot initially was a base plot of an overlay with its own members, the member plots are returned to the plot being removed.

Description

This function removes all of the elements of the *plot* array from the *base* plot and applies the *restore* value to each element. The restore parameter dictates how overlay elements that are themselves overlay bases handle their members. (See the HLU PlotManager description for more detail about overlays.)

NhlSetColor

The function is used to allocate new workstation color indexes. (See NhlSetColor)

Synopsis

```
        function NhlSetColor(
                workstations[*]:graphic,
                ci[*]:integer,
                red[*]:float,
                green[*]:float,
                blue[*]:float
        )
```

Arguments

workstations
: One or more HLU workstation references.

ci
: Array of integer color indexes between 0 and 255. Must be dimensioned the same as *red*, *blue*, and *green*.

red
: Array of floating-point values between 0 and 1 inclusive. Must be dimensioned the same as *green* and *blue*.

green
: Array of floating-point values between 0 and 1 inclusive. Must be dimensioned the same as *red* and *blue*.

blue
: Array of floating-point values between 0 and 1 inclusive. Must be dimensioned the same as *green* and *red*.

Description

For each workstation, the color indexes in *ci* are set to the corresponding rgb triplets.

NhlUpdateData

This procedure is used to force one or more DataComm instances to update their internal state to reflect any changes that have occurred in DataItem instances that have been associated with the DataComm instances. (See NhlUpdateData.)

Synopsis

```
procedure NhlUpdateData(
        dcomms[*]:graphic
)
```

Arguments

dcomms

Contains one or more instances of the DataComm class to be updated.

Description

This procedure is really only useful when the items in *dcomms* have the resource dcDelayCompute set to True. When this is True, DataComm instances are not notified of changes to their data items. This procedure tells each object in *dcomms* to check for changes in its data. These data were added with the NhlAddData or added using create or setvalues.

NhlUpdateWorkstation (update)

This procedure is used to update HLU workstation objects. (See NhlUpdateWorkstation .) Two names are provided for this NCL procedure: the formal name is NhlUpdateWorkstation, and the abbreviated name is update.

Synopsis

```
procedure NhlUpdateWorkstation(
        work : graphic
)

or

procedure update(
        work : graphic
)
```

Arguments

work

An array of one or more HLU Workstation instances of any dimensionality.

Description

The **NhlUpdateWorkstation** procedure is used to flush the internal buffers of HLU Workstation instances. It calls the HLU function NhlUpdateWorkstation for each element of the *work* array.

abs

This function is used to return the absolute values for integers.

Synopsis

```
function abs(
        value:integer
)
```

Arguments

value

One or more integer values of any dimension.

Returns the absolute value of each element in *value* as an array dimensioned the same as the *value* parameter. Missing values are ignored.

acos

This function returns the inverse cosine in radians.

Synopsis

```
function acos(
        value:numeric
)
```

Arguments

value
: One or more values of any dimension for which the inverse cosine is to be determined.

Description

The inverse cosine of each element of the *value* is determined and returned in **radians**. Missing values are ignored. Returns a floating point array dimensioned the same as the input parameter. Return value type is double if double value passed in.

addfile

This function is used to open a new data file for NCL.

Synopsis

```
function addfile (
        file_path[ 1 ] : string,
        status[ 1 ] : string
)
```

Arguments

file_path
: Single string that is the full or relative path of the data file to load.

status
: Single string that specifies whether the file should be open as a read only file or as a read-write file.

Description

The **addfile** function returns a reference to the file pointed to by the *file_path* argument. The file pointed to by the *file_path* string must be a supported file format and have the supported file extension at the end of its file name. For example, ".nc" and ".cdf" are supported file extensions for the netCDF data format. The *status* parameter can be either "r" for read only, "w" for read-write or "c" for create. When "c" is set, the file is created if it doesn't exist. If it does exist, an error message is printed and the default missing value for files is returned. When "w" is set, the file exists and the permissions allow reading and writing, and the file is opened for reading and writing. If any of these conditions fail, an error message is reported and the default file missing value is returned. Similarly, if "r" is set, the file must exist and the user must have read permissions on that file; otherwise an error message is printed and the default missing value is returned. See the ismissing function on how to detect the returned missing value in a program.

(See Supported data format information for a list of supported data formats, the appropriate file extensions and other specific information on each supported format.)

---

all

This function returns True if all the elements of the input parameter can be evaluated as True.

Synopsis

```
function all(
        logical_array : logical
)
```

Arguments

logical_array
: A logical array of any dimension.

Description

The **all** function returns a single logical scalar value only if all of the non-missing value elements of the logical input parameter are True, or can be evaluated as True. This function is useful for changing multidimensional logical results from logical expressions into a single scalar value, which is required by the if statement. All numeric values can be coerced to a logical value. Values that are 0 are interpreted as False and values that are non-zero are interpreted as True.

---

any

This function returns the scalar value True if any of the values of its parameter are True.

Synopsis

```
function any(
        logical_array : logical
)
```

Arguments

logical_array
    A logical array of any dimension.

Description

The **any** function returns a single logical scalar value if any of the non-missing value elements of the *logical_array* parameter are True, or can be evaluated as True. This function is useful for changing multidimensional results from logical expressions into a single scalar value, which is required by the if statement. All numeric values can be coerced to a logical value. Values that are 0 are interpreted as False and values that are non-zero are interpreted as True.

---

asciiread

This function is used to read in files that contain ASCII representations of basic data types.

Synopsis

```
function asciiread(
        filepath[1]:string,
        dimensions[*]:integer,
        datatype[1]:string
)
```

Arguments

filepath

Path needed to locate ASCII file.

dimensions

An array specifying the dimensions of the data to be read or -1 if dimensions are unknown.

datatype

A string representing the type of the data being read.

Description

The **asciiread** function is used to read in ascii data. For numeric types, all other characters are ignored and the numbers are read in the order they appear in the input file. For string data, each line of the file is read as a string. For character data, the file is read byte-by-byte.

An error message is generated whenever the number of elements read is less than the number implied by the *dimensions* parameter.

If -1 is given for the dimension sizes, all values in the file will be read into a single dimension variable. The dimension size of this variable will be equal to the number of elements in the file.

Example

If you have an ASCII data file called "data.asc" that looks like this:

```
 1.  2.  3.
 4.  5.  6.
 7.  8.  9.
10. 11. 12.
13. 14. 15.
```

then you can read it various ways as shown with the following ncl script:

```
begin
  z1 = asciiread("data.asc",(/5,3/),"float")
  z2 = asciiread("data.asc",(/4,2/),"float")
end
```

The results of `z1` and `z2` would be:

```
z1(0,0) = 1.
z1(0,1) = 2.
z1(0,2) = 3.
z1(1,0) = 4.
z1(1,1) = 5.
z1(1,2) = 6.
z1(2,0) = 7.
.
.
.
z1(4,0) = 13.
```

```
z1(4,1) = 14.

z2(0,0) = 1.
z2(0,1) = 2.
z2(1,0) = 3.
z2(1,1) = 4.
z2(2,0) = 5.
z2(2,1) = 6.
z2(3,0) = 7.
z2(3,1) = 8.
```

Example of using -1:

```
begin
        z3 = asciiread("data.asc",-1,"float")
end

z3(0) = 1.
z3(1) = 2.
z3(2) = 3.
.
.
.
```

---

asciiwrite

This procedure is used create an ascii text file of NCL numeric data type.

Synopsis

```
        procedure asciiwrite(
                filepath[1]:string,
                value
        )
```

Arguments

filepath
: Path for new ascii file.

value
: Any NCL value of any dimensionality.

Description

The **asciiwrite** function is used to create an ascii data file from an NCL variable. Each element of the *value* parameter is written in its string representation followed by a return. The data are ordered in their row x column internal ordering.

---

asin

This function returns the inverse sine of the input in radians.

Synopsis

```
function asin(
        value:numeric
)
```

Arguments

value

An array of one or more values of any dimension between -1 and 1 inclusive.

Description

The **asin** function returns a floating-point array with the same dimensions as the input parameter *value*. The inverse sine is computed for each element of *value* in **radians**. Return value type is double if double value passed in. Missing values are ignored.

---

atan

This function returns the inverse tangent of the input in radians.

Synopsis

```
function atan(
        value:numeric
)
```

Arguments

value

An array of one or more values of any dimension between -1 and 1 inclusive.

Description

The **atan** function returns a floating-point array with the same dimensions as the input parameter *value*. The inverse tangent is computed for each element of *value* in **radians**. Return value is double if double passed in. Missing values are ignored.

---

atan2

Returns the arctanget of y/x in radians.

Synopsis

```
function atan2(
        x:numeric,
        y:numeric
)
```

Arguments

x

An array of one or more values of any dimensionality

y

An array of one or more values with the same dimensionality as *x*

Description

The *atan2* routine returns the arctangent of y/x in the range -pi to pi using the signs of both arguments to determine the quadrant of the return value. The output dimensionality is the same as the input parameters and all missing values are ignored. Return value is of type double if double values are passed in.

---

attsetvalues

Given a list of HLU objects and a variable containing attributes which are resources, this procedure applies the resources to the objects.

Synopsis

```
procedure attsetvalues (
        object       : graphic,
        resources[1] : logical
)
```

Arguments

object
: The list of objects to apply the resources to.

resources
: A logical variable with the resources as attributes of this variable.

Description

Given a list of HLU objects and a variable containing attributes which are resources, *attsetvalues* builds a resource list and calls setvalues. This is useful since many resources must be set together at the same time, like "vpWidthF" and "vpHeightF".

---

avg

Computes the average of all the input points regardless of dimensionality.

Synopsis

```
function avg(
        x : numeric
)
```

Arguments

x

An array of one or more numeric values of any dimensionality.

Description

Returns the average of the input regardless of dimensionality. *avg* ignores missing values (*x*@_FillValue). To determine the number of data points used to calculate the average use:

```
N = num(.not.ismissing(x))
```

Example

The following is a simplistic example that demonstrates how *avg* works. Try running this example and print out the values to get a better idea of how *avg* works.

```
begin
;
; Create example array
;
  a = (/ (/1,2,3/), (/4,5,6/), (/7,-999,9/)/)
  a@_FillValue = -999

;
; Compute average
;
  average = avg(a)
  n = num(.not.ismissing(a))
  print(average)
  print(n)

end
```

---

bsearch

This function is not implemented yet.

Synopsis

```
function bsearch(
)
```

Arguments

ARG1
: ARG1 ...

ARG2
: ARG2 ...

Description

cbinread

This function is used to read in binary files that have been written using the C block I/O function write.

Synopsis

```
function cbinread(
        filepath[1]:string,
        dimensions[*]:integer,
        datatype[1]:string
)
```

Arguments

filepath
: Path needed to locate binary file.

dimensions
: An array specifying the dimensions of the data to be read or the value -1.

datatype
: A string representing the type of the data being read.

Description

The **cbinread** function is used to read in binary data. The data must be read on the same architecture machine as it was written. If the dimensions specified specify a size less than the total file size, only the first *n* bytes of the file will be read in, where *n* is the product of the dimensions times the size of the specified type. If the size implied by the dimension sizes is greater than the file size, then the remaining space in the returned value is padded with the default missing value for the specified type. In both of these cases, warnings are printed. Note there is no way for NCL to determine if the type of file and the *datatype* parameter represent the same type. If the value of -1 is used for *dimensions* then the entire file is read into a single dimension variable with a dimension length equal to the number of elements in the file.

cbinwrite

This procedure is used create a binary file in raw C block I/O format from and NCL numeric data type.

Synopsis

```
procedure cbinwrite(
        filepath[1]:string,
        value:numeric
)
```

Arguments

filepath
: Path of output file

value
: A numeric value of any dimensionality.

Description

The **cbinwrite** function is used to create a binary data file from an NCL numeric variable. The elements of *value* are written to the file in row x column order using the machine's native binary format.

ceil

This function returns the smallest integral value larger than the input.

Synopsis

```
function ceil(
        value:numeric
)
```

Arguments

value
: An array of one or more values of any dimension.

Description

For each element in the *value* parameter, the smallest integral value larger than the element is returned. The output dimensions are identical to the *value* parameter. Missing values in the input are ignored and propagated through to the output. Return value is of type double if values of type double are passed in.

chiinv

Function for evaluating the inverse chi-squared distribution function.

Synopsis

```
function chiinv(
    p   : float,
    df  : float
)
```

Arguments

p
: The integral from 0 to X of the chi-square distribution. [0<*p*<1]

df
: degrees of freedom of the chi-square distribution. (0, +infinity)

Description

Calculates the upper integration of the non-central chi-square distribution. This gives the same answers as IMSL's *chiin* function.

Example

```
begin
  p  = 0.99
  df = 2.
  x  = chiinv (p,df)
  print ("p="+p+"   df="+df+"   x="+x)

  df = 64.
  x  = chiinv (p,df)
  print ("p="+p+"   df="+df+"   x="+x)

end
```

The result is:

```
(0)     p=0.99   df=2    x= 9.21034
(0)     p=0.99   df=64   x=93.2169
```

---

csa1s

`csa1s` calculates an approximating cubic spline for one-dimensional input data. If you want to weight the input data values, calculate derivatives, or handle data sparse areas specially, then you will need to use csa1xs.

Note: This function is only available in version 4.1.1 of NCL. If your site is licensed for version 4.1, then you should receive version 4.1.1 automatically.

Synopsis

```
    function csa1s(
        xi[*]    : float,
        yi[*]    : float,
        knots[1] : integer
        xo[*]    : float
    )
```

Arguments

xi

A one-dimensional array of any size containing the X coordinates of the input data points.

yi

A one-dimensional array of the same size as *xi* containing the Y coordinates of the input data points.

knots

The number of knots to be used in constructing the approximation spline. *knots* must be at least 4. The larger the value for *knots*, the closer the approximated curve will come to passing through the input function values.

xo

A one-dimensional array containing the X coordinates of the output curve.

Return value

`csa1s` returns an array containing the calculated functional values. The returned value has the same size as *xo* and contains functional values for each element of *xo*.

Description

`csa1s` is in the csagrid package - a software package that implements a cubic spline approximation algorithm to fit a function to input data. The input for the approximation is a set of randomly-spaced data. These data may be one-dimensional, two-dimensional, or three-dimensional.

The general documentation for csagrid contains several complete examples for entries in the csagrid package.

Example

```
begin

;----------- Define original data ---------------------------------------

  xi = (/0.0, 0.1,  0.2,  0.3, 0.5,  0.6, 0.65,  0.8,  0.9, 1./)
  yi = (/0.0, 0.8, -0.9, -0.9, 0.9,  1.0, 0.90, -0.8, -0.8, 0./)

  npts   = 101
  xo     = fspan(0.0,1.0,npts)  ; Create the output X coordinate array.

;--- Calculate approximated function values using differing number of knots.

  knots = 4
  yo = csa1s(xi,yi,knots,xo)

end
```

csa1xs

`csa1xs` calculates an approximating cubic spline for one-dimensional input data. c_csa1xs is called if you want to weight the input data values, calculate derivatives, or handle data sparse areas specially. If you do not want to do any of these three things, then use c_csa1s.

Note: This function is only available in version 4.1.1 of NCL. If your site is licensed for version 4.1, then you should receive version 4.1.1 automatically.

Synopsis

```
function csa1xs(
    xi[*]     : float,
    yi[*]     : float,
    wts[*]    : float,
    knots[1]  : integer
    smth[1]   : float
    nderiv[1] : float
    xo[*]     : float
)
```

Arguments

xi
: A one-dimensional array of any size containing the X coordinates of the input data points.

yi
: A one-dimensional array of the same size as *xi* containing the Y coordinates of the input data points.

wts
: An array containing weights for the *yi* values at the input *xi* values, that is, *wts*(k) is a weight for the value of *yi*(k) for k=0,dimsizes(*xi*)-1. If you do not desire to weight the input *yi* values, then set *wts* to -1, and in that case *wts* can be a scalar. The weights in the *wts* array are relative and may be set to any non-negative value. When `csa1xs` is called, the weights are summed and the individual weights are normalized so that the weight sum is unity.

knots
: The number of knots to be used in constructing the approximation spline. *knots* must be at least 4. The larger the value for *knots*, the closer the approximated curve will come to passing through the input function values.

smth
: A parameter that controls extrapolation into data sparse regions. If *smth* is zero, then nothing special is done in data sparse regions. A good first choice for *smth* is 1.

nderiv

Specifies whether you want functional values (*nderiv*=0), first derivative values (*nderiv*=1), or second derivative values (*nderiv*=2).

xo

A one-dimensional array containing the X coordinates of the output curve.

Return value

`csa1xs` returns an array containing the calculated functional values. The returned value has the same size as *xo* and contains functional values for each element of *xo*.

Description

`csa1xs` is in the csagrid package - a software package in the ngmath library that implements a cubic spline approximation algorithm to fit a function to input data. The input for the approximation is a set of randomly-spaced data. These data may be one-dimensional, two-dimensional, or three-dimensional.

The general documentation for csagrid contains several complete examples for entries in the csagrid package.

Example

```
begin

;----------- Define original data ---------------------------------------

  xi  = (/0.0, 0.1,  0.2,  0.3, 0.5,  0.6, 0.65,  0.8,  0.9, 1./)
  yi  = (/0.0, 0.8, -0.9, -0.9, 0.9,  1.0, 0.90, -0.8, -0.8, 0./)

  npts   = 101
  xo     = fspan(0.0,1.0,npts)  ; Create the output X coordinate array.

;--- Calculate approximated first derivative values.

  knots = 4
  wts = -1.
  smth = 0.
  nderiv = 1
  yo = csa1xs(xi,yi,wts,knots,smth,nderiv,xo)

end
```

csa2ls

csa2ls is called to find values for an approximating cubic spline for two-dimensional input data at a list of specified coordinates. If you want to weight the input data values, calculate derivatives, or handle data sparse areas specially, then you will need to use csa2lxs.

Note: This function is only available in version 4.1.1 of NCL. If your site is licensed for version 4.1, then you should receive version 4.1.1 automatically.

Synopsis

```
function csa2ls(
    xi[*]    : float,
    yi[*]    : float,
    zi[*]    : float,
    knots[2] : integer
    xo[*]    : float
    yo[*]    : float
)
```

Arguments

xi
: A one-dimensional array of any size containing the X coordinates of the input data points.

yi
: A one-dimensional array of the same size as *xi* containing the Y coordinates of the input data points.

zi
: A one-dimensional array of the same size as *xi* and *yi* containing the functional values at the input data coordinates given by *xi* and *yi*. *zi*[k] is the input function value at (*xi*[k],*yi*[k]) for k=0 to dimsizes(*xi*)-1.

knots
: The number of knots to be used in constructing the approximating surface. *knots*(0) and *knots*(1) must both be at least 4. The larger the value for *knots*, the closer the approximated surface will come to passing through the input function values.

xo
: A one-dimensional array containing the X coordinates of a list of points where functional values are desired. *xo* can be a single point.

yo
: A one-dimensional array containing the Y coordinates of a list of points where functional values are desired. *yo* can be a single point.

Return value

`csa2ls` returns a one-dimensional array containing the calculated functional values. If *zo* is the returned value, then *zo* has the same size as *xo* and *yo* and *zo*(i) contains the functional value at coordinate (*xo*(i),*yo*(i)) for i=0,dimsizes(*xo*)-1.

Description

`csa2ls` is in the csagrid package - a software package that implements a cubic spline approximation algorithm to fit a function to input data. The input for the approximation is a set of randomly-spaced data. These data may be one-dimensional, two-dimensional, or three-dimensional.

The general documentation for csagrid contains several complete examples for entries in the csagrid package.

Example

```
begin

;
;  Create the input arrays.
;
  xmin = -1.4
  xmax =  1.4
  ymin = -1.2
  ymax =  1.2

  ndata = 500
  xi = new(ndata,float)
  yi = new(ndata,float)
  zi = new(ndata,float)

  do i=0,ndata-1
    xi(i) = xmin + (xmax-xmin)*rand()/32767.
    yi(i) = ymin + (ymax-ymin)*rand()/32767.
    zi(i) = xi(i) + yi(i)
    t1 = 1.0/((fabs(xi(i)-0.1))^2.75 + fabs(yi(i))^2.75 + 0.09)
    t2 = 1.0/((fabs(xi(i)+0.1))^2.75 + fabs(yi(i))^2.75 + 0.09)
    zi(i) = 0.3*(zi(i)+t1-t2)
  end do

;
;  Find an approximated value at a single point.
;
  knots = (/10,10/)
  xo = 0.
  yo = 0.
  zo = csa2ls(xi,yi,zi,knots,xo,yo)
end
```

---

csa2lxs

csa2lxs is called to find values for an approximating cubic spline for two-dimensional input data at a list of specified coordinates. csa2xs is called if you want to weight the input data values, calculate derivatives, or handle data sparse areas specially. If you do not want to do any of these three things, then use csa2ls.

Note: This function is only available in version 4.1.1 of NCL. If your site is licensed for version 4.1, then you should receive version 4.1.1 automatically.

Synopsis

```
function csa2lxs(
    xi[*]     : float,
    yi[*]     : float,
    zi[*]     : float,
    wts[*]    : float,
    knots[2]  : integer
    smth[1]   : float
    nderiv[2] : integer
    xo[*]     : float
    yo[*]     : float
)
```

Arguments

xi

A one-dimensional array of any size containing the X coordinates of the input data points.

yi

A one-dimensional array of the same size as *xi* containing the Y coordinates of the input data points.

zi

A one-dimensional array of the same size as *xi* and *yi* containing the functional values at the input data coordinates given by *xi* and *yi*. *zi*[k] is the input function value at (*xi*[k],*yi*[k]) for k=0 to dimsizes(*xi*)-1.

wts

An array containing weights for the *zi* values at the input *xi* and *yi* values, that is, *wts*(k) is a weight for the value of *zi*(k) for k=0,dimsizes(*xi*)-1. If you do not desire to weight the input *zi* values, then set *wts* to -1, and in that case *wts* can be a scalar. The weights in the *wts* array are relative and may be set to any non-negative value. When `csa2lxs` is called, the weights are summed and the individual weights are normalized so that the weight sum is unity.

knots

The number of knots to be used in constructing the approximating values. *knots*(0) and *knots*(1) must both be at least 4. The larger the value for *knots*, the closer the approximated surface will come to passing through the input function values.

smth

A parameter that controls extrapolation into data sparse regions. If *smth* is zero, then nothing special is done in data sparse regions. A good first choice for *smth* is 1.

nderiv

Specifies whether you want functional values (=0), first derivative values (=1), or second derivative values (=2) in each of the two coordinate directions.

xo

A one-dimensional array containing the X coordinates of a list of points where functional values are desired. *xo* can be a single point.

yo

A one-dimensional array containing the Y coordinates of a list of points where functional values are desired. *yo* can be a single point.

Return value

`csa2lxs` returns a one-dimensional array containing the calculated functional values. The returned value has the same size as *xo* and *yo*. If *zo* is the returned value, then *zo*(i) contains the functional value at coordinate (*xo*(i),*yo*(i)) for i=0,dimsizes(*xo*)-1.

Description

`csa2lxs` is in the csagrid package - a software package that implements a cubic spline approximation algorithm to fit a function to input data. The input for the approximation is a set of randomly-spaced data. These data may be one-dimensional, two-dimensional, or three-dimensional.

The general documentation for csagrid contains several complete examples for entries in the csagrid package.

Example

```
begin

;
;  Create the input arrays.
;
  xmin = -1.4
  xmax =  1.4
  ymin = -1.2
  ymax =  1.2

  ndata = 500
  xi = new(ndata,float)
  yi = new(ndata,float)
  zi = new(ndata,float)

  do i=0,ndata-1
```

```
    xi(i) = xmin + (xmax-xmin)*rand()/32767.
    yi(i) = ymin + (ymax-ymin)*rand()/32767.
    zi(i) = xi(i) + yi(i)
    t1 = 1.0/((fabs(xi(i)-0.1))^2.75 + fabs(yi(i))^2.75 + 0.09)
    t2 = 1.0/((fabs(xi(i)+0.1))^2.75 + fabs(yi(i))^2.75 + 0.09)
    zi(i) = 0.3*(zi(i)+t1-t2)
  end do

;
;  Find an approximated value for the second order mixed partial at
;  a single point.
;
  knots = (/10,10/)
  xo = 0.
  yo = 0.
  knots = 4
  wts = -1.
  smth = 0.
  nderiv = (/1,1/)
  zo = csa2lxs(xi,yi,zi,wts,knots,smth,nderiv,xo,yo)
end
```

---

csa2s

`csa2s` calculates an approximating cubic spline for two-dimensional input data. If you want to weight the input data values, calculate derivatives, or handle data sparse areas specially, then you will need to use csa2xs.

Note: This function is only available in version 4.1.1 of NCL. If your site is licensed for version 4.1, then you should receive version 4.1.1 automatically.

Synopsis

```
    function csa2s(
        xi[*]    : float,
        yi[*]    : float,
        zi[*]    : float,
        knots[2] : integer
        xo[*]    : float
        yo[*]    : float
    )
```

Arguments

xi

A one-dimensional array of any size containing the X coordinates of the input data points.

yi

A one-dimensional array of the same size as *xi* containing the Y coordinates of the input data points.

zi

A one-dimensional array of the same size as *xi* and *yi* containing the functional values at the input data coordinates given by *xi* and *yi*. *zi*[k] is the input function value at (*xi*[k],*yi*[k]) for k=0 to dimsizes(*xi*)-1.

knots

The number of knots to be used in constructing the approximating surface. *knots*(0) and *knots*(1) must both be at least 4. The larger the value for *knots*, the closer the approximated surface will come to passing through the input function values.

xo

A one-dimensional array containing the X coordinates of the output surface.

yo

A one-dimensional array containing the Y coordinates of the output surface.

Return value

`csa2s` returns a two-dimensional array containing the calculated functional values. The first dimension of the returned value has the same size as *xo* and the second dimension of the returned value has the same size as *yo*. If *zo* is the returned value, then *zo*(i,j) contains the functional value at coordinate (*xo*(i),*yo*(j)).

Description

`csa2s` is in the csagrid package - a software package that implements a cubic spline approximation algorithm to fit a function to input data. The input for the approximation is a set of randomly-spaced data. These data may be one-dimensional, two-dimensional, or three-dimensional.

The general documentation for csagrid contains several complete examples for entries in the csagrid package.

Example

```
begin

;
;  Create the input arrays.
;
  xmin = -1.
  xmax =  1.
  ymin = -1.
  ymax =  1.
```

```
  nx = 29
  ny = 25

  ndata = 1000
  xi = new(ndata,float)
  yi = new(ndata,float)
  zi = new(ndata,float)

;
;  Generate input data using the function f(x,y) = y**2 - 0.5*y*x**2
;
  do i=0,ndata-1
      xi(i) = xmin + (xmax-xmin)*rand()/32767.
      yi(i) = ymin + (ymax-ymin)*rand()/32767.
      zi(i) = yi(i)*yi(i) - 0.5*xi(i)*xi(i)*yi(i)
  end do

;
;  Set up the output grid.
;
  xo = fspan(xmin,xmax,nx)
  yo = fspan(ymin,ymax,ny)

  knots = (/4,4/)
;
;  Calculate the approximated function values.
;
  yo = csa2s(xi,yi,zi,knots,xo,yo)
end
```

---

csa2xs

`csa2xs` calculates an approximating cubic spline for two-dimensional input data. csa2xs is called if you want to weight the input data values, calculate derivatives, or handle data sparse areas specially. If you do not want to do any of these three things, then use csa2s.

Note: This function is only available in version 4.1.1 of NCL. If your site is licensed for version 4.1, then you should receive version 4.1.1 automatically.

Synopsis

```
    function csa2xs(
        xi[*]     : float,
        yi[*]     : float,
        zi[*]     : float,
        wts[*]    : float,
        knots[2]  : integer
        smth[1]   : float
        nderiv[2] : float
        xo[*]     : float
        yo[*]     : float
```

)

Arguments

xi
: A one-dimensional array of any size containing the X coordinates of the input data points.

yi
: A one-dimensional array of the same size as *xi* containing the Y coordinates of the input data points.

zi
: A one-dimensional array of the same size as *xi* and *yi* containing the functional values at the input data coordinates given by *xi* and *yi*. *zi*[k] is the input function value at (*xi*[k],*yi*[k]) for k=0 to dimsizes(*xi*)-1.

wts
: An array containing weights for the *zi* values at the input *xi* and *yi* values, that is, *wts*(k) is a weight for the value of *zi*(k) for k=0,dimsizes(*xi*)-1. If you do not desire to weight the input *zi* values, then set *wts* to -1, and in that case *wts* can be a scalar. The weights in the *wts* array are relative and may be set to any non-negative value. When `csa2xs` is called, the weights are summed and the individual weights are normalized so that the weight sum is unity.

knots
: The number of knots to be used in constructing the approximating surface. *knots*(0) and *knots*(1) must both be at least 4. The larger the value for *knots*, the closer the approximated surface will come to passing through the input function values.

smth
: A parameter that controls extrapolation into data sparse regions. If *smth* is zero, then nothing special is done in data sparse regions. A good first choice for *smth* is 1.

nderiv
: Specifies whether you want functional values (=0), first derivative values (=1), or second derivative values (=2) in each of the two coordinate directions.

xo
: A one-dimensional array containing the X coordinates of the output surface.

yo
: A one-dimensional array containing the Y coordinates of the output surface.

Return value

`csa2xs` returns a two-dimensional array containing the calculated functional values. The first dimension of the returned value has the same size as *xo* and the second dimension of the returned value has the same size as *yo*. If *zo* is the returned value, then *zo*(i,j) contains the functional values at coordinate (*xo*(i),*yo*(j)).

Description

`csa2xs` is in the csagrid package - a software package that implements a cubic spline approximation algorithm to fit a function to input data. The input for the approximation is a set of randomly-spaced data. These data may be one-dimensional, two-dimensional, or three-dimensional.

The general documentation for csagrid contains several complete examples for entries in the csagrid package.

Example

```
begin

;
;  Create the input arrays.
;
  xmin = -1.
  xmax =  1.
  ymin = -1.
  ymax =  1.

  nx = 29
  ny = 25

  ndata = 1000
  xi = new(ndata,float)
  yi = new(ndata,float)
  zi = new(ndata,float)

;
;  Generate input data using the function f(x,y) = y**2 - 0.5*y*x**2
;
  do i=0,ndata-1
      xi(i) = xmin + (xmax-xmin)*rand()/32767.
      yi(i) = ymin + (ymax-ymin)*rand()/32767.
      zi(i) = yi(i)*yi(i) - 0.5*xi(i)*xi(i)*yi(i)
  end do

;
;  Set up the output grid.
;
  xo = fspan(xmin,xmax,nx)
  yo = fspan(ymin,ymax,ny)

  knots = (/4,4/)
;
;  Calculate the approximated function values.
;
  wts = -1.
  smth = 0.
  nderiv = (/1,2/)
  yo = csa2xs(xi,yi,zi,wts,knots,smth,nderiv,xo,yo)
end
```

---

csa3ls

csa3ls is called to find values for an approximating cubic spline for three-dimensional input data at a list of specified coordinates. If you want to weight the input data values, calculate derivatives, or handle data sparse areas specially, then you will need to use csa3lxs.

Note: This function is only available in version 4.1.1 of NCL. If your site is licensed for version 4.1, then you should receive version 4.1.1 automatically.

Synopsis

```
function csa3ls(
    xi[*]    : float,
    yi[*]    : float,
    zi[*]    : float,
    ui[*]    : float,
    knots[3] : integer
    xo[*]    : float
    yo[*]    : float
    zo[*]    : float
)
```

Arguments

xi

A one-dimensional array of any size containing the X coordinates of the input data points.

yi

A one-dimensional array of the same size as *xi* containing the Y coordinates of the input data points.

zi

A one-dimensional array of the same size as *xi* and *yi* containing the Z coordinates of the input data points.

ui

A one-dimensional array of the same size as *xi*, *yi*, and *zi* containing the functional values at the input data coordinates given by *xi*, *yi*, and *zi*. That is *ui*[k] is the input function value at (*xi*[k],*yi*[k],*zi*[k]) for k=0 to dimsizes(*xi*)-1.

knots

The number of knots to be used in constructing the approximating surface. *knots*(0), *knots*(1), and *knots*(2) must all be at least 4. The larger the value for *knots*, the closer the approximated surface will come to passing through the input function values.

xo

A one-dimensional array containing the X coordinates of a list of points where functional values are desired. *xo* can be a single point.

yo

A one-dimensional array containing the Y coordinates of a list of points where functional values are desired. *yo* can be a single point.

zo

A one-dimensional array containing the Z coordinates of a list of points where functional values are desired. *zo* can be a single point.

Return value

`csa3ls` returns a one-dimensional array containing the calculated functional values. If *uo* is the returned value, then *uo* has the same size as *xo*, *yo*, and *zo*. That is, *uo*(i) contains the functional value at coordinate (*xo*(i), *yo*(i), *zo*(i)) for i=0,dimsizes(*xo*)-1.

Description

`csa3ls` is in the csagrid package - a software package that implements a cubic spline approximation algorithm to fit a function to input data. The input for the approximation is a set of randomly-spaced data. These data may be one-dimensional, two-dimensional, or three-dimensional.

The general documentation for csagrid contains several complete examples for entries in the csagrid package.

Example

```
begin

;
;  Create the input arrays.
;
  xmin = -2.
  xmax =  2.
  ymin = -2.
  ymax =  2.
  zmin = -2.
  zmax =  2.

  nx = 21
  ny = 21
  nz = 21

  ndata = 1000
  xi = new(ndata,float)
  yi = new(ndata,float)
  zi = new(ndata,float)
  ui = new(ndata,float)
```

```
  do i=0,ndata-1
      xi(i) = xmin + (xmax-xmin)*rand()/32767.
      yi(i) = ymin + (ymax-ymin)*rand()/32767.
      zi(i) = zmin + (zmax-zmin)*rand()/32767.
      ui(i) = xi(i)*xi(i) + yi(i)*yi(i) + zi(i)*zi(i)
  end do

;
;  Calculate an approximation value at a single point.
;
  knots = (/4,4,4/)
  xo = 1.5
  yo = 1.0
  zo = 0.5
  uo = csa3ls(xi,yi,zi,ui,knots,xo,yo,zo)
end
```

---

csa3lxs

csa3lxs is called to find values for an approximating cubic spline for three-dimensional input data at a list of specified coordinates. csa3xs is called if you want to weight the input data values, calculate derivatives, or handle data sparse areas specially. If you do not want to do any of these three things, then use csa3ls.

Note: This function is only available in version 4.1.1 of NCL. If your site is licensed for version 4.1, then you should receive version 4.1.1 automatically.

Synopsis

```
    function csa3lxs(
        xi[*]     : float,
        yi[*]     : float,
        zi[*]     : float,
        ui[*]     : float,
        wts[*]    : float,
        knots[3]  : integer
        smth[1]   : float
        nderiv[3] : integer
        xo[*]     : float
        yo[*]     : float
        zo[*]     : float
    )
```

Arguments

xi

A one-dimensional array of any size containing the X coordinates of the input data points.

yi

A one-dimensional array of the same size as *xi* containing the Y coordinates of the input data points.

zi

A one-dimensional array of the same size as *xi* and *yi* containing the Z coordinates of the input data points.

ui

A one-dimensional array of the same size as *xi*, *yi*, and *zi* containing the functional values at the input data coordinates given by *xi*, *yi*, and *zi*. That is *ui*[k] is the input function value at (*xi*[k],*yi*[k],*zi*[k]) for k=0 to dimsizes(*xi*)-1.

wts

An array containing weights for the *ui* values at the input *xi*, *yi*, and *zi* values. That is, *wts*(k) is a weight for the value of *ui*(k) for k=0,dimsizes(*xi*)-1. If you do not desire to weight the input *zi* values, then set *wts* to -1, and in that case *wts* can be a scalar. The weights in the *wts* array are relative and may be set to any non-negative value. When `csa3lxs` is called, the weights are summed and the individual weights are normalized so that the weight sum is unity.

knots

The number of knots to be used in constructing the approximating values. *knots*(0), *knots*(1), and *knots*(2) must all be at least 4. The larger the value for *knots*, the closer the approximated surface will come to passing through the input function values.

smth

A parameter that controls extrapolation into data sparse regions. If *smth* is zero, then nothing special is done in data sparse regions. A good first choice for *smth* is 1.

nderiv

Specifies whether you want functional values (=0), first derivative values (=1), or second derivative values (=2) in each of the three coordinate directions.

xo

A one-dimensional array containing the X coordinates of a list of points where functional values are desired. *xo* can be a single point.

yo

A one-dimensional array containing the Y coordinates of a list of points where functional values are desired. *yo* can be a single point.

zo

A one-dimensional array containing the Z coordinates of a list of points where functional values are desired. *zo* can be a single point.

Return value

`csa3lxs` returns a one-dimensional array containing the calculated functional values. If *uo* is the returned value, then *uo* has the same size as *xo*, *yo*, and *zo*. That is, *uo*(i) contains the functional value at coordinate (*xo*(i), *yo*(i), *zo*(i)) for i=0,dimsizes(*xo*)-1.

Description

`csa3lxs` is in the csagrid package - a software package that implements a cubic spline approximation algorithm to fit a function to input data. The input for the approximation is a set of randomly-spaced data. These data may be one-dimensional, two-dimensional, or three-dimensional.

The general documentation for csagrid contains several complete examples for entries in the csagrid package.

Example

```
begin

;
;  Create the input arrays.
;
  xmin = -2.
  xmax =  2.
  ymin = -2.
  ymax =  2.
  zmin = -2.
  zmax =  2.

  nx = 21
  ny = 21
  nz = 21

  ndata = 1000
  xi = new(ndata,float)
  yi = new(ndata,float)
  zi = new(ndata,float)
  ui = new(ndata,float)

  do i=0,ndata-1
      xi(i) = xmin + (xmax-xmin)*rand()/32767.
      yi(i) = ymin + (ymax-ymin)*rand()/32767.
      zi(i) = zmin + (zmax-zmin)*rand()/32767.
      ui(i) = xi(i)*xi(i) + yi(i)*yi(i) + zi(i)*zi(i)
  end do

;
;  Calculate an approximation value for the first order partial
;  with respect to X at a single point.
;
  knots = (/4,4,4/)
  wts = -1.
  smth = 0.
  nderiv = (/0,1,0/)
  xo = 1.5
  yo = 1.0
  zo = 0.5
  uo = csa3lxs(xi,yi,zi,ui,wts,knots,smth,nderiv,xo,yo,zo)
end
```

---

csa3s

`csa3s` calculates an approximating cubic spline for three-dimensional input data. If you want to weight the input data values, calculate derivatives, or handle data sparse areas specially, then you will need to use csa3xs.

Note: This function is only available in version 4.1.1 of NCL. If your site is licensed for version 4.1, then you should receive version 4.1.1 automatically.

Synopsis

```
function csa3s(
    xi[*]     : float,
    yi[*]     : float,
    zi[*]     : float,
    ui[*]     : float,
    knots[3] : integer
    xo[*]     : float
    yo[*]     : float
    zo[*]     : float
)
```

Arguments

xi

A one-dimensional array of any size containing the X coordinates of the input data points.

yi

A one-dimensional array of the same size as *xi* containing the Y coordinates of the input data points.

zi

A one-dimensional array of the same size as *xi* and *yi* containing the Z coordinates of the input data points.

ui

A one-dimensional array of the same size as *xi*, *yi*, and *zi* containing the functional values at the input data coordinates given by *xi*, *yi*, and *zi*. That is *ui*[k] is the input function value at (*xi*[k],*yi*[k],*zi*[k]) for k=0 to dimsizes(*xi*)-1.

knots

The number of knots to be used in constructing the approximating surface. *knots*(0) and *knots*(1) must both be at least 4. The larger the value for *knots*, the closer the approximated surface will come to passing through the input function values.

xo

A one-dimensional array of any size containing the X coordinates of the output function.

yo

A one-dimensional array of any size containing the Y coordinates of the output function.

zo

A one-dimensional array of any size containing the Z coordinates of the output function.

Return value

`csa3s` returns a three-dimensional array containing the calculated functional values. The first dimension of the returned value has the same size as *xo*, the second dimension of the returned value has the same size as *yo*, and the third dimension of the returned value has the same size as *zo*. If *uo* is the returned value, then *uo*(i,j,k) contains the functional value at coordinate (*xo*(i),*yo*(j),*zo*(k)).

Description

`csa3s` is in the csagrid package - a software package that implements a cubic spline approximation algorithm to fit a function to input data. The input for the approximation is a set of randomly-spaced data. These data may be one-dimensional, two-dimensional, or three-dimensional.

The general documentation for csagrid contains several complete examples for entries in the csagrid package.

Example

```
begin

;
;  Create the input arrays.
;
  xmin = -2.
  xmax =  2.
  ymin = -2.
  ymax =  2.
  zmin = -2.
  zmax =  2.

  nx = 21
  ny = 21
  nz = 21

  ndata = 1000
  xi = new(ndata,float)
  yi = new(ndata,float)
  zi = new(ndata,float)
  ui = new(ndata,float)

  do i=0,ndata-1
      xi(i) = xmin + (xmax-xmin)*rand()/32767.
      yi(i) = ymin + (ymax-ymin)*rand()/32767.
```

```
      zi(i) = zmin + (zmax-zmin)*rand()/32767.
      ui(i) = xi(i)*xi(i) + yi(i)*yi(i) + zi(i)*zi(i)
  end do

;
;  Set up the output grid.
;
  xo = fspan(xmin,xmax,nx)
  yo = fspan(ymin,ymax,ny)
  zo = fspan(zmin,zmax,nz)

;
;  Calculate the values for the approximating cubic spline.
;
  knots = (/4,4,4/)
  uo = csa3s(xi,yi,zi,ui,knots,xo,yo,zo)
end
```

---

csa3xs

`csa3xs` calculates an approximating cubic spline for three-dimensional input data. csa3xs is called if you want to weight the input data values, calculate derivatives, or handle data sparse areas specially. If you do not want to do any of these three things, then use csa3s.

Note: This function is only available in version 4.1.1 of NCL. If your site is licensed for version 4.1, then you should receive version 4.1.1 automatically.

Synopsis

```
    function csa3xs(
        xi[*]      : float,
        yi[*]      : float,
        zi[*]      : float,
        ui[*]      : float,
        wts[*]     : float,
        knots[3]   : integer
        smth[1]    : float
        nderiv[3] : float
        xo[*]      : float
        yo[*]      : float
        zo[*]      : float
    )
```

Arguments

xi

A one-dimensional array of any size containing the X coordinates of the input data points.

yi

A one-dimensional array of the same size as *xi* containing the Y coordinates of the input data points.

zi

A one-dimensional array of the same size as *xi* and *yi* containing the Z coordinates of the input data points.

ui

A one-dimensional array of the same size as *xi*, *yi*, and *zi* containing the functional values at the input data coordinates given by *xi*, *yi*, and *zi*. That is *ui*[k] is the input function value at (*xi*[k],*yi*[k],*zi*[k]) for k=0 to dimsizes(*xi*)-1.

wts

An array containing weights for the *ui* values at the input *xi*, *yi*, and *zi* values. That is, *wts*(k) is a weight for the value of *ui*(k) for k=0,dimsizes(*xi*)-1. If you do not desire to weight the input *ui* values, then set *wts* to -1, and in that case *wts* can be a scalar. The weights in the *wts* array are relative and may be set to any non-negative value. When `csa3xs` is called, the weights are summed and the individual weights are normalized so that the weight sum is unity.

knots

The number of knots to be used in constructing the approximating spline. *knots*(0), *knots*(1), and *knots*(2) must all be at least 4. The larger the value for *knots*, the closer the approximated surface will come to passing through the input function values.

smth

A parameter that controls extrapolation into data sparse regions. If *smth* is zero, then nothing special is done in data sparse regions. A good first choice for *smth* is 1.

nderiv

Specifies whether you want functional values (=0), first derivative values (=1), or second derivative values (=2) in each of the three coordinate directions.

xo

A one-dimensional array containing the X coordinates of the approximating spline.

yo

A one-dimensional array containing the Y coordinates of the approximating spline.

zo

A one-dimensional array containing the Z coordinates of the approximating spline.

Return value

`csa3xs` returns a three-dimensional array containing the calculated functional values. The first dimension of the returned value has the same size as *xo*, the second dimension of the returned value has the same size as *yo*, and the third dimension of the returned value has the same size as *zo*. If *uo* is the returned value, then *uo*(i,j,k) contains the functional value at coordinate (*xo*(i),*yo*(j),*zo*(k)).

Description

`csa3xs` is in the csagrid package - a software package that implements a cubic spline approximation algorithm to fit a function to input data. The input for the approximation is a set of randomly-spaced data. These data may be one-dimensional, two-dimensional, or three-dimensional.

The general documentation for csagrid contains several complete examples for entries in the csagrid package.

Example

```
begin

;
;  Create the input arrays.
;
  xmin = -2.
  xmax =  2.
  ymin = -2.
  ymax =  2.
  zmin = -2.
  zmax =  2.

  nx = 21
  ny = 21
  nz = 21

  ndata = 1000
  xi = new(ndata,float)
  yi = new(ndata,float)
  zi = new(ndata,float)
  ui = new(ndata,float)

  do i=0,ndata-1
      xi(i) = xmin + (xmax-xmin)*rand()/32767.
      yi(i) = ymin + (ymax-ymin)*rand()/32767.
      zi(i) = zmin + (zmax-zmin)*rand()/32767.
      ui(i) = xi(i)*xi(i) + yi(i)*yi(i) + zi(i)*zi(i)
  end do

;
;  Set up the output grid.
;
  xo = fspan(xmin,xmax,nx)
  yo = fspan(ymin,ymax,ny)
  zo = fspan(zmin,zmax,nz)

;
;  Calculate the values for the approximating cubic spline for the
;  first partial derivative with respect to Y.
;
  knots = (/4,4,4/)
  wts = -1.
  smth = 0.
  nderiv = (/0,1,0/)
  uo = csa3xs(xi,yi,zi,ui,wts,knots,smth,nderiv,xo,yo,zo)
end
```

---

cos

This function computes the cosine from radian input.

Synopsis

```
function cos(
        value:numeric
)
```

Arguments

value
    An array of one or more values of any dimension in radians.

Description

The cosine of each element of the *value* parameter is determined and returned. Missing values are ignored. Returns a floating-point array dimensioned the same as the input parameter. Return values are of type double if input is of type double

cosh

This function computes the hyperbolic cosine from radian input.

Synopsis

```
function cosh(
        value:numeric
)
```

Arguments

value
    An array of one or more values of any dimension in radians.

Description

The hyperbolic cosine of each element of the *value* parameter is determined and returned. Missing values are ignored. Returns a floating-point array dimensioned the same as the input parameter. Return values are of type double if input is of type double.

craybinnumrec

Returns the number of unformatted sequential access Fortran records in a cray binary file.

Synopsis

```
function craybinnumrec(
        path [1]: string
)
```

Arguments

path
: pathname of file

Description

This function returns the number of records in a COS blocked Fortran unformatted sequential access binary file.

craybinrecread

This function allows direct access to COS blocked unformatted sequential access Fortran binary files.

Synopsis

```
function craybinrecread(
        path [1] : string,
```

```
                rec_num [1] : integer,
                dims [*] : integer,
                type [1] : string
        )
```

Arguments

path
: pathname to binary file

rec_num
: record number to read from the file beginning at 0

dims
: a singly dimensioned array of integer values that describe how to shape the data, or -1 if the size of the record is unknown.

type
: string name of the data type of the record.

Description

craybinrecread reads the *rec_num* record of the file *path* and shapes it according to the dimension sizes in parameter. The data type of the record is specified by the parameter *type*. If the size and dimensionality of the record are unknown the value -1 can be used for parameter *dims*. In this case *craybinrecread* will read in the entire record as a singly dimensioned array. The file must be COS blocked.

Example

Example pending

---

cz2ccm

Computes geopotential height for a CCM2 hybrid coordinates.

Note: This function is only available in version 4.1.1 of NCL. If your site is licensed for version 4.1, then you should receive version 4.1.1 automatically.

Synopsis

```
function cz2ccm(
    ps    : float,
    phis  : float,
    tv    : float,
    p0    : float,
    hyam  : float,
    hybm  : float,
    hyai  : float,
    hybi  : float
)
```

Arguments

ps
: input, surface pressure (Pascals). The rightmost dimensions should correspond to latitute (*nlat*) and longitude (*mlon*).

phis
: input, surface geopotential (m^2/s^2). A 2-dimensional array dimensioned *nlat* x *mlon*. **Note**: if the utility "ccm2nc" was used to convert the original CCM History Tape file to netCDF, this field may have a time dimension of length one. In this case, just use the first time stamp: "phis(0,:,:)" as the argument to the function.

tv
: input, virtual temperature. The rightmost three dimensions must correspond to *klev*, *nlat*, and *mlon*.

p0
: scalar, input, hybrid base pressure (Pascals). It not available, use 100000.

hyam
: hybrid coord coefficients for base pressure (layer midpoints; ground to top)

hybm
: hybrid coordinate coefficients for surface pressure (layer midpoints; ground to top)

hyai
: hybrid coordinate coefficients for base pressure (layer interfaces; ground to top)

hybi
: hybrid coordinate coefficients for surface pressure (layer interfaces; ground to top)

Description

`cz2ccm` calculates geopotential heights for a CCM2 hybrid coordinate system. Note that the hybrid coefficients are ordered ground-to-top (which is the conventional way). Missing values are not allowed.

If any of the input arrays have the missing value attribute "_FillValue" set, then a warning message will be printed. **Note**: This is the *exact* routine used within the CCM Processor.

Example 1

This example reads two files. The first one contains information the second file does not contain. The second file contains daily means of numerous variables for one month (30 time steps). The temperature (T) and specific humidities (Q) are read directly from the file (f->T and f->Q) and are used to calculate the virtual temperature: `tv` with dimensions *time* x *lev* x *lat lon*. Invoking `cz2ccm` as indicated below will dynamically return the geopotential heights to `z2` [*time*, *lev*, *lat*, *lon*]. Note that the order of the hybrid coordinates has been temporarily reversed [(::-1)] for the hybrid coefficients to match the input requirements.

```
begin

  diri    = "/fs/scd/home1/shea/ncldata_input/"
  filphis = "01-50.nc"                          ; file with PHIS field
  f       = addfile (diri+filphis+".nc" , "r")
                                             ; strip off various fields
  phis    = f->PHIS                          ; read PHIS to memory
  lat     = f->lat
  lon     = f->lon
  lev     = f->lev
  ilev    = f->ilev
  hyam    = f->hyam
  hybm    = f->hybm
  hyai    = f->hyai
  hybi    = f->hybi
  mlon    = dimsizes(lon)
  nlat    = dimsizes(lat)
  klev    = dimsizes(lev)
  delete (f)                             ; not required but good programming practice

  fili    = (/ "ha0001.nc" /)
  f       = addfile (diri+fili , "r")   ; daily files
  ntimes  = dimsizes(f->time)           ; determine no. time steps (30)

  ps    = f->PS                          ; get sfc pres
  tv    = f->T*(1.+0.61*f->Q)            ; calculate virtual temperature
  p0    = f->P0                          ; get constant

                                         ; calculate z2 at all 30 time steps
  z2 = cz2ccm(ps, phis, tv, p0 \
             ,hyam(::-1), hybm(::-1), hyai(::-1), hybi(::-1) )

end
```

Example 2

This is a variation of the previous example. The goal is to get geopotential heights at specified pressure levels after calculating `z2` at the hybrid levels. This uses the NCL function "vinth2p" to do the vertical interpolation.

```
begin

  [snip part one above]

  fili    = (/ "ha0001.nc" /)
  f       = addfile (diri+fili , "r") ; daily files
  ntimes  = dimsizes(f->time)         ; determine no. time steps (30)
  klvl    = dimsizes(f->lev)
  nlat    = dimsizes(f->lat)
  mlon    = dimsizes(f->lon)

  ps      = f->PS                     ; get sfc pres
  tv      = f->T*(1.+0.61*f->Q)       ; calculate virtual temperature
  p0      = f->P0                     ; get constant

                                      ; calculate Z2 at all 30 time steps
  Z2 = cz2ccm(ps, phis, tv, p0 \      ; (time,lev,lat,lon)
             ,hyam(::-1), hybm(::-1), hyai(::-1), hybi(::-1) )

  pnew = (/ 850., 500., 200.  /)      ; desired output levels [mb]
  knew = dimsizes(pnew)
  P0mb = 1000.                        ; reference pressure [mb]

  Znew = new ( (/ntimes,knew,nlat,mlon/) , float)
  do nt=0,ntimes-1                    ; step thru each time step
     Znew(nt,:,:,:) = vinth2p (Z2(nt,:,:,:) ,hyam, hybm, pnew ,ps, 1 ,P0mb, 2, True)
  end do

end
```

---

day_of_year, days_in_month, monthday, isleapyear

Date functions

Synopsis

```
function day_of_year(
    year  : integer,
    month : integer,
    day   : integer
)

function days_in_month(
    year  : integer,
    month : integer
)

function monthday(
    year : integer,
    day  : integer
)
```

```
function isleapyear(
    year : integer
)
```

Arguments

year
: Array of any dimensionality, each value must be >= 0.

month
: Array of any dimensionality, each value must be an integer from 1 to 12.

day
: Array of any dimensionality. For the `day_of_year` function, *day* represents the day of the month (1 to 31). For the `monthday` function, *day* represents the day of the year (1 to 366).

Description

`day_of_year` returns an integer array (of the same dimensionality as *year*, *day*, and *month*), where each value represents the day of the year (Gregorian calendar). For example, given month 3, day 1, and year 1996, `day_of_year` would return a value of 61.

`days_in_month` returns an integer array (of the same dimensionality as *year* and *month*), where each value represents the number of days in a particular month (Gregorian calendar).

`monthday` returns an integer array (same dimensionality as *year*), where each value represents the concatenated month and day as an integer, (Gregorian calendar). For example, given year 1933 and day 245, `monthday` would return 902 for September 2.

`isleapyear` returns a logical array (with same dimensionality as *year*), where each value is true or false, depending if the corresponding year is a leap year (Gregorian calendar).

Example

```
;
; Every element of 'i' will be false in this case.
;
i = isleapyear((/1997,1993,1900,1990/))

;
; 'j' will be equal to (/61,60/).
;
j = day_of_year((/1996,1990/),(/3,3/),(/1,1/))
```

```
;
; 'k' will be equal to (/29,28,31/)
;
k = days_in_month((/1996,1997,1/),(/2,2,1/))

;
; 'l' will be equal to (/902,1231,1231,301/)
;
l = monthday((/1933,1996,1997,1996/),(/245,366,365,61/))
```

---

delete

This function deletes variables, attributes, and coordinate variables from NCL.

Synopsis

```
        procedure delete(
                data
        )
```

Arguments

data

Any variable of any type or dimensionality.

Description

This function is used to delete variables, attributes, and coordinate variables from NCL. delete does not work for deleting file variables, file attributes, or file coordinate variables. When a variable is passed in, all of the memory associated with the variable's value and its reference in the symbol table are removed. If an attribute or coordinate variable is passed in, it is deleted from the variable it belongs to. Note that if the variable being deleted is an HLU object and the variable contains the only reference to that HLU object, the HLU object will be destroyed. See NhlDestroy for the ability to destroy HLU objects without deleting the variable it points to.

---

dim_avg

For each index of dimensions 0..n-2, this function computes the average of the n-1 dimension

---

Synopsis

```
function dim_avg(
        x : numeric
)
```

Arguments

x

An array of one or more numeric values of any dimensionality

Description

The *dim_avg* function computes the average of all elements of the n-1 dimension for each index of the dimensions 0..n-2. *dim_avg* ignores missing values (*x*@_FillValue). The output dimensionality is the same as the first n-2 dimensions of the input.

Example

The following is a simplistic example that demonstrates how *dim_avg* works. Try running this example and print out the values to get a better idea of how *dim_avg* works.

```
begin
;
;       Create example array
;
        a = onedtond(ispan(1,150,1),(/3,5,10/))
;
;       Assign dimension names
;
        a!0 = "time"
        a!1 = "latitude"
        a!2 = "longitude"
;
;       Compute average of dimension longitude  at all time and latitude points
;
        pxy = dim_avg(a)
;
;       Compute average of dimension time  at all points latitude and longitude poin
;       The following uses Named Subscripting to reorder the input array
;       such that time is the n-1 dimension.
;
        pyz = dim_avg( a(latitude | :, longitude | : , time | :) )

end
```

dim_max

For each index of dimensions 0..n-2, this function computes the maximum of the n-1 dimension

Synopsis

```
        function dim_max(
                x : numeric
        )
```

Arguments

x

An array of one or more numeric values of any dimensionality

Description

The *dim_max* function computes the maximum of all elements of the n-1 dimension for each index of the dimensions 0..n-2. *dim_max* ignores missing values. The output dimensionality is the same as the first n-2 dimensions of the input.

Example

The following is simplistic example that demonstrates how *dim_max* works. Try running this example and print out the values to get a better idea of how *dim_max* works.

```
begin
;
;       Create example array
;
        a = onedtond(ispan(1,150,1),(/3,5,10/))
;
;       Assign dimension names
;
        a!0 = "time"
        a!1 = "latitude"
        a!2 = "longitude"
;
;       Compute maximum of dimension longitude  at all time and latitude points
;
        pxy = dim_max(a)
```

```
;
;       Compute maximum of dimension time  at all points latitude and longitude poin
;       The following uses Named Subscripting to reorder the input array
;       such that time is the n-1 dimension.
;
        pyz = dim_max( a(latitude | :, longitude | : , time | :) )

end
```

---

dim_min

For each index of dimensions 0..n-2, this function computes the minimum of the n-1 dimension

---

Synopsis

```
        function dim_min(
                x : numeric
        )
```

Arguments

x

An array of one or more numeric values of any dimensionality

Description

The *dim_min* function computes the minimum of all elements of the n-1 dimension for each index of the dimensions 0..n-2. *dim_min* ignores missing values. The output dimensionality is the same as the first n-2 dimensions of the input.

Example

The following is simplistic example that demonstrates how *dim_min* works. Try running this example and print out the values to get a better idea of how *dim_min* works.

```
begin
;
;       Create example array
;
        a = onedtond(ispan(1,150,1),(/3,5,10/))
```

```
;
;       Assign dimension names
;
        a!0 = "time"
        a!1 = "latitude"
        a!2 = "longitude"
;
;       Compute minimum of dimension longitude  at all time and latitude points
;
        pxy = dim_min(a)
;
;       Compute minimum of dimension time  at all points latitude and longitude poin
;       The following uses Named Subscripting to reorder the input array
;       such that time is the n-1 dimension.
;
        pyz = dim_min( a(latitude | :, longitude | : , time | :) )

end
```

---

dim_product

For each index of dimensions 0..n-2, this function computes the product of the n-1 dimension.

Synopsis

```
        function dim_product(
                x : numeric
        )
```

Arguments

x

An array of one or more numeric values of any dimensionality

Description

The *dim_product* function computes the product of all elements of the n-1 dimension for each index of the dimensions 0..n-2. *dim_product* ignores missing values. The output dimensionality is the same as the first n-2 dimensions of the input.

Example

The following is simplistic example that demonstrates how *dim_product* works. Try running this example and print out the values to get a better idea of *dim_product*.

```
begin
;
;       Create example array
;
        a = onedtond(ispan(1,150,1),(/3,5,10/))
;
;       Assign dimension names
;
        a!0 = "x"
        a!1 = "y"
        a!2 = "z"
;
;       Compute product of dimension z at all points x and y
;
        pxy = dim_product(a)
;
;       Compute product of dimension x at all points y and z
;       The following uses Named Subscripting to reorder the input array
;       such that x is the n-1 dimension.
;
        pyz = dim_product( a(y | :, z | : , x | :) )

end
```

---

dim_stddev

For each index of dimensions 0..n-2, this function computes the standard deviation of the n-1 dimension.

Synopsis

```
        function dim_stddev(
                x : numeric
        )
```

Arguments

x

An array of one or more numeric values of any dimensionality

Description

The *dim_stddev* function computes the standard deviation of all elements of the n-1 dimension for each index of the dimensions 0..n-2. *dim_stddev* ignores missing values (*x*@_FillValue). The output dimensionality is the same as the first n-2 dimensions of the input. To determine the number of data points used to calculate the standard deviation use:

```
N = num(.not.ismissing(x))
```

Technically, this function calculates the population standard deviation.

Example

The following is a simplistic example that demonstrates how *dim_stddev* works. Try running this example and print out the values to get a better idea of how *dim_stddev* works.

```
begin
;
;       Create example array
;
        a = onedtond(ispan(1,150,1),(/3,5,10/))
;
;       Assign dimension names
;
        a!0 = "time"
        a!1 = "latitude"
        a!2 = "longitude"
;
;       Compute standard deviation of dimension "longitude" at all time and latitude
;
        pxy = dim_stddev(a)
        N = num(.not.ismissing(a))    ; compute number of values used in calculation
;
;       Compute standard deviation of dimension "time" at all points latitude and lo
;       The following uses Named Subscripting to reorder the input array
;       such that time is the n-1 dimension.
;
        pyz = dim_stddev( a(latitude | :, longitude | : , time | :) )

end
```

---

dim_sum

For each index of dimensions 0..n-2, this function computes the sum of the n-1 dimension

---

Synopsis

```
        function dim_sum(
```

```
        x : numeric
)
```

Arguments

x

An array of one or more numeric values of any dimensionality

Description

The *dim_sum* function computes the sum of all elements of the n-1 dimension for each index of the dimensions 0..n-2. *dim_sum* ignores missing values. The output dimensionality is the same as the first n-2 dimensions of the input.

Example

The following is simplistic example that demonstrates how *dim_sum* works. Try running this example and print out the values to get a better idea of how *dim_sum* works.

```
begin
;
;       Create example array
;
        a = onedtond(ispan(1,150,1),(/3,5,10/))
;
;       Assign dimension names
;
        a!0 = "x"
        a!1 = "y"
        a!2 = "z"
;
;       Compute sum of dimension z at all points x and y
;
        pxy = dim_sum(a)
;
;       Compute sum of dimension x at all points y and z
;       The following uses Named Subscripting to reorder the input array
;       such that x is the n-1 dimension.
;
        pyz = dim_sum( a(y | :, z | : , x | :) )

end
```

dim_variance

For each index of dimensions 0..n-2, this function computes the variance of the n-1 dimension.

Synopsis

```
function dim_variance(
        x : numeric
)
```

Arguments

x

An array of one or more numeric values of any dimensionality

Description

The *dim_variance* function computes the variance of all elements of the n-1 dimension for each index of the dimensions 0..n-2. *dim_variance* ignores missing values (*x*@_FillValue). The output dimensionality is the same as the first n-2 dimensions of the input. To determine the number of data points used to calculate the standard deviation use:

```
N = num(.not.ismissing(x))
```

Example

The following is a simplistic example that demonstrates how *dim_variance* works. Try running this example and print out the values to get a better idea of how *dim_variance* works.

```
 begin ; ; Create example array ;
        a = onedtond(ispan(1,150,1),(/3,5,10/))
;
;       Assign dimension names
;
        a!0 = "time"
        a!1 = "latitude"
        a!2 = "longitude"
;
;       Compute variance of dimension "longitude" at all time and latitude points
;
        pxy = dim_variance(a)
        N = num(.not.ismissing(a))   ; compute number of values used in calculation
```

```
;
;       Compute variance of dimension "time" at all points latitude and longitude po
;       The following uses Named Subscripting to reorder the input array
;       such that time is the n-1 dimension.
;
        pyz = dim_variance( a(latitude | :, longitude | : , time | :) )

end
```

---

dimsizes

This function returns the dimension sizes of a data value.

Synopsis

```
        function dimsizes (
                data
        )
```

Arguments

data
: Any type of data array of any number of dimensions.

Description

The **dimsizes** function returns an integer array of dimension sizes for the parameter *data*. Each index in the array corresponds to the dimension number of the same value.

---

dsgrid2s, dsgrid2d

Functions for gridding 2-d data. These routines are part of the Dsgrid package which implements a simple inverse distance weighted interpolation algorithm.

Synopsis

```
function dsgrid2s(
    x[*]  : float,
    y[*]  : float,
    z[*]  : float,
    xo[*] : float,
    yo[*] : float
)

function dsgrid2d(
    x[*]  : double,
    y[*]  : double,
    z[*]  : double,
    xo[*] : double,
    yo[*] : double
)
```

Arguments

x
: 1D array of any length (*npts*) containing the X coordinates of the input data points

y
: 1D array of length *npts* containing the Y coordinates of the input data points

z
: 1D array of length *npts* containing the functional values of the input data points (*z(i)* is the value of the input function at coordinate (*x(i)*, *y(i)*) for i=1,*npts*).

xo
: 1D array of any length (*numxout*) containing the X coordinates of the output data grid. The values in *xo* must be increasing, but need not be equally spaced.

yo
: 1D array of any length (*numyout*) containing the Y coordinates of the output data grid. The values in *yo* must be increasing, but need not be equally spaced.

Description

The `dsgrid2s` and `dsgrid2d` functions do interpolation from 2D random data to a 2D float array dimensioned *numxout* x *numyout*.

Both `dsgrid2s` and `dsgrid2d` are called after all of the desired values for control parameters have been set using the procedure dssetp. `dsgrid2s` is the single precision version and `dsgrid2d` is the double precision version.

Example

```
begin
```

```
  NUM = 6
  NX = 61
  NY = 61

  xi = (/0.00, 1.00, 0.00, 1.00, 0.40, 0.75/)
  yi = (/0.00, 0.00, 1.00, 1.00, 0.20, 0.65/)
  zi = (/0.00, 0.00, 0.00, 0.00, 1.25, 0.80/)
  xeye =  3.3
  yeye = -3.3
  zeye =  3.3

  xo = new((/NX/),float)
  yo = new((/NY/),float)

;
; Create the output grid.
;
  xinc = 1./(NX-1)
  yinc = 1./(NY-1)
  ii = fspan(0.,60.,NX)
  xo = xinc * ii
  yo = yinc * ii
;
; Exponent equals 0.5
;
  dssetp("exp", 0.5)
  output = dsgrid2s(xi, yi, zi, xo, yo)
end
```

---

dsgrid3s, dsgrid3d

Functions for gridding 3-d data. These routines are part of the Dsgrid package which implements a simple inverse distance weighted interpolation algorithm.

Synopsis

```
    function dsgrid3s(
        x[*]  : float,
        y[*]  : float,
        z[*]  : float,
        u[*]  : float,
        xo[*] : float,
        yo[*] : float,
        zo[*] : float
    )

    function dsgrid3d(
        x[*]  : double,
        y[*]  : double,
        z[*]  : double,
        u[*]  : double,
        xo[*] : double,
```

```
        yo[*] : double,
        zo[*] : double
    )
```

Arguments

x
: 1D array of any length (*npts*) containing the X coordinates of the input data points

y
: 1D array of length *npts* containing the Y coordinates of the input data points

z
: 1D array of length *npts* containing the Z coordinates of the input data points

u
: 1D array of length *npts* containing the functional values of the input data points (*u(i)* is the value of the input function at coordinate (*x(i)*, *y(i)*, *z(i)*) for i=1,*npts*).

xo
: 1D array of any length (*numxout*) containing the X coordinates of the output data grid. The values in *xo* must be increasing, but need not be equally spaced.

yo
: 1D array of any length (*numyout*) containing the Y coordinates of the output data grid. The values in *yo* must be increasing, but need not be equally spaced.

zo
: 1D array of any length (*numzout*) containing the Z coordinates of the output data grid. The values in *zo* must be increasing, but need not be equally spaced.

Description

The `dsgrid3s` and `dsgrid3d` functions do interpolation from 3D random data to a 3D gridded array dimensioned *numxout* x *numyout* x *numzout*.

`dsgrid3s` returns an array of float values and `dsgrid3d` returns an array of double values. Both `dsgrid3s` and `dsgrid3d` are called after all of the desired values for control parameters have been set using the procedure dssetp.

`dsgrid3s` is the single precision version and `dsgrid3d` is the double precision version.

Example

```
begin

  NUM = 1000
```

```
  NX = 21
  NY = 21
  NZ = 21
  RAND_MAX = 32767.0

  xi = new((/NUM/),float)
  yi = new((/NUM/),float)
  zi = new((/NUM/),float)
  u = new((/NUM/),float)

  xo = new((/NX/),float)
  yo = new((/NY/),float)
  zo = new((/NZ/),float)

  xmin = -2.0
  ymin = -2.0
  zmin = -2.0
  xmax =  2.0
  ymax =  2.0
  zmax =  2.0

;
; Create random data in three space and define a function.
;
  rand1 = new((/NUM/),float)
  rand2 = new((/NUM/),float)
  rand3 = new((/NUM/),float)
  srand(1)
  do i=0,NUM-1
    rand1(i) = rand
    rand2(i) = rand
    rand3(i) = rand
  end do
  xi = xmin+(xmax-xmin)*(rand1 / RAND_MAX)
  yi = ymin+(ymax-ymin)*(rand2 / RAND_MAX)
  zi = zmin+(zmax-zmin)*(rand3 / RAND_MAX)

  u = xi*xi + yi*yi + zi*zi
;
; Create the output grid.
;
  ii = fspan(0,20.,21)
  xo = xmin + (ii/(NX-1)) * (xmax-xmin)
  yo = ymin + (ii/(NY-1)) * (ymax-ymin)
  zo = zmin + (ii/(NZ-1)) * (zmax-zmin)

;
; Interpolate.
;
  output = dsgrid3s(xi, yi, zi, u, xo, yo, zo)

end
```

---

dssetp, dsgetp

Set and retrieve parameters for Dsgrid routines. The ncl language affords a convenient implementation of the parameter setting and retrieval functions.

Synopsis

```
    procedure dssetp(
        pnam[1] : string,
        pval[1]
    )

    function dsgetp(
        pnam[1] : string
    )
```

Arguments

pnam
name of the parameter you want to set or retrieve

pval
value of the parameter you want to set; this value must be of the type appropriate to the parameter being set.

Description

The procedure `dssetp` is used to set values for natgrid parameters; the function `dsgetp` is used to retrieve the current value of the named parameter. `dsgetp` returns a value that is of the type appropriate to the parameter type.

Example

```
begin

  NUM = 6
  NX = 61
  NY = 61

  xi = (/0.00, 1.00, 0.00, 1.00, 0.40, 0.75/)
  yi = (/0.00, 0.00, 1.00, 1.00, 0.20, 0.65/)
  zi = (/0.00, 0.00, 0.00, 0.00, 1.25, 0.80/)
  xeye =  3.3
  yeye = -3.3
  zeye =  3.3

  xo = new((/NX/),float)
```

```
  yo = new((/NY/),float)

;
; Create the output grid.
;
  xinc = 1./(NX-1)
  yinc = 1./(NY-1)
  ii = fspan(0.,60.,NX)
  xo = xinc * ii
  yo = yinc * ii
;
; Exponent equals 0.5
;
  dssetp("exp", 0.5)
  output = dsgrid2s(xi, yi, zi, xo, yo)
end
```

---

dspnt2s, dspnt2d

Routines that interpolate 2-d data at specified points. These routines are part of the Dsgrid package which implements a simple inverse distance weighted interpolation algorithm.

Synopsis

```
    procedure dspnt2s(
        x[*]  : float,
        y[*]  : float,
        z[*]  : float,
        xo[*] : float,
        yo[*] : float
    )

    procedure dspnt2d(
        x[*]  : float,
        y[*]  : float,
        z[*]  : float,
        xo[*] : float,
        yo[*] : float
    )
```

Arguments

x

1D array of any length (*npts*) containing the X coordinates of the points where interpolation is desired

y

1D array of length *npts* containing the Y coordinates of the points where interpolation is desired

z

1D array of length *npts* containing the functional values of the input data points. *z(i)* is the value of the input function at coordinate (*x(i)*, *y(i)*) for i=1,*npts*.

xo

1D array of length *m* containing the X coordinates of the output data grid. The values in *xo* may be in any order and *m* can be equal to one.

yo

1D array of length *m* containing the Y coordinates of the output data grid. The values in *yo* may be in any order and *m* can be equal to one.

Description

Both the `dspnt2s` and `dspnt2d` functions interpolate 2D data at a list of specified points and return the results as a 1D float array dimensioned *m*.

`dspnt2s` is the single precision version and `dspnt2d` is the double precision version.

Example

```
begin

  NUM = 171
  NX = 21
  NY = 21

  RAND_MAX = 32767.0

  xi = new((/NUM/),float)
  yi = new((/NUM/),float)
  zi = new((/NUM/),float)

  xo = new((/NX/),float)
  yo = new((/NY/),float)
  output2 = new((/NX,NY/),float)

  xminin = -0.2
  yminin = -0.2
  xmaxin = 1.2
  ymaxin = 1.2
  xminot =  0.0
  yminot =  0.0
  xmaxot = 1.0
  ymaxot = 1.0

  xeye = 1.3
  yeye = -1.8
  zeye = 3.6
;
```

```
; Create random data in three space and define a function.
;
  rand1 = new((/NUM/),float)
  rand2 = new((/NUM/),float)
  srand(1)
  do i=0,NUM-1
    rand1(i) = rand
    rand2(i) = rand
  end do
  xi = xminin+(xmaxin-xminin)*(rand1 / RAND_MAX)
  yi = yminin+(ymaxin-yminin)*(rand2 / RAND_MAX)
  zi = (xi-0.25)*(xi-0.25) + (yi-0.50)*(yi-0.50)

;
; Create the output grid.
;
  ii = fspan(0.,NX-1,NX)
  xo = xminot + ( ii / (NX-1)) * (xmaxot-xminot)
  yo = yminot + ( ii / (NY-1)) * (ymaxot-yminot)

;
; Interpolate using both dsgrid2s and dspnt2s.
;
  output1 = dsgrid2s(xi, yi, zi, xo, yo)

  do i=0,NX-1
    do j=0,NY-1
      dspnt2s(xi, yi, zi, xo(i), yo(j), output2(i,j))
    end do
  end do
end
```

---

dspnt3s, dspnt3d

Routines that interpolate 3-d data at specified points. These routines are part of the Dsgrid package which implements a simple inverse distance weighted interpolation algorithm.

Synopsis

```
function dspnt3s(
    x[*]  : float,
    y[*]  : float,
    z[*]  : float,
    u[*]  : float,
    xo[*] : float,
    yo[*] : float,
    zo[*] : float
)

function dspnt3d(
    x[*] :  double,
    y[*] :  double,
```

```
        z[*] :  double,
        u[*] :  double,
        xo[*] : double,
        yo[*] : double,
        zo[*] : double
    )
```

Arguments

x
: 1D array of any length (*npts*) containing the X coordinates of the points where interpolation is desired

y
: 1D array of length *npts* containing the Y coordinates of the points where interpolation is desired

z
: 1D array of length *npts* containing the Z coordinates of the points where interpolation is desired

u
: 1D array of length *npts*containing the functional values of the input data points. *u(i)* is the value of the input function at coordinate (*x(i)*, *y(i)*, *z(i)*) for i=1,*npts*.

xo
: 1D array of length *m* containing the X coordinates of the output data grid. The values in *xo* may be in any order and *m* can be equal to one.

yo
: 1D array of length *m* containing the Y coordinates of the output data grid. The values in *yo* may be in any order and *m* can be equal to one.

zo
: 1D array of length *m* containing the Z coordinates of the output data grid. The values in *zo* may be in any order and *m* can be equal to one.

Description

Both the `dspnt3s` and `dspnt3d` functions interpolate 3D data at a list of specified points and return the results as a 1D float array dimensioned *m*.

`dspnt3s` is the single precision version and `dspnt3d` is the double precision version.

Example

```
begin

  NUM = 1000
  NX = 21
  NY = 21
```

```
  NZ = 21
  RAND_MAX = 32767.0

  xi = new((/NUM/),float)
  yi = new((/NUM/),float)
  zi = new((/NUM/),float)
  u = new((/NUM/),float)

  xo = new((/NX/),float)
  yo = new((/NY/),float)
  zo = new((/NZ/),float)
  output = new((/NX,NY,NZ/),float)

  xmin = -2.0
  ymin = -2.0
  zmin = -2.0
  xmax =  2.0
  ymax =  2.0
  zmax =  2.0

;
; Create random data in three space and define a function.
;
  rand1 = new((/NUM/),float)
  rand2 = new((/NUM/),float)
  rand3 = new((/NUM/),float)
  srand(1)
  do i=0,NUM-1
    rand1(i) = rand
    rand2(i) = rand
    rand3(i) = rand
  end do
  xi = xmin+(xmax-xmin)*(rand1 / RAND_MAX)
  yi = ymin+(ymax-ymin)*(rand2 / RAND_MAX)
  zi = zmin+(zmax-zmin)*(rand3 / RAND_MAX)

  u = xi*xi + yi*yi + zi*zi
;
; Create the output grid.
;
  ii = fspan(0,20.,21)
  xo = xmin + (ii/(NX-1)) * (xmax-xmin)
  yo = ymin + (ii/(NY-1)) * (ymax-ymin)
  zo = zmin + (ii/(NZ-1)) * (zmax-zmin)

  do i=0,NX-1
    do j=0,NY-1
      do k=0,NZ-1
        dspnt3s(xi, yi, zi, u, xo(i), yo(j), zo(k),output(i,j,k))
      end do
    end do
  end do
end
```

---

dtrend

Removes the least squares linear trend and estimates of the linear trend at all grid points.

Note: This function is only available in version 4.1.1 of NCL. If your site is licensed for version 4.1, then you should receive version 4.1.1 automatically.

Synopsis

```
function dtrend(
    x            : float,
    return_slope : logical
)
```

Arguments

x

Input float array, series to be detrended. The dimension to be detrended (usually "time") should be the fastest varying dimension (i.e. the rightmost dimension). The mean of *x* is also removed.

return_slope

Set this to True or False, depending on whether you want the slope(s) returned as an attribute of the value returned by the function. The attribute will be called *slope*. If *return_slope* is True, then the slope(s) will be a returned as a float one-dimensional array or a scalar, with the same dimensions as *x* only with the last dimension omitted. If *x* is a one-dimensional array, then *slope* will be a scalar.

Description

Using least squares, `dtrend` removes the linear trend at all grid points and returns a float array of the detrended series. It will also return the linear trend per unit time interval (slope) as an attribute called *slope* if the second argument is set to True. The input data cannot contain any missing values.

Example 1

Let `x`(lat,lon,time) and sizes nlat=64, mlon=128 and ktime = 120 (months). Then

```
xnew = dtrend (x,False)
```

will return an array `xnew`(lat,lon,time). `xnew` will contain the detrended data. The mean is also removed.

Example 1a

```
xnew = dtrend (x,True)
```

will return an array `xnew`(lat,lon,time). `xnew` will contain the detrended data and the slopes as an attribute called `slope` containing nlat*mlon elements. Thus, if `x` contained temperatures [degrees K] and time was in months then the units of `xnew@slope` would be [K/month]. Since attributes can not be returned as two-dimensional arrays, the user should use the NCL function "onedtond" to create a two-dimensional array if it is desired for something like plotting purposes:

```
slope2D = onedtond(xnew@slope, (/nlat,mlon/) )
delete (xnew@slope)
slope2D = slope2D*120        ; would give [K/decade]
```

Example 2

Let `x`(time,lat,lon) with named dimensions "time", "lat", "lon", then one must use NCL's named dimensions to reorder:

```
xnew = dtrend (x(lat|:,lon|:,time|:), False)
```

`xnew` will have dimension order (lat,lon,time).

---

dv2uvf, dv2uvg

Given a divergent array, compute the divergent (irrotational) wind components via Spherepack.

Synopsis

```
procedure dv2uvf(
    dv : float,
    ud : float,
    vd : float
)

procedure dv2uvg(
    dv : float,
    ud : float,
    vd : float
)
```

Arguments

dv

divergence array (input, two or more dimensions, last two dimensions must be *nlat* x *nlon* and input values must be in ascending latitude order)

ud, vd

divergent wind components (output, same dimensions as *dv*, values will be in ascending latitude order)

Description

dv2uvf and *dv2uvg* both compute the divergent wind components given a divergent array *dv* and return the results in the arrays *ud* and *vd*. *dv2uvf* operates on an equal (fixed) grid, and *dv2uvg* operates on a Gaussian grid.

Note: For the arrays whose last two dimensions are *nlat* x *nlon*, the rest of the dimensions (if any) are collectively referred to as *nt*. If the input/output arrays are just two dimensions, then *nt* can either be considered equal to 1 or nothing at all.

Arrays which have dimensions *nt* x *nlat* x *nlon* should **not** include the cyclic (wraparound) points when invoking the procedures and functions which use spherical harmonics (Spherepack).

For example, if an array *x* has dimensions *nlat* = 64 and *nlon* = 129, where the "129" represents the cyclic points, then the user should pass the data to the procedure/function via:

```
    z = sample ( x([...],:,0:nlon-2) )  ; does not include cyclic points
```

Example

```
begin
  nlat  =  64                                       ; dimensions
  mlon  = 128
  mlon1 = mlon+1
  fbfile = "uv300.hs"
                                                ; Generic Workstation setup
  nrec  = fbinnumrec(fbfile)        ; total number of records in the file
  ntim  = nrec/2                        ; number of time steps in dataset

  uvmsg = 1e+36

  work  = new ( (/nlat,mlon1/), float, uvmsg )
  u     = new ( (/nlat,mlon /), float, uvmsg )  ; source u
  v     = new ( (/nlat,mlon /), float, uvmsg )  ; source v
  dvx   = new ( (/nlat,mlon /), float, uvmsg )  ; divergence (same as dv)
  vrx   = new ( (/nlat,mlon /), float, uvmsg )  ; vorticity  (same as vr)
  sf    = new ( (/nlat,mlon /), float, uvmsg )  ; stream function
  vp    = new ( (/nlat,mlon /), float, uvmsg )  ; velocity potential
  uy    = new ( (/nlat,mlon /), float, uvmsg )  ; latitudinal derivative (u)
  vy    = new ( (/nlat,mlon /), float, uvmsg )  ; latitudinal derivative (v)
  uu    = new ( (/nlat,mlon /), float, uvmsg )  ; reconstructed u
  vv    = new ( (/nlat,mlon /), float, uvmsg )  ; reconstructed v
  ux    = new ( (/nlat,mlon /), float, uvmsg )  ; reconstructed u
  vx    = new ( (/nlat,mlon /), float, uvmsg )  ; reconstructed v
```

```
  do i = 0,nrec-1,2
   month = 1                         ; january
   if (i .ge. 2) then
       month = 7                     ; july
   end if

   work = fbinrecread(fbfile,i  ,(/nlat,mlon1/),"float")
   u    = work(:,0:mlon-1)
   work = fbinrecread(fbfile,i+1,(/nlat,mlon1/),"float")
   v    = work(:,0:mlon-1)

   dv = uv2dvG   (u,v)               ; u,v ==> divergence
   vr = uv2vrG   (u,v)               ; u,v ==> vorticity (rel)
   uv2vrdvg (u,v, vrx,dvx)           ; u,v ==> div and vort
   uv2sfvpg (u,v, sf,vp)             ; u,v ==> stream function + velocity pot
   ud    = new ( (/nlat,mlon /), float, uvmsg )
   vd    = new ( (/nlat,mlon /), float, uvmsg )
   ur    = new ( (/nlat,mlon /), float, uvmsg )

   dv2uvg   (dv,  ud,vd)             ; dv  ==> divergent  wind components
   vr2uvg   (vr,  ur,vr)             ; vr  ==> rotational wind components
   vrdv2uvg (vr,dv,   uu,vv)         ; vr,dv > reconstruct original wind
   vrdv2uvg (vrx,dvx, ux,vx)         ; vr,dv > reconstruct original wind
  end do
end
```

Error messages

If *jer* or *ker* is equal to:

1 : error in the specification of *nlat*
2 : error in the specification of *nlon*
4 : error in the specification of *nt* (*jer* only)

---

eofcor, eofcov, eofcor_ts, eofcov_ts

Empirical orthogonal functions

Synopsis

```
function eofcor(
    data[*][*]  : float,
    neval[1] : integer
)

function eofcov(
    data[*][*]  : float,
```

```
        neval[1] : integer
    )

    function eofcor_ts(
        data[*][*] : float,
        evec[*][*] : float
    )

    function eofcov_ts(
        data[*][*] : float,
        evec[*][*] : float
    )
```

Arguments

data
: 2D array dimensioned *num_stations* x *time*

neval
: Scalar integer that specifies the number of eigenvalues and eigen vectors to be computed

evec
: 2D array returned from `eofcor` or `eofcov` dimensioned *neval* x *num_stations*

Description

`eofcov` computes empirical orthogonal functions (eofs) via the covariance matrix (*data*) and returns a 2D float array dimensioned *neval* x *num_stations*. `eofcor` computes empirical orthogonal functions (eofs) via the correlation matrix (*data*) and returns a 2D float array dimensioned *neval* x *num_stations*.

Both of these functions return the following attributes:

trace (scalar, float)
: trace of the variance-covariance matrix

eval (1D float array of length *neval*)
: eigenvalues in descending order

pcvar (1D float array of length *neval*)
: percent variance associated with each eigenvalue

Both `eofcov` and `eofcor` will ignore missing values if *data*@_FillValue is set. It is assumed that all stations (grid points) have at least some data present. This is the user's responsibility. if *data*@_FillValue is set, then the return array will have the same attribute set to this value.

`eofcor_ts` computes the time series of eof amplitudes for each eigenvalue returned by `eofcor` and returns a 2D float array dimensioned *neval* x *time*. `eofcov_ts` computes the time series of eof amplitudes for each eigenvalue returned by `eofcov` and returns a 2D float array dimensioned *neval* x *time*.

Example

```
begin
;
; Parameters
;
  ncol   = 7                        ; # columns (stations or grid points)
  ntime  = 25                       ; # time steps

;
; Read the ASCII file.
;                                   ; open "data " as 2-dim array
  data = asciiread ("data",(/ntime,ncol/), "float")
;
; Eof routines want "time" to be fastest varying dimension.
;
  data!0 = "time"
  data!1 = "ncol"

  neval    = 3
  evecv    = eofcov    (data(ncol | :,time | :),neval)
  evecv_ts = eofcov_ts (data(ncol | :,time | :),evecv)

  print ("evecv@trace= " + evecv@trace)
  print ("evecv@pcvar= " + evecv@pcvar)
;
; here is another way to do the calls
;
  data2    = data(ncol | :,time | :)
  evecr    = eofcor    (data2,neval)
  evecr_ts = eofcor_ts (data2,evecr)

  print ("evecr@trace= " + evecr@trace)
  print ("evecr@pcvar= " + evecr@pcvar)
end
```

---

esacr, esacv, esccr, esccv

Functions for computing autocovariances, autocorrelations, crosscovariances, and crosscorrelations for an input vector with missing data. The normalizing factor in the denominator (xvar) is kept constant.

Synopsis

```
    function esacr(
        x        : float,
```

```
        mxlag[1] : integer
    )

    function esacv(
        x        : float,
        mxlag[1] : integer
    )

    function esccr(
        x        : float,
        y        : float,
        mxlag[1] : integer
    )

    function esccv(
        x        : float,
        y        : float,
        mxlag[1] : integer
    )
```

Arguments

x

Input array of any dimensionality (last dimension is *npts*). Missing values should be indicated by *x*@_FillValue. If *x*@_FillValue is not set, then the NCL default (-999) will be assumed.

y

Input array of same dimensionality as *x*.Missing values should be indicated by *y*@_FillValue. If *y*@_FillValue is not set, then the NCL default (-999) will be assumed.

mxlag

single scalar maximum lag to be estimated (0 <= *mxlag* <= *npts*)

Description

Given vector/array *x*, `esacv` and `esacr` compute sample autocovariances and autocorrelations respectively, returning a multi-dimensional float array of the same dimensions as *x* except with the last dimension *npts* replaced by *mxlag*+1.

Given vectors/arrays (*x, y*), `esccv` and `esccr` compute sample crosscovariances and crosscorrelations respectively, returning a multi-dimensional array of the same dimensions as *x* and *y* except with the last dimension *npts* replaced by *mxlag*+1.

In the crosscovariance and crosscorrelation computations the y data "lead" the x data.

It is recommended that *mxlag* not be greater than *npts*/3, for statistical reasons.

Currently, to calculate positive and negative lags one must invoke this function twice. (See example 5 below.)

Example 1

Consider a one-dimensional array, x. Then,

```
acr = esacr(x,10)
acv = esacv(x,10)
```

will calculate the autocorrelations and variances at 11 lags (0->10). acr and acv will be one-dimensional variables of length 11.

Example 2

Consider two three-dimensional arrays, y(*lat*,*lon*,*time*) and z(*time*,*lat*,*lon*) with named dimensions "time", "lat" and "lon". Then

```
mxlag = 10
acr_y = esacr(y,mxlag)
acv_z = esacv(z(lat|:,lon|:,time|:),mxlag) ; dimension reordering
```

will calculate the autocorrelations and variances at (*mxlag*+1) lags (0->*mxlag*). acr_y and acv_z will have dimension sizes (*nlat*,*mlon*,*mxlag*+1). (NCL's dimension reordering was used in the latter case.)

Example 3

Consider two one-dimensional series x and y of the same size.

```
ccr = esacr(x,y,10)
ccv = esacv(x,y,10)
```

will calculate the cross-correlations and variances at 11 lags (0->10). ccr and ccv will be one-dimensional variables of length 11. The y series leads the x series in the computations.

Example 4

Consider two three-dimensional arrays, y(*lat*,*lon*,*time*) and z(*lat*,*lon*,*time*). Then

```
mxlag  = 10
ccr_yz = esccr(y,z,mxlag)
ccv_zy = esacv(z,y,mxlag)
```

will calculate the autocorrelations and variances at (*mxlag*+1) lags (0->*mxlag*). ccr_yz and ccv_zy will have dimension sizes (*nlat*,*mlon*,*mxlag*+1). Of course, users might have to use dimension reordering if the dimensions are not in the desired order.

Example 5

Consider two one-dimensional series x and y of the same size and the user wishes to calculate cross-correlations at positive and negative lags. Then the user will have to use the (admittedly cumbersome) procedure:

```
mxlag    = 9
y_Lead_x = esccr(x,y,mxlag)
x_Lead_y = esccr(y,x,mxlag)    ; switch the order of the series

ccr = new ( 2*mxlag+1, float)
ccr(0:mxlag-1) = x_Lead_y(1:mxlag:-1)  ; "negative lag", -1 reverses order
ccr(mxlag:)    = y_Lead_x(0:mxlag)     ; "positive lag"
```

Remember that NCL is C based and does not allow negative subscripts like Fortran does (in Fortran, ccr(-*mxlag*:*mxlag*)). Thus in the above example `ccr`(0:*mxlag*-1) corresponds to lags -9,-8,...,-2,-1 and `ccr`(*mxlag*:2**mxlag*) corresponds to lags 0,1,2,...,8,9.

In a future release, a single interface which does the above process will be provided.

---

exit

Used to prematurely force a script to exit. This is different from the quit statement.

Synopsis

```
procedure exit()
```

Description

Calls unix exit command with the value of 0.

---

fabs

This function is used to return the absolute values for floating-point numbers.

Synopsis

```
function fabs(
        value:numeric
)
```

Arguments

value
: One or more values of any dimension.

The absolute value of each element of the *value* parameter is returned. The return array is dimensioned the same as the *value* parameter. Missing values are ignored. Return value is of type **double** if input type is **double**.

---

fbindirread

This function reads records written by Fortran code that opened the file with 'access=direct'.

All records in the file must be the same dimensionality and type.

Synopsis

```
        function fbindirread(
                path [1] : string,
                rec_num [1] : integer,
                dims [*] : integer,
                type [1] : string
        )
```

Arguments

path
: pathname to binary file

rec_num
: record number to read from the file beginning at 0

dims
: a singly dimensioned array of integer values that describe how to shape the data, or -1 if the size of the record is unknown.

type
: string name of the data type of the record.

Description

fbindirread reads the *rec_num* record of the file *path* and shapes it according to the dimension sizes specified in the *dims* parameter. The data type of the record is specified by the parameter *type*. If the size and dimensionality of the record are unknown, the value -1 can be used for parameter *dims*. In this case *fbindirread* will read in the entire record as a singly dimensioned array.

All records in the file must have the same dimensions and must be the same type.

Example

```
path = "/dummy/file"
nrec = 5
dims = (/10,30/)
x = fbindirread(path,nrec,dims,"float")
```

This will read the 6th record from "/dummy/file" and return a two-dimensional [10 by 30] variable of type float to the variable "x".

---

fbindirwrite

This function writes records to a file in a manner similar to that written by Fortran code that opens a file with "access=direct".

Synopsis

```
        procedure fbindirwrite(
                path [1] : string,
                value
        )
```

Arguments

path
: pathname to binary file

value
: an array of any dimensionality and type

Description

fbindirwrite writes records to a file in a manner similar to that written by Fortran code that opens a file with 'access=direct'. If the file exists value is appended to the file.

Example

```
ntimes = 100
z    = new ( (/ntimes,64,128/), float)
          .
path = "/dummy/file"
do n=0,ntimes-1
   fbindirwrite(path,z(nt,:,:) )
end do
```

This will write `ntimes` records, each of length 64x128 float words.

---

fbinnumrec

Returns the number of unformatted sequential Fortran records in a binary file.

Synopsis

```
        function fbinnumrec(
                path [1]: string
        )
```

Arguments

path

pathname of file

Description

This function returns the number of records in a Fortran binary file.

---

fbinread

This function is used to read a binary file that has been written using an UNFORMATTED FORTRAN write and contains only one record. The **fbinrecread** function should be used for files with multiple records and **fbindirread** should be used for files written with Fortran direct access files.

Synopsis

```
function fbinread(
        filepath[1]:string,
        dimensions[*]:integer,
        datatype[1]:string
)
```

Arguments

filepath
Path needed to locate binary file.
dimensions
An array specifying the dimensions of the data to be read or the value -1.
datatype
A string representing the type of the data being read.

Description

The **fbinread** function is used to read binary data. To guarantee this function will work, the data must have been written on a machine with the same architecture and Fortran compiler as the machine on which they are being read. If the specified dimensions do not define a size equal to the total file data size, an error message is generated. Note that there is no way for NCL to determine if the type of file and the *datatype* parameter represent the same type. If the value of -1 is used for *dimensions* then all of the values of the Fortran record are read and the output is a single dimension variable with a dimension size equal to the number of elements of the record.

The purpose of this function is to "skip" over control words often written into Fortran records. This function, technically, is limited to only reading one Fortran record. However, if the size of the control words are equal to the data type being read and the file contains only a single data type, there are techniques for reading multiple record Fortran binary files. (See Binary data input in the NCL user guide tutorial)

fbinrecread

This function reads records from binary files written using UNFORMATTED SEQUENTIAL access.

Synopsis

```
function fbinrecread(
        path [1] : string,
        rec_num [1] : integer,
        dims [*] : integer,
        type [1] : string
)
```

Arguments

path
: pathname to binary file

rec_num
: record number to read from the file beginning at 0

dims
: a singly dimensioned array of integer values that describe how to shape the data, or -1 if the size of the record is unknown.

type
: string name of the data type of the record.

Description

fbinrecread reads the *rec_num* record of the file *path* and shapes it according to the dimension sizes in parameter. The data type of the record is specified by the parameter *type*. If the size and dimensionality of the record are unknown the value -1 can be used for parameter *dims*. In this case *fbinrecread* will read in the entire record as a singly dimensioned array.

Example

Example pending

---

fbinrecwrite

Writes a single sequential access IEEE Fortran record to the file path.

Synopsis

```
procedure fbinrecwrite(
```

```
            path [1] : string,
            rec_num [1] : integer,
            value
    )
```

Arguments

path
    pathname to binary file
rec_num
    record number to read from the file beginning at 0, -1 for append
value
    an array of any dimensionality and type

Description

fbinrecwrite writes a single sequential access IEEE Fortran record to the file *path*. If *rec_num* is equal to -1 then *value* is appended to the end of the file. If *rec_num* is not equal to -1 then it is the record number to write to. If the size of *value* is different than the record size in the file an error message is generated and the file *path* is not modified. If *rec_num* is greater than the number of records in the file, *value* is then appended to the file and a warning message is generated.

fbinwrite

This function is used write a binary file containing a single record. The file can be read usings access = sequential from FORTRAN.

Synopsis

```
    procedure fbinwrite(
            filepath[1]:string,
            value:numeric
    )
```

Arguments

filepath
    Path needed to locate binary file.
value

A numeric value of any dimensionality.

Description

The **fbinwrite** function is used to create a binary data file from an NCL numeric variable. The variable is written to a file as a single sequential access unformatted record.

---

ezfftf, ezfftb

Functions for computing Fourier coefficients of a real periodic sequence (Fourier analysis) and vice versa.

Synopsis

```
function ezfftf(
    x[*] : float
)

function ezfftb(
    cf[2][*]   : float,
    xbar[1] : float
)
```

Arguments

x
periodic sequence to be transformed (1D array of length *npts* where *npts* need not be a power of 2)

cf
fourier coefficients (2D array dimensioned 2 x (*npts*/2)+1)

xbar
constant fourier coefficient

Description

Given periodic sequence *x*, `ezfftf` computes its Fourier coefficients and returns it as a 2D float array dimensioned 2 x (*npts*/2)+1. It also returns the constant Fourier coefficient *xbar* as an attribute of *cf*.

Given Fourier coefficients *cf* and constant Fourier coefficient *xbar*, `ezfftb` computes the periodic sequences and returns a 1D float array of length *npts*.

---

fileattdef

Given a reference to a file, this procedure is used to define global attributes.

Synopsis

```
        procedure fileattdef (
                thefile[1] : file,
                variable
        )
```

Arguments

thefile
: The reference to the file that you want to write the attributes to.

variable
: A variable of any type whose attributes will be copied to *thefile* as global attributes

---

Description

Given a reference to a file, *fileattdef* is used to define attributes applicable to the file as a whole. These are sometimes known as global file attributes. Using this procedure is much more efficient than writing a file's attributes one at a time.

This procedure is part of a set of procedures called filedimdef, filevarattdef, and filevardef.

Example

```
  ncf = addfile("myfile.nc","c")
;
; Define global attributes.
;
; globalAtt can be of any type. Here logical is used
;
  globalAtt=True
  nl = integertochar(10)  ; newline character
```

```
globalAtt@history     = nl+\
    systemfunc("date") + ": ncl < TQ_NCEP2nc.ncl"
globalAtt@sigma_level = nl+\
    "Pressure at a grid point (lon(i),lat(j),lev(k)) is computed    "+nl+\
    "using the formula:                                              "+nl+\
    "           p(i,j,k) = B(k)*PS(i,j)                              "+nl+\
    "where B and PS are contained in the variables whose names      "+nl+\
    "are given respectively by the attributes B_var and PS_var      "+nl+\
    "of the vertical coordinate variable.                             "
globalAtt@center      = ps@center
globalAtt@model       = ps@model
globalAtt@title       = "T, Q, and PS from NCEP/NCAR reanalysis data"
globalAtt@source      = "NCEP/NCAR reanalysis data"
globalAtt@Conventions = "NCAR-CSM"
fileattdef( ncf, globalAtt )
```

---

filedimdef

Given a reference to a file, this procedure is used to define a list of dimension names, dimension sizes, and a logical array indicating whether the dimensions are unlimited.

Synopsis

```
procedure filedimdef (
        thefile[1] : file,
        dimnames : string,
        dimsizes : integer,
        is_unlimited : logical
)
```

Arguments

thefile
: The reference to the file that you want to write the dimensions to.

dimnames
: An array of dimension names you want to write to *thefile*.

dimsizes
: An array of dimension sizes of the dimensions you want to write to *thefile*.

is_unlimited
: An array of logical values indicating whether the corresponding dimensions are unlimited in size, note netCDF only allows one unlimited dimension.

Description

This procedure is used to pre-define dimensions in a file. Usually the file is being created and must have been opened as either read/write or create. The fourth parameter allows the dimension to be defined as unlimited. NetCDF files only allow one dimension to be unlimited. Unlimited dimensions can grow while normal dimensions can not. A typical use of unlimited dimensions is to create a data from multiple input files. For unlimited dimensions the size value is ignored. Unlimited dimensions have a size of 0 until a variable with the unlimited dimension name is assigned to the file.

Using this procedure is much more efficient than writing a variable's dimensions one at a time.

This procedure is part of a set of procedures called fileattdef, filevarattdef, and filevardef.

Example

```
  ncf = addfile("myfile.nc","c")
;
; Define dimensions.
;
  dimNames = (/ "lon", "lat", "lev", "time" /)
  dimSizes = (/ nlon, nlat, nlev, -1 /)
  dimUnlim = (/ False, False, False, True /)
  filedimdef( ncf, dimNames, dimSizes, dimUnlim )
```

---

filevarattdef

Given a reference to a file, this procedure copies attributes from an input variable to one or more file variables.

Synopsis

```
        procedure filevarattdef (
                thefile[1] : file,
                varnames : string,
                variable
        )
```

Arguments

thefile

The reference to the file that you want to write the variable attributes to.

varnames

A list of file variable names to which you want to copy attributes.

variable

A variable, of any type, which contains one or more attributes to assign to the file variables listed in *varnames*.

Description

This procedure copies attributes from the input variable to the output file variables, named by *varnames*. The variables in *varnames* must already be defined either through the procedure filevardef or through normal assignment.

This procedure is part of a set of procedures called filedimdef, fileattdef, and filevardef.

Example

```
  ncf = addfile("myfile.nc","c")
;
; Define variable attributes.
;
  filedimdef(ncf , "lat", 73 , False)
  filevardef( ncf, "lat", "float", "lat" )
  latAtt=0
  latAtt@units     = "degrees_north"
  latAtt@long_name = "latitude"
  filevarattdef( ncf, "lat", latAtt )

  filevardef( ncf, "gw", "float", "lat" )
  gwAtt=0
  gwAtt@long_name  = "gauss weights"
  filevarattdef( ncf, "gw", gwAtt )
```

---

filevardef

Given a reference to a file, this procedure is used to define list of variable names, variable types, and variable dimension names.

Synopsis

```
        procedure filevardef (
```

```
            thefile[1] : file,
            varnames : string,
            types : string,
            dimnames : string
    )
```

Arguments

thefile
    The reference to the file that you want to define the variables in.
varnames
    An array of variable names you want to define.
types
    An array of variable types of the variables you want to define
dimnames
    The set of dimensions which apply to all names in varnames

---

Description

This procedure defines one or more variables in and output netCDF or HDF file. *varnames* can contain multiple names of variables and *types* must contain a type name for each variable name in *varnames*. All variables to be defined will be defined with the set of dimension names in *dimnames*. The dimensions must be in the order desired. Dimension 0 is the first entry in *dimnames*. The dimensions must also have been defined already either with the procedure filedimdef or by normal variable assignment.

This procedure is part of a set of procedures called fileattdef, filedimdef, and filevarattdef.

Example

```
ncf = addfile("myfile.nc","c")
filedimdef(ncf,(/"lat","lon"/),(/73,144/),(/False,False/)
varNames2D = (/ "PS" /)
varTypes2D = (/ "float" /)
varNames3D = (/ "T", "Q" /)
varTypes3D = (/ "float", "float" /)
filevardef( ncf, varNames2D, varTypes2D, (/  "lat", "lon" /) )
filevardef( ncf, varNames3D, varTypes3D, (/  "lev", "lat", "lon" /) )
```

---

filevardimsizes

Returns the dimension sizes of a file variable.

Synopsis

```
function filevardimsizes(
        thefile [1] : file,
        varname [1] : string
)
```

Arguments

thefile
: reference to a file

varname
: string name of variable whose dimension sizes are desired

Description

This function works just like dimsizes but for file variables. This function should be used rather that dimsizes when the variable is in a file. *dimsizes* is very inefficient to use on file variables because it reads the entire variable in. Therefore, *filevardimsizes* should be used exclusively when querying the dimensionality of variables in files.

floor

This function returns the largest integral value smaller than the input.

Synopsis

```
function floor(
        value:numeric
)
```

Arguments

value
: An array of one or more floating-point values of any dimension.

Description

For each element in the *value* parameter, the largest integral value smaller than the element is returned. The output dimensions are identical to the *value* parameter. Missing values in the input are ignored and propagated through to the output. The output type is **double** if the input is **double**.

---

fluxEddy

Calculates time averaged eddy flux quantities.

Note: This function is only available in version 4.1.1 of NCL. If your site is licensed for version 4.1, then you should receive version 4.1.1 automatically.

Synopsis

```
function fluxEddy(
    x  : float,
    y  : float
)
```

Arguments

x
: A float array with one or more dimensions. The rightmost dimension must be "time". Missing values should be indicated by *x*@_FillValue. If *x*@_FillValue is not set, then the NCL default (-999) will be assumed.

y
: A float array with one or more dimensions. It must have the same dimensions as *x*. Missing values should be indicated by *y*@_FillValue. If *y*@_FillValue is not set, then the NCL default (-999) will be assumed.

Description

`fluxEddy` returns a float array of the same dimensionality as its input arrays with the last dimension omitted. The resulting quantities are computed in one pass. The approach used is let x=X+x' and y=Y+y' where X and Y are time averages, then x'y' = ave{x*y} - X*Y. Missing values are allowed.

Example 1

Let *x* and *y* be dimensioned *nlat* x *mlon* x *ktimes* (*nlat*=64, *mlon*=128, *ktimes*=90):

```
xpyp = fluxEddy (x,y)   ; xpyp=x'y' will be an nlat x mlon array
```

Example 2

Let *u* and *v* be dimensioned *ntimes* x *klev* x *nlat* x *mlon* with named dimensions "time" , "lev" , "lat" , "lon". Then:

```
upvp = fluxEddy (u(lev|:,lat|:,lon|:,time|:), v(lev|:,lat|:,lon|:,time|:))
```

upvp will be a 3-dimensional array dimensioned *klev* x *nlat* x *mlon*.

Example 3

To calculate the eddy variance of a quantity *z*. (I.e. ave{z'z'}):

```
zpzp = fluxEddy (z,z)
```

or

```
zpzp = fluxEddy (z(lev|:,lat|:,lon|:,time|:), z(lev|:,lat|:,lon|:,time|:))
```

depending upon the dimension order.

fspan

Creates a span of evenly spaced floating point numbers.

Synopsis

```
function fspan(
        start [1] : float,
        finish [1] : float,
        num[1] : integer
)
```

Arguments

start
: floating point value to start at

finish
: floating point value to end at

num
: number of equally spaced points between *start* and *finish*

Description

fspan returns an singly dimensioned array with *num* equally spaced points from *start* to *finish* inclusive.

---

ftcurv

`ftcurv` calculates an interpolatory spline through a sequence of functional values. `ftcurv` is in the Fitgrid package -- a package containing 1D and 2D interpolators using cubic splines under tension.

Synopsis

```
function ftcurv(
    xi[*] : float,
    yi[*] : float,
    xo[*] : float
)
```

Arguments

xi
: A 1D array of any size (*npts*) containing the abscissae for the input function.

yi
: A 1D array containing the *npts* functional values of the input function -- yi(k) is the functional value at xi(k) for k=0,npts-1.

xo
: A 1D array containing the abscissae for the interpolated values.

Return value

`ftcurv` returns a 1D array that contains the interpolated functional values at the points specified in *xo*.

Description

There are some parameters that can alter the behavior of ftcurv. These parameters all have reasonable default values. However, users may change any of these parameters by invoking ftsetp prior to calling ftcurv. `ftcurv` is called *after* all of the desired values for control parameters have been set.

Control parameters that apply to `ftcurv` are: sig, sl1, sln, sf1.

The value for the parameter sig specifies the tension factor. Values near zero result in a cubic spline; large values (e.g. 50) result in nearly a polygonal line. A typical value is 1. (the default).

The values for sl1 and sln specify the slope of the curve at the first point and last point, respectively.

The value of sf1 controls whether to use the values for sl1 and sln, or compute those values internally. Specifically, sf1

= 0 if sl1 and sln are user-specified.
= 1 if sl1 is user-specified, but sln is internally calculated.
= 2 if sln is user-specified, but sl1 is internally calculated.
= 3 if sl1 and sln are internally calculated.

By default the slopes at the end points are computed internally.

You can extrapolate values with `ftcurv` (that is calculate interpolated values for abscissae outside of the domain of the input), but these values are, in general, unreliable.

Example

```
begin
  xi = (/  0.00,   2.00,   5.00,   8.00,  10.00,  13.00,      \
          15.00,  18.00,  21.00,  23.00,  30.00           /)
  yi = (/  1.00,   0.81,   0.00,  -0.81,  -1.00,  -0.84,      \
          -0.56,   0.04,   0.73,   1.18,   2.0            /)

  npts = 201
  xo   = fspan(0.,30.,npts)

  yo = ftcurv(xi, yi, xo)
end
```

ftcurvd

`ftcurvd` calculates the derivatives of an interpolatory spline under tension. `ftcurvd` is in the Fitgrid package -- a package containing 1D and 2D interpolators using cubic splines under tension.

Synopsis

```
function ftcurvd(
    xi[*] : float,
    yi[*] : float,
    xo[*] : float
)
```

Arguments

xi
: A 1D array of any size (*npts*) containing the abscissae for the input function.

yi
: A 1D array containing the *npts* functional values of the input function -- yi(k) is the functional value at xi(k) for k=0,npts-1.

xo
: A 1D array containing the abscissae for the calculated derivatives.

Return value

`ftcurvd` returns a 1D array that contains the derivatives of the interpolated curve at the points specified in *xo*.

Description

There are some parameters that can alter the behavior of ftcurvd. These parameters all have reasonable default values. However, users may change any of these parameters by invoking ftsetp prior to calling ftcurvd. `ftcurvd` is called *after* all of the desired values for control parameters have been set.

Control parameters that apply to `ftcurvd` are: sig, sl1, sln, sf1.

The value for the parameter sig specifies the tension factor. Values near zero result in a cubic spline; large values (e.g. 50) result in nearly a polygonal line. A typical value is 1. (the default).

The values for sl1 and sln specify the slope of the curve at the first point and last point, respectively.

The value of sf1 controls whether to use the values for sl1 and sln, or compute those values internally.

Specifically, sf1

= 0 if sl1 and sln are user-specified.
= 1 if sl1 is user-specified, but sln is internally calculated.
= 2 if sln is user-specified, but sl1 is internally calculated.
= 3 if sl1 and sln are internally calculated.

By default the slopes at the end points are computed internally.

You can extrapolate values with `ftcurvd` (that is calculate interpolated values for abscissae outside of the domain of the input), but these values are, in general, unreliable.

Example

```
begin

xi = (/  0.00,   2.00,   5.00,   8.00,  10.00,  13.00,      \
        15.00,  18.00,  21.00,  23.00,  30.00          /)
yi = (/  1.00,   0.81,   0.00,  -0.81,  -1.00,  -0.84,      \
        -0.56,   0.04,   0.73,   1.18,   2.0           /)

npts = 201
xo   = fspan(0.,30.,npts)

yd = ftcurvd(xi, yi, xo)
```

---

ftcurvi

`ftcurvi` calculates integrals of an interpolatory spline under tension between two user-specified limits. `ftcurvi` is in the Fitgrid package -- a package containing 1D and 2D interpolators using cubic splines under tension.

Synopsis

```
    function ftcurvi(
        xl[1] : float,
        xr[1]:  float,
        xi[*] : float,
        yi[*] : float,
    )
```

Arguments

xl

A scalar value containing the lower limit of the integration.

xr

A scalar value containing the upper limit of the integration.

xi

A 1D array of any size (*npts*) containing the abscissae for the input function.

yi

A 1D array containing the *npts* functional values of the input function -- yi(k) is the functional value at xi(k) for k=0,npts-1.

Return value

`ftcurvi` returns a scalar value that contains the integral of the interpolated function from *xl* to *xr*.

Description

There are some parameters that can alter the behavior of ftcurvi. These parameters all have reasonable default values. However, users may change any of these parameters by invoking ftsetp prior to calling ftcurvi. `ftcurvi` is called *after* all of the desired values for control parameters have been set.

Control parameters that apply to `ftcurvi` are: sig, sl1, sln, sf1.

The value for the parameter sig specifies the tension factor. Values near zero result in a cubic spline; large values (e.g. 50) result in nearly a polygonal line. A typical value is 1. (the default).

The values for sl1 and sln specify the slope of the curve at the first point and last point, respectively.

The value of sf1 controls whether to use the values for sl1 and sln, or compute those values internally. Specifically, sf1

= 0 if sl1 and sln are user-specified.
= 1 if sl1 is user-specified, but sln is internally calculated.
= 2 if sln is user-specified, but sl1 is internally calculated.
= 3 if sl1 and sln are internally calculated.

By default the slopes at the end points are computed internally.

You can extrapolate values with `ftcurvi` (that is calculate interpolated values for abscissae outside of the domain of the input), but these values are, in general, unreliable.

Example

```
begin
  xi = (/  0.00,   2.00,   5.00,   8.00,  10.00,  13.00,     \
          15.00,  18.00,  21.00,  23.00,  30.00          /)
  yi = (/  1.00,   0.81,   0.00,  -0.81,  -1.00,  -0.84,     \
          -0.56,   0.04,   0.73,   1.18,   2.0           /)

  npts = 201
  xo   = fspan(0.,30.,npts)

  integral = ftcurvi(10., 30., xi, yi)
end
```

---

ftcurvp

`ftcurvp` calculates an interpolatory spline under tension through a sequence of functional values for a periodic function. `ftcurvp` is in the Fitgrid package -- a package containing 1D and 2D interpolators using cubic splines under tension.

Synopsis

```
function ftcurvp(
    xi[*] : float,
    yi[*] : float,
    p[1]  : float,
    xo[*] : float
)
```

Arguments

xi

A 1D array of any size (*npts*) containing the abscissae for the input function.

yi

A 1D array containing the *npts* functional values of the input function -- yi(k) is the functional value at xi(k) for k=0,npts-1.

p

A scalar value specifying the period of the input function. The value of p must not be less than xi(n-1) - xi(0).

xo

A 1D array containing the abscissae for the interpolated values.

Return value

`ftcurvp` returns a 1D array that contains the interpolated functional values at the points specified in *xo*.

Description

There are some parameters that can alter the behavior of ftcurvp. These parameters all have reasonable default values. However, users may change any of these parameters by invoking ftsetp prior to calling ftcurvp. `ftcurvp` is called *after* all of the desired values for control parameters have been set.

The only control parameter that applies to `ftcurvp` is: sig.

The value for the parameter sig specifies the tension factor. Values near zero result in a cubic spline; large values (e.g. 50) result in nearly a polygonal line. A typical value is 1. (the default).

You can extrapolate values with `ftcurvp` (that is calculate interpolated values for abscissae outside of the domain of the input), but these values are, in general, unreliable.

Example

```
begin
  xi = (/  0.00,   2.00,   5.00,   8.00,  10.00,  13.00,      \
        15.00,  18.00,  21.00,  23.00,  30.00          /)
  yi = (/  1.00,   0.81,   0.00,  -0.81,  -1.00,  -0.84,      \
        -0.56,   0.04,   0.73,   1.18,   2.0           /)

  npts = 201
  xo   = fspan(0.,35.,npts)

  yo = ftcurvp(xi, yi, 31., xo)
end
```

ftcurvpi

`ftcurvpi` calculates an integral of an interpolatory spline between two specified points. `ftcurvpi` is in the Fitgrid package -- a package containing 1D and 2D interpolators using cubic splines under tension.

Synopsis

```
    function ftcurvpi(
        xl[1] : float,
        xr[1] : float,
        p[1]  : float,
        xi[*] : float,
        yi[*] : float
```

```
    )
```

Arguments

xl
: A scalar value containing the lower limit of the integration.

xr
: A scalar value containing the upper limit of the integration.

p
: A scalar value specifying the period of the input function. The value of p must not be less than xi(n-1) - xi(0).

xi
: A 1D array of any size (*npts*) containing the abscissae for the input function.

yi
: A 1D array containing the *npts* functional values of the input function -- yi(k) is the functional value at xi(k) for k=0,npts-1.

Return value

`ftcurvpi` returns a scalar value that contains the integral of the interpolated function from *xl* to *xr*.

Description

There are some parameters that can alter the behavior of ftcurvpi. These parameters all have reasonable default values. However, users may change any of these parameters by invoking ftsetp prior to calling ftcurvpi. `ftcurvpi` is called *after* all of the desired values for control parameters have been set.

The only control parameter that applies to `ftcurvpi` is: sig.

The value for the parameter sig specifies the tension factor. Values near zero result in a cubic spline; large values (e.g. 50) result in nearly a polygonal line. A typical value is 1. (the default).

You can extrapolate values with `ftcurvpi` (that is calculate interpolated values for abscissae outside of the domain of the input), but these values are, in general, unreliable.

Example

```
begin
  xi = (/  0.00,   2.00,   5.00,   8.00,  10.00,  13.00,     \
```

```
          15.00,   18.00,   21.00,   23.00,   30.00          /)
  yi = (/  1.00,    0.81,    0.00,   -0.81,   -1.00,   -0.84,     \
          -0.56,    0.04,    0.73,    1.18,    2.0            /)

  npts = 201
  xo   = fspan(0.,35.,npts)

  integral = ftcurvpi(10., 30., 31., xi, yi)
end
```

---

ftcurvps

`ftcurvps` calculates a smoothing spline. `ftcurvps` is in the Fitgrid package -- a package containing 1D and 2D interpolators using cubic splines under tension.

Synopsis

```
    function ftcurvps(
        xi[*] : float,
        yi[*] : float,
        p[1]  : float,
        d[*]  : float,
        xo[*] : float
    )
```

Arguments

xi

A 1D array of any size (*npts*) containing the abscissae for the input function.

yi

A 1D array containing the *npts* functional values of the input function -- yi(k) is the functional value at xi(k) for k=0,npts-1.

p

A scalar specifying the period of the function.

d

A user-specified value containing the observed weights. d may either be a 1D array (of length *npts*) or a scalar.

xo

A 1D array containing the abscissae for the interpolated values.

Return value

`ftcurvps` returns a 1D array that contains the interpolated functional values of the smooting spline at the points specified in *xo*.

Description

There are some parameters that can alter the behavior of ftcurvps. These parameters all have reasonable default values. However, users may change any of these parameters by invoking ftsetp prior to calling ftcurvps. `ftcurvps` is called *after* all of the desired values for control parameters have been set.

Control parameters that apply to `ftcurvps` are: sig, smt, eps, sf2.

Two parameters and one function argument used to control the degree of smoothness -- the parameters are smt, and eps and the function argument is d.

The argument d is a value indicating the degree of confidence in the accuracy of the input function values -- it should be an approximation of the standard deviation of error. Effectively the value of d controls how close the smoothed curve comes to the input data points. If d is small, then the interpolated curve will pass close to the input data. The larger the value of d, the more freedom the smooth curve has in how close it comes to the input data values.

The parameter smt is a more subtle global smoothing parameter; smt must be non-negative. For small values of smt, the curve approximates the tension spline and for larger values of smt, the curve is smoother. A reasonable value for smt is (float) n.

The parameter eps controls the precision to which smt is interpreted; eps must be between 0. and 1. inclusive. A reasonable value for eps is sqrt(2./(float) n).

The value for the parameter sig specifies the tension factor. Values near zero result in a cubic spline; large values (e.g. 50) result in nearly a polygonal line. A typical value is 1. (the default).

You can extrapolate values with `ftcurvps` (that is calculate interpolated values for abscissae outside of the domain of the input), but these values are, in general, unreliable.

Example

```
begin
  xi = (/  0.00,   2.00,   5.00,   8.00,  10.00,  13.00,     \
          15.00,  18.00,  21.00,  23.00,  30.00          /)
  yi = (/  1.00,   0.81,   0.00,  -0.81,  -1.00,  -0.84,     \
          -0.56,   0.04,   0.73,   1.18,   2.0           /)

  npts = 201
  xo   = fspan(0.,35.,npts)

  yo = ftcurvps(xi, yi, 31., 0.1, xo)
end
```

ftcurvs

`ftcurvs` calculates a smoothing spline. `ftcurvs` is in the Fitgrid package -- a package containing 1D and 2D interpolators using cubic splines under tension.

Synopsis

```
function ftcurvs(
    xi[*] : float,
    yi[*] : float,
    d[*]  : float,
    xo[*] : float
)
```

Arguments

xi
: A 1D array of any size (*npts*) containing the abscissae for the input function.

yi
: A 1D array containing the *npts* functional values of the input function -- yi(k) is the functional value at xi(k) for k=0,npts-1.

d
: A user-specified value containing the observed weights. d may either be a 1D array (of length *npts*) or a scalar.

xo
: A 1D array containing the abscissae for the interpolated values.

Return value

`ftcurvs` returns a 1D array that contains the interpolated functional values of the smooting spline at the points specified in *xo*.

Description

There are some parameters that can alter the behavior of ftcurvs. These parameters all have reasonable default values. However, users may change any of these parameters by invoking ftsetp prior to calling ftcurvs. `ftcurvs` is called *after* all of the desired values for control parameters have been set.

Control parameters that apply to `ftcurvs` are: sig, smt, eps, sf2.

Two parameters and one function argument used to control the degree of smoothness -- the parameters are smt, and eps and the function argument is d.

The argument d is a value indicating the degree of confidence in the accuracy of the input function values -- it should be an approximation of the standard deviation of error. Effectively the value of d controls how close the smoothed curve comes to the input data points. If d is small, then the interpolated curve will pass close to the input data. The larger the value of d, the more freedom the smooth curve has in how close it comes to the input data values.

The parameter smt is a more subtle global smoothing parameter; smt must be non-negative. For small values of smt, the curve approximates the tension spline and for larger values of smt, the curve is smoother. A reasonable value for smt is (float) n.

The parameter eps controls the precision to which smt is interpreted; eps must be between 0. and 1. inclusive. A reasonable value for eps is sqrt(2./(float) n).

The value for the parameter sig specifies the tension factor. Values near zero result in a cubic spline; large values (e.g. 50) result in nearly a polygonal line. A typical value is 1. (the default).

You can extrapolate values with `ftcurvs` (that is calculate interpolated values for abscissae outside of the domain of the input), but these values are, in general, unreliable.

Example

```
begin
  xi = (/  0.00,   2.00,   5.00,   8.00,  10.00,  13.00,     \
         15.00,  18.00,  21.00,  23.00,  30.00          /)
  yi = (/  1.00,   0.81,   0.00,  -0.81,  -1.00,  -0.84,     \
         -0.56,   0.04,   0.73,   1.18,   2.0           /)

  npts = 201
  xo   = fspan(0.,30.,npts)

  yo = ftcurvs(xi, yi, 0.1, xo)
end
```

---

ftkurv

`ftkurv` calculates an interpolatory spline for parametric curves. `ftkurv` is in the Fitgrid package -- a package containing 1D and 2D interpolators using cubic splines under tension.

Synopsis

```
procedure ftkurv(
    xi[*] : float,
    yi[*] : float,
    t[*]  : float,
    xo[*] : float,
    yo[*] : float
)
```

Arguments

xi
: A 1D array of any size (*npts*) containing the abscissae for the input function.

yi
: A 1D array containing the *npts* functional values of the input function -- yi(k) is the functional value at xi(k) for k=0,npts-1.

t
: Contains a 1D array of *mpts* values for the parameter mapping onto the interpolated curve.

xo
: A 1D array containing the X values for the interpolated points. t[k] maps to (xo[k],yo[k]) for k=0,mpts-1.

yo
: A 1D array An array containing the Y values for the interpolated points. t[k] maps to (xo[k],yo[k]) for k=0,mpts-1.

Description

There are some parameters that can alter the behavior of ftkurv. These parameters all have reasonable default values. However, users may change any of these parameters by invoking ftsetp prior to calling ftkurv. `ftkurv` is called *after* all of the desired values for control parameters have been set.

Control parameters that apply to `ftkurv` are: sig, sl1, sln, sf1.

Given a sequence of input points ((xi[0],yi[0]), ... , (xi[npts-1],yi[npts-1]), the interpolated curve is parameterized by mapping points in the interval [0.,1.] onto the interpolated curve. The resulting curve has a parametric representation both of whose components are splines under tension and functions of the polygonal arc length. The value 0. is mapped onto (xi[0],yi[0]) and the value 1. is mapped onto (xi[mpts-1],yi[mpts-1]). Values outside the interval [0.,1.] are mapped to extrapolated values.

The value for the parameter sig specifies the tension factor. Values near zero result in a cubic spline; large values (e.g. 50) result in nearly a polygonal line. A typical value is 1. (the default).

The value for parameter sl1 is in radians and contains the slope at (xi[0],yi[0]). The angle is measured counter-clockwise from the X axis and the positive sense of the curve is assumed to be that moving from point 0 to point npts-1. A value for sl1 may be omitted as indicated by the switch sf1.

The value for parameter sln is in radians and contains the slope at (xi[npts-1],yi[npts-1]). The angle is measured counter-clockwise from the X axis and the positive sense of the curve is assumed to be that moving from point 0 to point npts-1. A value for sln may be omitted as indicated by the switch sf1.

The value of sf1 controls whether to use the values for sl1 and sln, or compute those values internally. Specifically, sf1

> = 0 if sl1 and sln are user-specified.
> = 1 if sl1 is user-specified, but sln is internally calculated.
> = 2 if sln is user-specified, but sl1 is internally calculated.
> = 3 if sl1 and sln are internally calculated.

You can extrapolate values with `ftkurv` (that is calculate interpolated values for abscissae outside of the domain of the input), but these values are, in general, unreliable.

Example

```
begin
  x = (/  3.0,  4.0,  9.0, 16.0, 21.0, 27.0,    \
        34.0, 36.0, 34.0, 26.0, 18.0          /)
  y = (/  2.4,  9.6, 14.4, 12.0,  9.6,  8.4,    \
        13.2, 21.6, 30.0, 37.2, 38.4          /)

  mpts = 201
  u = fspan(0.,1.,mpts)

  xo = new( (/ mpts /), float)
  yo = new( (/ mpts /), float)
  ftkurv(x, y, u, xo, yo)
end
```

---

ftkurvd

`ftkurvd` calculates an interpolatory spline for parametric curves; it also calculates first and second derivatives of the interpolatory spline. `ftkurvd` is in the Fitgrid package -- a package containing 1D and 2D interpolators using cubic splines under tension.

Synopsis

```
    procedure ftkurvd(
        xi[*]  : float,
        yi[*]  : float,
        t[*]   : float,
        xo[*]  : float,
```

```
        yo[*]  : float,
        xd[*]  : float,
        yd[*]  : float,
        xdd[*] : float,
        ydd[*] : float
    )
```

Arguments

xi
: A 1D array of any size (*npts*) containing the abscissae for the input function.

yi
: A 1D array containing the *npts* functional values of the input function -- yi(k) is the functional value at xi(k) for k=0,npts-1.

t
: Contains a 1D array of *mpts* values for the parameter mapping onto the interpolated curve.

xo
: A 1D array containing the X values for the interpolated points. t[k] maps to (xo[k],yo[k]) for k=0,mpts-1.

yo
: A 1D array An array containing the Y values for the interpolated points. t[k] maps to (xo[k],yo[k]) for k=0,mpts-1.

xd
: A 1D array containing the first derivatives of the X component with respect to t.

yd
: A 1D array containing the first derivatives of the Y component with respect to t.

xdd
: A 1D array containing the second derivatives of the X component with respect to t.

ydd
: A 1D array containing the second derivatives of the Y component with respect to t.

Description

There are some parameters that can alter the behavior of ftkurvd. These parameters all have reasonable default values. However, users may change any of these parameters by invoking ftsetp prior to calling ftkurvd. `ftkurvd` is called *after* all of the desired values for control parameters have been set.

Control parameters that apply to `ftkurvd` are: sig, sl1, sln, sf1.

Given a sequence of input points ((xi[0],yi[0]), ... , (xi[npts-1],yi[npts-1]), the interpolated curve is parameterized by mapping points in the interval [0.,1.] onto the interpolated curve. The resulting curve has a parametric representation both of whose components are splines under tension and functions of the polygonal arc length. The value 0. is mapped onto (xi[0],yi[0]) and the value 1. is mapped onto (xi[mpts-1],yi[mpts-1]). Values outside the interval [0.,1.] are mapped to extrapolated values.

The value for the parameter sig specifies the tension factor. Values near zero result in a cubic spline; large values (e.g. 50) result in nearly a polygonal line. A typical value is 1. (the default).

The value for parameter sl1 is in radians and contains the slope at (xi[0],yi[0]). The angle is measured counter-clockwise from the X axis and the positive sense of the curve is assumed to be that moving from point 0 to point npts-1. A value for sl1 may be omitted as indicated by the switch sf1.

The value for parameter sln is in radians and contains the slope at (xi[npts-1],yi[npts-1]). The angle is measured counter-clockwise from the X axis and the positive sense of the curve is assumed to be that moving from point 0 to point npts-1. A value for sln may be omitted as indicated by the switch sf1.

The value of sf1 controls whether to use the values for sl1 and sln, or compute those values internally. Specifically, sf1

> = 0 if sl1 and sln are user-specified.
> = 1 if sl1 is user-specified, but sln is internally calculated.
> = 2 if sln is user-specified, but sl1 is internally calculated.
> = 3 if sl1 and sln are internally calculated.

You can extrapolate values with `ftkurvd` (that is calculate interpolated values for abscissae outside of the domain of the input), but these values are, in general, unreliable.

Example

```
begin
  x = (/  3.0,  4.0,  9.0, 16.0, 21.0, 27.0,    \
        34.0, 36.0, 34.0, 26.0, 18.0          /)
  y = (/  2.4,  9.6, 14.4, 12.0,  9.6,  8.4,    \
        13.2, 21.6, 30.0, 37.2, 38.4          /)

  mpts = 201
  u = fspan(0.,1.,mpts)

  xo  = new( (/ mpts /), float)
  yo  = new( (/ mpts /), float)
  xd  = new( (/ mpts /), float)
  yd  = new( (/ mpts /), float)
  xdd = new( (/ mpts /), float)
  ydd = new( (/ mpts /), float)
  ftkurvd(x, y, u, xo, yo, xd, yd, xdd, ydd)
end
```

---

ftkurvp

`ftkurvp` calculates an interpolatory spline under tension through a sequence of points in the plane forming a closed curve. `ftkurvp` is in the Fitgrid package -- a package containing 1D and 2D interpolators using cubic splines under tension.

Synopsis

```
procedure ftkurvp(
    xi[*] : float,
    yi[*] : float,
    t[*]  : float,
    xo[*] : float,
    yo[*] : float
)
```

Arguments

xi
: A 1D array of any size (*npts*) containing the abscissae for the input function.

yi
: A 1D array containing the *npts* functional values of the input function -- yi(k) is the functional value at xi(k) for k=0,npts-1.

t
: Contains a 1D array of *mpts* values for the parameter mapping onto the interpolated curve.

xo
: A 1D array containing the X values for the interpolated points. t[k] maps to (xo[k],yo[k]) for k=0,mpts-1.

yo
: A 1D array An array containing the Y values for the interpolated points. t[k] maps to (xo[k],yo[k]) for k=0,mpts-1.

Description

There are some parameters that can alter the behavior of ftkurvp. These parameters all have reasonable default values. However, users may change any of these parameters by invoking ftsetp prior to calling ftkurvp. `ftkurvp` is called *after* all of the desired values for control parameters have been set.

The only control parameter that applies to ftkurvp is: sig.

Given a sequence of distinct input points ((x[0],y[0]), ... , (x[n-1],y[n-1]), the interpolated curve is parameterized by mapping points in the interval [0.,1.] onto the interpolated curve. The resulting curve has a parametric representation both of whose components are splines under tension and functions of the polygonal arc length. The value 0. is mapped onto (x[0],y[0]) and the value 1. is mapped onto (x[0],y[0]) as well (completing the closed curve).

The value for the parameter sig specifies the tension factor. Values near zero result in a cubic spline; large values (e.g. 50) result in nearly a polygonal line. A typical value is 1. (the default).

You can extrapolate values with `ftcurvp` (that is calculate interpolated values for abscissae outside of the domain of the input), but these values are, in general, unreliable.

Example

```
begin
  x = (/  3.0,  4.0,  9.0, 16.0, 21.0, 27.0,    \
        34.0, 36.0, 34.0, 26.0, 18.0         /)
  y = (/  2.4,  9.6, 14.4, 12.0,  9.6,  8.4,    \
        13.2, 21.6, 30.0, 37.2, 38.4         /)

  mpts = 201
  u = fspan(0.,1.,mpts)

  xo = new( (/ mpts /), float)
  yo = new( (/ mpts /), float)
  ftkurvp(x, y, u, xo, yo)
end
```

---

ftkurvpd

`ftkurvpd` calculates an interpolatory spline for closed parametric curves; it also calculates first and second derivatives of the interpolatory spline. `ftkurvpd` is in the Fitgrid package -- a package containing 1D and 2D interpolators using cubic splines under tension.

Synopsis

```
    procedure ftkurvpd(
        xi[*]  : float,
        yi[*]  : float,
        t[*]   : float,
        xo[*]  : float,
        yo[*]  : float,
        xd[*]  : float,
        yd[*]  : float,
        xdd[*] : float,
        ydd[*] : float
    )
```

Arguments

xi

A 1D array of any size (*npts*) containing the abscissae for the input function.

yi

A 1D array containing the *npts* functional values of the input function -- yi(k) is the functional value at xi(k) for k=0,npts-1.

t

Contains a 1D array of *mpts* values for the parameter mapping onto the interpolated curve.

xo

A 1D array containing the X values for the interpolated points. t[k] maps to (xo[k],yo[k]) for k=0,mpts-1.

yo

A 1D array An array containing the Y values for the interpolated points. t[k] maps to (xo[k],yo[k]) for k=0,mpts-1.

xd

A 1D array containing the first derivatives of the X component with respect to t.

yd

A 1D array containing the first derivatives of the Y component with respect to t.

xdd

A 1D array containing the second derivatives of the X component with respect to t.

ydd

A 1D array containing the second derivatives of the Y component with respect to t.

Description

There are some parameters that can alter the behavior of ftkurvpd. These parameters all have reasonable default values. However, users may change any of these parameters by invoking ftsetp prior to calling ftkurvpd. `ftkurvpd` is called *after* all of the desired values for control parameters have been set.

The only control parameter that applies to ftkurvpd is: sig.

Given a sequence of distinct input points ((x[0],y[0]), ... , (x[n-1],y[n-1]), the interpolated curve is parameterized by mapping points in the interval [0.,1.] onto the interpolated curve. The resulting curve has a parametric representation both of whose components are splines under tension and functions of the polygonal arc length. The value 0. is mapped onto (x[0],y[0]) and the value 1. is mapped onto (x[0],y[0]) as well (completing the closed curve).

The value for the parameter sig specifies the tension factor. Values near zero result in a cubic spline; large values (e.g. 50) result in nearly a polygonal line. A typical value is 1. (the default).

You can extrapolate values with `ftcurvpd` (that is calculate interpolated values for abscissae outside of the domain of the input), but these values are, in general, unreliable.

Example

```
begin
  x = (/  3.0,   4.0,   9.0, 16.0, 21.0, 27.0,     \
         34.0, 36.0, 34.0, 26.0, 18.0          /)
```

```
  y = (/  2.4,   9.6, 14.4, 12.0,   9.6,   8.4,     \
         13.2, 21.6, 30.0, 37.2, 38.4          /)

  mpts = 201
  u = fspan(0.,1.,mpts)

  xo  = new( (/ mpts /), float)
  yo  = new( (/ mpts /), float)
  xd  = new( (/ mpts /), float)
  yd  = new( (/ mpts /), float)
  xdd = new( (/ mpts /), float)
  ydd = new( (/ mpts /), float)
  ftkurvpd(x, y, u, xo, yo, xd, yd, xdd, ydd)
end
```

---

ftsurf

`ftsurf` calculates an interpolatory surface passing through a rectangular grid of function values. The surface computed is a tensor product of splines under tension. `ftsurf` is in the Fitgrid package -- a package containing 1D and 2D interpolators using cubic splines under tension.

Synopsis

```
    function ftsurf(
        xi[*]     : float,
        yi[*]     : float,
        zi[*][*] : float,
        xo[*]     : float,
        yo[*]     : float
    )
```

Arguments

xi

A 1D array of arbitrary length (say *mi*) containing X coordinates for grid lines in the X direction.

yi

A 1D array of arbitrary length (say *ni*) containing Y coordinates for grid lines in the X direction.

zi

An array of size *mi* x *ni* which contains the functional values at the grid points defined by *xi* and *yi*.

xo

A 1D array specifying the X (say *mo* in number) coordinates for the output interpolation grid.

yo

A 1D array specifying the Y (say *no* in number) coordinates for the output interpolation grid.

Return value

`ftsurf` returns a 2D array of interpolated values defined on the specified output grid. `ftsurf` is of size *mo* x *no* , where mo is the size of *xo* and no is the size of *yo*.

`ftsurf` does not recognize missing values.

Description

There are some parameters that can alter the behavior of ftstep. These parameters all have reasonable default values. However, users may change any of these parameters by invoking ftsetp prior to calling ftsurf. `ftsurf` is called *after* all of the desired values for control parameters have been set.

Control parameters that apply to `ftsurf` are: sig, zx1, zxm, zy1, zyn, z11, zm1, z1n, zmn, df1, df2, df3, df4, df5, df6, df7, df8.

The value for the parameter sig specifies the tension factor. Values near zero result in a cubic spline; large values (e.g. 50) result in nearly a polygonal line. A typical value is 1. (the default).

zx1 is an array containing mi X-partial derivatives of the function along the line xi[0], that is zx1[j] is the X-partial derivative at point (x[0],y[j]) for j=0,ni-1. This parameter may be defaulted by setting the value for df1 appropriately. The default is to compute zx1 internally. Values for zx1 can be set using the procedure ftsetp.

zxm is an array containing n X-partial derivatives of the function along the line xi[mi-1], that is zxm[j] is the X-partial derivative at point (xi[mi-1],yi[j]) for j=0,ni-1. This parameter may be defaulted by setting the value for df2 appropriately. The default is to compute zx2 internally. Values for zxm can be set using the procedure ftsetp.

zy1 is an array containing m Y-partial derivatives of the function along the line yi[0], that is zy1[j] is the Y-partial derivative at point (x[i],y[0]) for i=0,mi-1. This parameter may be defaulted by setting the value for df3 appropriately. The default is to compute zy1 internally. Values for zy1 can be set using the procedure ftsetp.

zyn is an array containing m Y-partial derivatives of the function along the line yi[ni-1], that is zyn[j] is the Y-partial derivative at point (x[i],y[ni-1]) for i=0,mi-1. This parameter may be defaulted by setting the value for df4 appropriately. The default is to compute zyn internally. Values for zyn can be set using the procedure ftsetp.

z11, zm1, z1n, zmn specify X-Y-partial derivatives of the function at the four corners (xi[0],yi[0]), (xi[mi-1],yi[0]), (xi[0],yi[ni-1]), (xi[mi-1],yi[ni-1]), These parameters may be defaulted by setting the values for df5, df6, df7, df8, appropriately. The default is to compute z11, zm1, z1n, zmn internally.

You can extrapolate values with `ftsurv` (that is calculate interpolated values for coordinates outside of the scope of the input grid), but these values are, in general, unreliable.

Example

```
begin
  nxi = 11
  nyi = 17
  xi = fspan(0.,1.,nxi)
  yi = fspan(0.,1.,nyi)
  zi = new((/nxi,nyi/),float)
  do i=0,nxi-1
    do j=0,nyi-1
      zi(i,j) = 0.5 + 0.25*sin(-7.*xi(i)) + 0.25*cos(5.*yi(j))
    end do
  end do
  nxo = 31
  nyo = 21
  xo = fspan(0.,1.,nxo)
  yo = fspan(0.,1.,nyo)
  zo = ftsurf(xi,yi,zi,xo,yo)
end
```

$ $

ftsetp, ftgetp

Set and retrieve parameters for Fitgrid routines. NCL affords a convenient implementation of these parameter setting and retrieval functions.

Synopsis

```
    procedure ftsetp(
        pnam[1] : string,
        pval[1]
    )

    function ftgetp(
        pnam[1] : string,
    )
```

Arguments

pnam
: name of the parameter you want to set or retrieve

pval
: value of the parameter you want to set; this value must be of the type appropriate to the parameter being set.

Description

The procedure `ftsetp` is used to set values for fitgrid parameters; the function `ftgetp` is used to retrieve the current value of the named parameter. `ftgetp` returns a value that is of the type appropriate to the parameter type.

Example

```
begin
  ftsetp("sig",0.9)
  cur_sig = ftgetp("sig")
  print(cur_sig)
end
```

f2fsh, f2gsh, g2fsh, g2gsh, fo2fsh, f2fosh

Grid-to-grid functions for scalar quantities using Spherepack routines.

Synopsis

```
    function f2fsh(
        grid       : float,
        outdims[2] : integer
    )

    function g2fsh(
        grid       : float,
        outdims[2] : integer
    )

    function f2gsh(
        grid       : float,
        outdims[2] : integer,
        twave[1]   : integer
    )

    function g2gsh(
        grid       : float,
        outdims[2] : integer,
        twave[1]   : integer
    )

    function fo2fsh(
        grid : float
    )
```

```
function f2fosh(
    grid : float
)
```

Arguments

grid

input float array of 2 or more dimensions (last two dimensions must be *nlata* x *nlona*, values must be in ascending latitude order)

outdims

indicates dimensions of rightmost two dimensions of output grid (outdims[0] = *nlatb*, outdims[1] = *nlonb*)

twave

scalar integer indicating the optional wave truncation:

twave = 0 => exact interpolation
twave > 0 => exact interpolation and triangular truncation at *twave*
twave < 0 => exact interpolation, triangular truncation at *twave* and spectral coefficient tapering (the effect is to smooth the data)

Description

Fixed or regular grids have grid points that align with the Grenwich meridian and the equator. Eg: a 2.5 x 2.5 degree grid will have values at the equator:

```
(-90  ,0), (-90,  2.5),..., (-90,  357.5)
(-87.5,0), (-87.5,2.5),..., (-87.5,357.5)
                     .
(0,0), (0,2.5), (0,5.0),..., (0,357.5)
(2.5,0),(2.5,2.5),..., (2.5,357.5)
```

For a 2.5 degree fixed/regular grid there are 73 latitude points and 144 longitude points. Both pole points are present.

Fixed offset grids are grids 'offset' from Grenwich meridian and equator. For example, a 2.5 x 2.5 degree fixed-offset grid has values at:

```
(-88.75,1.25), (-88.75,3.75),..., (-88.75,357.5)

                   .

(1.25,1.25),   (1.25,3.75),...,(1.25,358.75)

(3,75,1.25),   (3.75,3.75),...,(3.75,358.75)
```

For a 2.5 degree fixed-offset grid there are 72 latitude points and 144 longitude points. There are no pole points.

`g2gsh` interpolates a scalar quantity from one Gaussian grid to another (optional truncation) using spherical harmonics (values will be in ascending latitude order). The output array's dimensions are the same as *grid*'s dimensions, except *nlata* and *nlona* are replaced by *nlatb* and *nlonb*.

`g2fsh` interpolates a scalar quantity on a Gaussian grid to a fixed grid using spherical harmonics (values will be in ascending latitude order). The output array's dimensions are the same as *grid*'s dimensions, except *nlata* and *nlona* are replaced by *nlatb* and *nlonb*.

`f2gsh` interpolates a scalar quantity on a fixed grid to a Gaussian (optional truncation) grid using spherical harmonics (values will be in ascending latitude order). The output array's dimensions are the same as *grid*'s dimensions, except *nlata* and *nlona* are replaced by *nlatb* and *nlonb*.

`f2fsh` interpolates a scalar quantity from one fixed grid to another using spherical harmonics (values will be in ascending latitude order). The output array's dimensions are the same as *grid*'s dimensions, except *nlata* and *nlona* are replaced by *nlatb* and *nlonb*.

`fo2fsh` interpolates a scalar quantity on a fixed-offset grid to a fixed grid using spherical harmonics (values will be in ascending latitude order). The output array dimensions are the same as *grid*'s dimensions, except the latitude dimension (*nlata*) is increased by one. For example, a 5 degree fixed-offset grid would have dimensions nlata=36, nlona=72 and the return fixed grid would have dimensions nlatb=37, nlonb=72.

`f2fosh` interpolates a scalar quantity on a fixed grid (including pole points) to a fixed-offset grid using spherical harmonics (values will be in ascending latitude order). The output array dimensions are the same as *grid*'s dimensions, except the latitude dimension (*nlata*) is decreased by one. For example, a 5 degree fixed grid would have dimensions nlata=37, nlona=72, and the return fixed-offset grid would have dimensions nlatb=36, nlonb=72.

Arrays which have dimensions [...] x *nlat* x *nlon* should **not** include the cyclic (wraparound) points when invoking the procedures and functions which use spherical harmonics (Spherepack).

For example, if an array *x* has dimensions *nlat* = 64 and *nlon* = 129, where the "129" represents the cyclic points, then the user should pass the data to the procedure/function via:

```
    z = sample ( x([...] :,0:nlon-2) )  ; does not include cyclic points
```

Example

```
begin

  jlat  =  94                      ; NCEP grid
  ilon  = 192
  nga   =   3                      ; there will be "nga" (192,94) grids
  grida = new( (/nga,jlat,ilon/), float)

  nlat  = 40
```

```
  mlon  = 48
  ngb   =  3                          ; there will be "ngb" (48,40) grids
  gridb = new( (/ngb,nlat,mlon/), float)

;
; Read in the source grid.
; This is the NCEP grid created as a direct access fortran file
;
  grida(0,:,:) = cbinread ("T2m_T62.7901" ,(/jlat,ilon/), "float" )
;
; Put the North pole at the top of the plot.
;
  grida(0,:,:) = grida(0,::-1,:)

  print ("grida mid point:")
  print (grida(0,jlat/2,ilon/2))
;
; Create grids with different resolutions.
; Gaussian =>fixed equally spaced
;
  llat  = 180                           ; 1.0
  klon  = 360                           ; 1.0
  grid1 = g2fsh (grida(0,:,:), (/llat,klon/) )
;
; Gaussian =>fixed unequally spaced
;
  gridb(0,:,:) = g2fsh (grida(0,:,:), (/nlat,mlon/) )
;
; Fixed (equal) to fixed (unequal)
;
  gridb(1,:,:) = f2fsh (grid1, (/nlat,mlon/) )
;
; Fixed => Gaussian (T62)
;
  grida(1,:,:) = f2gsh (grid1, (/jlat,ilon/), 62 )
;
; Difference fields
;
  grida(2,:,:) = grida(1,:,:) - grida(0,:,:)
  gridb(2,:,:) = gridb(1,:,:) - gridb(0,:,:)
end
```

Error messages

ier is equal to:

> 4 or 10 if *nlona* is less than 4
> 5 or 10 if *nlata* is less than 3
> 8 if *nlonb* is less than 4
> 9 if *nlatb* is less than 3

---

f2fshv, f2gshv, g2fshv, g2gshv, f2foshv, fo2fshv

Grid-to-grid procedures for vector quantities using Spherepack routines.

Synopsis

```
procedure f2fshv(
    ua    : float,
    va    : float,
    ub    : float,
    vb    : float
)

procedure g2fshv(
    ua    : float,
    va    : float,
    ub    : float,
    vb    : float
)

procedure f2gshv(
    ua    : float,
    va    : float,
    ub    : float,
    vb    : float,
    twave[1] : integer
)

procedure g2gshv(
    ua    : float,
    va    : float,
    ub    : float,
    vb    : float,
    twave[1] : integer
)

procedure fo2fshv(
    uoff  : float,
    voff  : float,
    ureg  : float,
    vreg  : float
)

procedure f2foshv(
    ureg  : float,
    vreg  : float,
    uoff  : float,
    voff  : float
)
```

Arguments

ua, va

vector arrays (input) of 2 or more dimensions (last two dimensions must be *nlata* x *nlona* and values must be in ascending latitude order)

ub, vb

vector arrays (output) of 2 or more dimensions (last two dimensions must be *nlatb* x *nlonb*, values will be in ascending latitude order)

uoff, voff

vector arrays (input or output depending on procedure) of 2 or more dimensions (last two dimensions must be *jlat* x *ilon*, values must/will be in ascending latitude order)

ureg, vreg

vector arrays (input or output depending on procedure) of 2 or more dimensions (last two dimensions must be *jlat1* x *ilon*, where *jlat1* is *jlat*+1, values must/will be in ascending latitude order)

twave

scalar integer (input) indicating the optional wave truncation:

twave = 0 => exact interpolation
twave > 0 => exact interpolation and triangular truncation at *twave*
twave < 0 => exact interpolation, triangular truncation at *twave* and spectral coefficient tapering (the effect is to smooth the data)

Description

Fixed or regular grids have grid points that align with the Grenwich meridian and the equator. Eg: a 2.5 x 2.5 degree grid will have values at the equator:

```
(-90  ,0), (-90,  2.5),..., (-90,  357.5)
(-87.5,0), (-87.5,2.5),..., (-87.5,357.5)
                   .
(0,0), (0,2.5), (0,5.0),..., (0,357.5)
(2.5,0),(2.5,2.5),..., (2.5,357.5)
```

For a 2.5 degree fixed/regular grid there are 73 latitude points and 144 longitude points. Both pole points are present.

Fixed offset grids are grids 'offset' from Grenwich meridian and equator. For example, a 2.5 x 2.5 degree fixed-offset grid has values at:

```
(-88.75,1.25), (-88.75,3.75),..., (-88.75,357.5)

                      .

(1.25,1.25),   (1.25,3.75),...,(1.25,358.75)

(3,75,1.25),   (3.75,3.75),...,(3.75,358.75)
```

For a 2.5 degree fixed-offset grid there are 72 latitude points and 144 longitude points. There are no pole points.

`g2gshv` interpolates a vector pair from one Gaussian grid (ua,va) to another (ub,vb) (optional truncation) using spherical harmonics. The output array dimensions must be the same as the input array dimensions, except *nlata* and *nlona* are replaced by *nlatb* and *nlonb*.

`g2fshv` interpolates a vector pair on a Gaussian grid (ua,va) to a fixed grid (ub,va) using spherical harmonics. The output array dimensions must be the same as the input array dimensions, except *nlata* and *nlona* are replaced by *nlatb* and *nlonb*.

`f2gshv` interpolates a vector pair on a fixed grid (ua,va) to a Gaussian grid (ub,vb) (optional truncation) using spherical harmonics. The output array dimensions must be the same as the input array dimensions, except *nlata* and *nlona* are replaced by *nlatb* and *nlonb*.

`f2fshv` interpolates a vector pair from one fixed grid (ua,va) to another (ub,vb) using spherical harmonics. The output array dimensions must be the same as the input array dimensions, except *nlata* and *nlona* are replaced by *nlatb* and *nlonb*.

`fo2fshv` interpolates a vector pair on a fixed-offset grid (uoff,voff) (including pole points) to a fixed grid (ureg,vreg) using spherical harmonics. The output array dimensions must be the same as the input array dimensions, except *jlat* is replaced by *jlat*+1. For example, a 5 degree fixed-offset grid would have dimensions nlata=36, nlona=72, and the return fixed grid would have dimensions nlatb=37, nlonb=72.

`f2foshv` interpolates a vector pair on a fixed grid (ureg,vreg) (including pole points) to a fixed-offset grid (uoff,voff) using spherical harmonics. The output array dimensions must be the same as the input array dimensions, except *jlat1* is replaced by *jlat1*-1. For example, a 5 degree fixed grid would have dimensions nlata=37, nlona=72, and the return fixed-offset grid would have dimensions nlatb=36, nlonb=72.

Arrays which have dimensions [...] x *nlat* x *nlon* should **not** include the cyclic (wraparound) points when invoking the procedures and functions which use spherical harmonics (Spherepack).

For example, if an array *x* has dimensions *nlat* = 64 and *nlon* = 129, where the "129" represents the cyclic points, then the user should pass the data to the procedure/function via:

```
z = sample ( x([...] :,0:nlon-2) )  ; does not include cyclic points
```

Example

Error messages

ier is equal to:

> 4 or 10 if *nlona* (*ilon*) is less than 4

5 or 10 if *nlata* (*jlat*) is less than 3
8 if *nlonb* is less than 4
9 if *nlatb* is less than 3

---

gaus

Computes gaussian latitudes and weights.

Synopsis

```
function gaus(
        nlat[1]:integer
)
```

Arguments

nlat
: number of latitude points PER hemisphere.

Description

Returns a two dimensional value. The 0th element of the 0th dimension contains the computed gaussian latitudes. The 1st element of the 0th dimension contains the gaussian weights.

---

getenv

This function is used to retrieve the string value of a shell environment variable.

Synopsis

```
function getenv(
        ename[1]:string
)
```

Arguments

ename
: A singly dimensioned string name of an environment parameter.

Description

The **getenv** function returns either an empty string when the shell environment variable does not exist, or the string value of the environment parameter requested.

---

getfilevaratts

Retrieves a list of attribute names for a file variable

Synopsis

```
function getfilevaratts(
        thefile [1] : file,
        varname [1] : string
)
```

Arguments

thefile
: file reference containing variable

varname
: string name of variable

Description

Returns a list of attribute names for a file variable. *getfilevaratts* returns a missing value if no attributes exist or the variable is not defined.

---

getfilevardims

Returns dimension names of file variables.

Synopsis

```
function getfilevardims(
        thefile [1] : file,
        varname [1] : string
)
```

Arguments

thefile
: reference to a file

varname
: string name of variable

Description

Returns a list of dimension names for a variable in a file. If the variable is not defined then a missing value is returned.

---

getfilevarnames

This function returns an array of file variable names for a given file.

Synopsis

```
function getfilevarnames(
        the_file [1] : file
)
```

Arguments

the_file
: A reference to a file that was added with the addfile function.

Description

This function returns an array of strings with each element containing the name of a file variable. The length of this array is equal to the number of variables in the file. This function is useful when accessing file variables by string name. (See Files.)

---

getvaratts

Returns a list of attribute names for a given variable

Synopsis

```
        function getvaratts(
                var
        )
```

Arguments

var
: any variable

Description

Returns a list of attribute names for a given variable. If the variable has no attributes the a missing value is returned.

---

gradsf, gradsg

Given array z, compute its gradient via Spherepack.

Synopsis

```
    procedure gradsf(
        z   : float,
        gzx : float,
        gzy : float
```

```
)

procedure gradsg(
    z   : float,
    gzx : float,
    gzy : float
)
```

Arguments

z

array to compute gradient of (input, two or more dimensions, last two dimensions must be *nlat* x *nlon* and input values must be in ascending latitude order).

gzx, gzy

gradient arrays (output, same dimensions as *z*, values will be in ascending latitude order)

Description

gradsf and *gradsg* both compute the gradient given the array *z* and return the results in the arrays *gzy* and *gzy*. *gradsf* operates on an equal (fixed) grid, and *gradsg* operates on a Gaussian grid.

Note: For the arrays whose last two dimensions are *nlat* x *nlon*, the rest of the dimensions (if any) are collectively referred to as *nt*. If the input/output arrays are just two dimensions, then *nt* can either be considered equal to 1 or nothing at all.

Arrays which have dimensions *nt* x *nlat* x *nlon* should **not** include the cyclic (wraparound) points when invoking the procedures and functions which use spherical harmonics (Spherepack).

For example, if an array *x* has dimensions *nlat* = 64 and *nlon* = 129, where the "129" represents the cyclic points, then the user should pass the data to the procedure/function via:

```
    z = sample ( x([...],:,0:nlon-2) ) ; does not include cyclic
points
```

Error messages

If *jer* or *ker* is equal to:

1 : error in the specification of *nlat*
2 : error in the specification of *nlon*
4 : error in the specification of *nt* (*jer* only)

hydro

Function for computing the geopotential height using the hydrostatic equation.

Synopsis

```
function hydro(
    p    : float,
    tkv  : float,
    zsfc : float
)
```

Arguments

p
: pressure (mb) (any dimensionality, last dimension is *nlvl*)

tkv
: temperature (K) at each *p* (same dimensionality as *p*)

zsfc
: surface geopotential height (gpm) (same dimensions as *p*, minus the last dimension *nlvl*)

Description

Function for computing the geopotential height using the hydrostatic equation. The results are returned in an array with the same dimensions as *p* and *tkv*.

No missing values are allowed.

Example

```
begin
  p  =(/ 1008.,1000.,950.,900.,850.,800.,750.,700.,650.,600., \
          550.,500.,450.,400.,350.,300.,250.,200., \
          175.,150.,125.,100., 80., 70., 60., 50., \
           40., 30., 25., 20. /)
  t  =(/  29.3,28.1,23.5,20.9,18.4,15.9,13.1,10.1, 6.7, 3.1,   \
          -0.5,-4.5,-9.0,-14.8,-21.5,-29.7,-40.0,-52.4,   \
         -59.2,-66.5,-74.1,-78.5,-76.0,-71.6,-66.7,-61.3, \
         -56.3,-51.7,-50.7,-47.5 /)
```

```
  q  =(/  20.38,19.03,16.14,13.71,11.56,9.80,8.33,6.75,6.06,5.07, \
           3.88, 3.29, 2.39, 1.70,1.00,0.60,0.20,0.00,0.00, \
           0.00, 0.00, 0.00, 0.00,0.00,0.00,0.00,0.00,0.00, \
           0.00, 0.00 /)

  zsfc = 17.0
  q    =  q*0.001
  tkv  = (t+273.15)*(1.+q*0.61)

  zh = hydro (p,tkv,zsfc)
end
```

---

idsfft

This function is used to call the NCAR 3.2 Fortran function IDSFFT.

Synopsis

```
        function idsfft(
                dim0[*]:float,
                dim1[*]:float,
                values[*]:float,
                dimensions[2]:integer
        )
```

Arguments

dim0
: A single-dimension array of four or more coordinates that represent dimension 0 of output.

dim1
: A single-dimension array of four or more coordinates that represent dimension 1 of output, must be same dimension size of *dim0*.

values
: A single-dimension array of four or more values at coordinates (*dim0*,*dim1*)with the same dimension size as *dim0* and *dim1*.

dimensions
: A single-dimension array of size 2 containing the requested output dimensions for dimensions 0 and 1, respectively.

Description

idsfft performs smooth surface fitting when the projections of the data points in the X-Y plane are irregularly distributed in the plane. *dim0* and *dim1* are randomly spaced coordinates in the X-Y plane, and *values* contains the values at each X-Y coordinate pair. No two coordinates (*dim0*,*dim1*) can be equal to each other. If this condition occurs, **idsfft** prints an error message.

NOTE: This function no longer exits when an error occurs.

idsfft returns a float array sized by the values specified in the *dimensions* parameter. The range of coordinates are determined by taking the min and max values of *dim0* and *dim1*, and dividing the range by the size of the requested dimension. The *dim0* values define the coordinate range of the first dimension (dimension #0) of the output array, and the *dim1* values determine the coordinate range of the second dimension (dimension #1). The output variable returned contains the derived coordinate variables (See Coordinate variables description). Dimension 0 of the output is named "ncl0" and dimension 1 of the output is named "ncl1". For more information, see the NCAR Graphics Version 3.2 man page for IDSFFT.

---

igradsf, igradsg

Given an array whose gradient is a vector function, compute its scalar function via Spherepack.

Synopsis

```
procedure igradsf(
    gzx : float,
    gzy : float,
    z   : float
)

procedure igradsg(
    gzx : float,
    gzy : float
    z   : float
)
```

Arguments

gzx, gzy
: gradient arrays (input, two or more dimensions, last two dimensions must be *nlat* x *nlon* and input values must be in ascending latitude order).

z
: scalar array (output, same dimensions as *gzx, gzy*, values will be in ascending latitude order)

Description

igradsf and *igradsg* both compute a scalar function given gradient *gzx* and *gzy* and return it in the array *z*. *igradsf* operates on an equal (fixed) grid, and *igradsg* operates on a Gaussian grid.

Note: For the arrays whose last two dimensions are *nlat* x *nlon*, the rest of the dimensions (if any) are collectively referred to as *nt*. If the input/output arrays are just two dimensions, then *nt* can either be considered equal to 1 or nothing at all.

Arrays which have dimensions *nt* x *nlat* x *nlon* should **not** include the cyclic (wraparound) points when invoking the procedures and functions which use spherical harmonics (Spherepack).

For example, if an array *x* has dimensions *nlat* = 64 and *nlon* = 129, where the "129" represents the cyclic points, then the user should pass the data to the procedure/function via:

```
z = sample ( x([...],:,0:nlon-2) )  ; does not include cyclic points
```

Example

Error messages

If *jer* or *ker* is equal to:

1 : error in the specification of *nlat*
2 : error in the specification of *nlon*
4 : error in the specification of *nt* (*jer* only)

---

igradsF, igradsG

Given an array whose gradient is a vector function, compute its scalar function via Spherepack.

Synopsis

```
function igradsF(
    gzx : float,
    gzy : float
)

function igradsG(
```

```
        gzx : float,
        gzy : float
    )
```

Arguments

gzx, gzy

gradient arrays (input, two or more dimensions, last two dimensions must be *nlat* x *nlon* and input values must be in ascending latitude order).

Description

igradsF and *igradsG* both compute a scalar function given gradient *gzx* and *gzy* and return it as a float array with the same dimensions as *gzy, gzy* (values will be in ascending latitude order). *igradsF* operates on an equal (fixed) grid, and *igradsG* operates on a Gaussian grid.

Note: For the arrays whose last two dimensions are *nlat* x *nlon*, the rest of the dimensions (if any) are collectively referred to as *nt*. If the input/output arrays are just two dimensions, then *nt* can either be considered equal to 1 or nothing at all.

Arrays which have dimensions *nt* x *nlat* x *nlon* should **not** include the cyclic (wraparound) points when invoking the procedures and functions which use spherical harmonics (Spherepack).

For example, if an array *x* has dimensions *nlat* = 64 and *nlon* = 129, where the "129" represents the cyclic points, then the user should pass the data to the procedure/function via:

```
    z = sample ( x([...],:,0:nlon-2) )  ; does not include cyclic points
```

Error messages

If *jer* or *ker* is equal to:

1 : error in the specification of *nlat*
2 : error in the specification of *nlon*
4 : error in the specification of *nt* (*jer* only)

---

ilapsf, ilapsg

Invert the Laplacian via Spherepack.

Synopsis

```
procedure ilapsf(
    zlap   : float,
    zlmbda : float,
    z      : float
)

procedure ilapsg(
    zlap   : float,
    zlmbda : float,
    z      : float
)
```

Arguments

zlap

The Laplacian array to invert (input, two or more dimensions, last two dimensions must be *nlat* x *nlon* and input values must be in ascending latitude order).

zlmbda

If *zlap* is a two dimensional array then *zlmbda* may be a constant. If *zlap* has 3 or more dimensions then *zlmbda* must be an array with the same dimensions as *zlap* (minus the rightmost two dimensions). If *zlmbda* is identically zero, the poisson equation is solved. Otherwise, the Helmholtz equation is solved.

z

the inverted Laplacian array (output, same dimensions as *zlap*, values will be in ascending latitude order)

Description

ilapsf and *ilapsg* both invert the Laplacian *zlap* and return it in the array *z*. *ilapsf* operates on an equal (fixed) grid, and *ilapsg* operates on a Gaussian grid.

Note: For the arrays whose last two dimensions are *nlat* x *nlon*, the rest of the dimensions (if any) are collectively referred to as *nt*. If the input/output arrays are just two dimensions, then *nt* can either be considered equal to 1 or nothing at all.

Arrays which have dimensions *nt* x *nlat* x *nlon* should **not** include the cyclic (wraparound) points when invoking the procedures and functions which use spherical harmonics (Spherepack).

For example, if an array *x* has dimensions *nlat* = 64 and *nlon* = 129, where the "129" represents the cyclic points, then the user should pass the data to the procedure/function via:

```
z = sample ( x([...],:,0:nlon-2) )  ; does not include cyclic points
```

Error messages

If *jer* or *ker* is equal to:

1 : error in the specification of *nlat*
2 : error in the specification of *nlon*
4 : error in the specification of *nt* (*jer* only)

---

ilapsF, ilapsG

Invert the Laplacian via Spherepack.

Synopsis

```
function ilapsF(
    zlap   : float,
    zlmbda : float
)

function ilapsG(
    zlap   : float,
    zlmbda : float
)
```

Arguments

zlap
: the Laplacian array to invert (input, two or more dimensions, last two dimensions must be *nlat* x *nlon* and input values must be in ascending latitude order).

zlmbda
: If *zlap* is a two dimensional array then *zlmbda* may be a constant. If *zlap* has 3 or more dimensions then *zlmbda* must be an array with the same dimensions as *zlap* (minus the rightmost two dimensions). If *zlmbda* is identically zero, the poisson equation is solved. Otherwise, the Helmholtz equation is solved.

Description

ilapsF and *ilapsG* both invert the Laplacian *zlap* and return a float array with the same dimensions as *zlap* (values will be in ascending latitude order). *ilapsF* operates on an equal (fixed) grid, and *ilapsG* operates on a Gaussian grid.

Note: For the arrays whose last two dimensions are *nlat* x *nlon*, the rest of the dimensions (if any) are collectively referred to as *nt*. If the input/output arrays are just two dimensions, then *nt* can either be considered equal to 1 or nothing at all.

Arrays which have dimensions *nt* x *nlat* x *nlon* should **not** include the cyclic (wraparound) points when invoking the procedures and functions which use spherical harmonics (Spherepack).

For example, if an array *x* has dimensions *nlat* = 64 and *nlon* = 129, where the "129" represents the cyclic points, then the user should pass the data to the procedure/function via:

```
z = sample ( x([...],:,0:nlon-2) )  ; does not include cyclic points
```

Error messages

If *jer* or *ker* is equal to:

1 : error in the specification of *nlat*
2 : error in the specification of *nlon*
4 : error in the specification of *nt* (*jer* only)

---

ilapvf, ilapvg

Invert the vector Laplacian using Spherepack.

Synopsis

```
procedure ilapvf(
    ulap : float,
    vlap : float,
    u    : float,
    v    : float
)

procedure ilapvg(
    ulap : float,
    vlap : float,
```

```
    u    : float,
    v    : float
)
```

Arguments

ulap, vlap
: the vector Laplacian arrays to invert (input, two or more dimensions, last two dimensions must be *nlat* x *nlon* and input values must be in ascending latitude order).

u, v
: vector arrays (output, same dimensions as *ulap, vlap*, values will be in ascending latitude order)

Description

ilapvf and *ilapvg* both invert the Laplacian vector *ulap, vlap* and return the results in the arrays *u* and *v*. *ilapvf* operates on an equal (fixed) grid, and *ilapvg* operates on a Gaussian grid.

Note: For the arrays whose last two dimensions are *nlat* x *nlon*, the rest of the dimensions (if any) are collectively referred to as *nt*. If the input/output arrays are just two dimensions, then *nt* can either be considered equal to 1 or nothing at all.

Arrays which have dimensions *nt* x *nlat* x *nlon* should **not** include the cyclic (wraparound) points when invoking the procedures and functions which use spherical harmonics (Spherepack).

For example, if an array *x* has dimensions *nlat* = 64 and *nlon* = 129, where the "129" represents the cyclic points, then the user should pass the data to the procedure/function via:

```
z = sample ( x([...],:,0:nlon-2) )  ; does not include cyclic points
```

Error messages

If *jer* or *ker* is equal to:

1 : error in the specification of *nlat*
2 : error in the specification of *nlon*
4 : error in the specification of *nt* (*jer* only)

---

ind

Returns the indexes into the input where the input is True.

Synopsis

```
function ind(
        larray[*] : logical
)
```

Arguments

larray
A singly dimensioned logical array.

Description

ind returns the integer indexes where the input is True.

Example

The following demonstrates the usefulness of the *ind* function. Try running NCL and entering the following statements.

```
;
; Create sample array
;
        a = (/1,2,3,4,5,5,4,3,2,1,1,2,3,4,5/)
;
; Assign missing value
;
        a@_FillValue = 5
;
; Assigns 0 to locations that are missing values
;
        a(ind(ismissing(a))) = 0;

;
; Performs an expression on non-zero values
;
        a(ind(a.ne.0)) = -a(ind(a.ne.0)) * 2 + 1
```

int2p

Function for interpolating from one set of pressure levels to another set using linear or ln(p) interpolation.

Synopsis

```
function int2p(
    pin       : float,
    xin       : float,
    pout      : float,
    linlog[1] : integer
)
```

Arguments

pin
: input pressure levels in decreasing order (array of any dimensions, last dimension is *npin*)

xin
: data at input pressure levels (array of same dimensions as *pin*)

linlog
: linear interpolation in pressure (=1) or linear interpolation in ln(p) (!=1)

pout
: output pressure levels in decreasing order (array of same dimensions as *pin*, only last dimension is replaced by *npout*)

Description

Function for interpolating from one set of pressure levels to another set using linear or ln(p) interpolation. The results are returned in a float array the same dimensions as *pout*. Missing values are allowed, but they are ignored.

Example

```
begin
  pi =(/ 1000.,925.,850.,700.,600.,500.,400.,300.,250., \
         200.,150.,100.,70.,50.,30.,20.,10. /)

  po =(/ 1000.,950.,900.,850.,800.,750.,700.,600.,500., \
         425.,400.,300.,250.,200.,100.,85.,70.,50.,40.,\
          30.,25.,20.,15.,10. /)

  pi@_FillValue = -999.9
```

```
  xi   = pi
  nin  = dimsizes(pi)
  nout = dimsizes(po)
  linlog = 1

  pi(2) = pi@_FillValue
  pi(10) = pi@_FillValue

  xo = int2p (pi,xi,po,linlog)
  print(xo)
end
```

---

isatt

This function is used to check if variable attributes are defined in a variables before accessing them.

Synopsis

```
        function isatt(
                var,
                attnames:string
        )
```

Arguments

var
: var must be a variable or variable subsection of any type or dimension.

attnames
: attnames must be an array of strings but can have any dimension.

Description

For each element in the *attnames* parameter, **isatt** returns True if the element is an attribute of the variable and False if not. The output of **isatt** is a logical array with the same dimensions as the *attnames* parameter. If the parameter *var* is not a variable, then a single missing value is returned.

---

isbyte

Returns True if input is of type **byte**

Synopsis

```
function isbyte(
        arg
)
```

Arguments

arg
: An array of any type and dimensionality.

Description

If the input is of type **byte** then *isbyte* returns a single scalar **logical** True value.

---

ischar

Returns True if input is of type **char**

Synopsis

```
function ischar(
        arg
)
```

Arguments

arg
: An array of any type and dimensionality.

Description

If the input is of type **char** then *ischar* returns a single scalar **logical** True value.

---

iscoord

Checks to see if input string names are coordinate variables of the *var* parameter.

Synopsis

```
function iscoord(
        var,
        coord_names : string
)
```

Arguments

var
Must be a variable parameter of any type and dimensionality

coord_names
An array of strings of any dimensionality

Description

Returns True for every element of *coord_names* that is a coordinate variable of the *var* parameter. Returns False otherwise.

---

isdefined

Returns True for every elment of the input that is a defined identifier.

Synopsis

```
function isdefined(
        idn_names : string
)
```

Arguments

idn_names

An array of strings of any dimensionality

Description

For each element of the input *isdefined* returns True if the element is the name of a defined identifier.

---

isdim

This function is used to check if variable dimensions are defined in variables before accessing them.

Synopsis

```
function isdim(
        var,
        dimnames:string
)
```

Arguments

var
: var must be a variable or variable subsection of any type or dimension.

dimnames
: dimnames must be an array of strings but can have any dimension.

Description

For each element in the *dimnames* parameter, **isdim** returns True if the element is a dimension of the variable and False if not. The output of **isdim** is a logical array with the same dimensions as the *dimnames* parameter. If the parameter *var* is not a variable, then a single missing value is returned.

---

isdouble

Returns True if input is of type **double**

Synopsis

```
function isdouble(
        arg
)
```

Arguments

arg
: An array of any type and dimensionality.

Description

If the input is of type **double** then *isdouble* returns a single scalar **logical** True value.

---

isfile

Returns True if input is of type **file**

Synopsis

```
function isfile(
        arg
)
```

Arguments

arg
: An array of any type and dimensionality.

Description

If the input is of type **file** then *isfile* returns a single scalar **logical** True value.

---

isfilevar

This function is used to check if file variables are defined in a file before accessing them.

Synopsis

```
function isfilevar(
        thefile[1]:file,
        varnames:string
)
```

Arguments

thefile
: thefile is the file to check.

varnames
: varnames must be an array of strings but can have any dimension.

Description

For each element in the *varnames* parameter, **isfilevar** returns True if the element is a variable of the file referenced by *thefile* and False if not. The output of **isfilevar** is a logical array with the same dimensions as the *varnames* parameter. If the parameter *thefile* is not a valid file, then a single missing value is returned.

isfilevaratt

This function is used to check if file variable attributes are defined in a file variable before accessing them.

Synopsis

```
function isfilevaratt(
        thefile[1]:file,
        varname[1]:string
        attnames:string
)
```

Arguments

thefile
: thefile is the file to check.

varname
: varname must be single string variable name.

attnames
: attnames must be an array of strings, but can have any dimension.

Description

For each element in the *attnames* parameter, **isfilevaratt** returns True if the element is an attribute of the file variable *varname* in the file *thefile* and False if not. The output of **isfilevaratt** is a logical array with the same dimensions as the *attnames* parameter. If the parameter *thefile* is not a valid file or *varname* isn't a defined file variable, then a single missing value is returned.

---

isfilevarcoord

Returns True if a coordinate variable exist.

Synopsis

```
function isfilevarcoord(
        thefile [1] : file,
        varname [1] : string,
        coordname [1] : string
)
```

Arguments

thefile
: reference to a file

varname
: name of file variable

coordname
: name of coordinate variable

Description

If a given dimension has a coordinate variable this function will return True. If the variable or dimension name do not exist then this function returns a missing value. It returns False only when no coordinate variable exists for the variable.

isfilevardim

This function is used to check if file variable dimensions are defined in a file variable before accessing them.

Synopsis

```
function isfilevardim(
        thefile[1]:file,
        varname[1]:string
        dimnames:string
)
```

Arguments

thefile
: thefile is the file to check.

varname
: varname must be single string variable name.

dimnames
: dimnames must be an array of strings, but can have any dimension.

Description

For each element in the *dimnames* parameter, **isfilevardim** returns True if the element is a dimension of the file variable *varname* in the file *thefile* and False if not. The output of **isfilevardim** is a logical array with the same dimensions as the *dimnames* parameter. If the parameter *thefile* is not a valid file or *varname* isn't a defined file variable, then a single missing value is returned.

isfloat

Returns True if input is of type **float**

Synopsis

```
function isfloat(
        arg
)
```

Arguments

arg
: An array of any type and dimensionality.

Description

If the input is of type **float** then *isfloat* returns a single scalar **logical** True value.

isfunc

Returns True for every elment of the input that is a defined procedure.

Synopsis

```
function isfunc(
        func_names : string
)
```

Arguments

func_names
: An array of strings of any dimensionality

Description

For each element of the input *isfunc* returns True if the element is the name of a defined function.

isgraphic

Returns True if input is of type **graphic**

Synopsis

```
function isgraphic(
        arg
)
```

Arguments

arg
: An array of any type and dimensionality.

Description

If the input is of type **graphic** then *isgraphic* returns a single scalar **logical** True value.

isinteger

Returns True if input is of type **integer**

Synopsis

```
function isinteger(
        arg
)
```

Arguments

arg
: An array of any type and dimensionality.

Description

If the input is of type **integer** then *isinteger* returns a single scalar **logical** True value.

---

islogical

Returns True if input is of type **logical**

Synopsis

```
function islogical(
        arg
)
```

Arguments

arg
: An array of any type and dimensionality.

Description

If the input is of type **logical** then *islogical* returns a single scalar **logical** True value.

---

islong

Returns True if input is of type **long**

Synopsis

```
function islong(
        arg
)
```

Arguments

arg

An array of any type and dimensionality.

Description

If the input is of type **long** then *islong* returns a single scalar **logical** True value.

---

ismissing

This function returns an n-dimensional logical array where elements are True if the input contains a missing value (determined by the _FillValue attribute).

Synopsis

```
function ismissing (
        data
)
```

Arguments

data

Can be of any dimensionality and any type of NCL data value.

Description

The **ismissing** function returns an n-dimensional array where elements are True when the input passed to it contains a missing value. This is useful for filtering missing values.

---

isnumeric

Returns True if input is of type **numeric**

Synopsis

```
function isnumeric(
        arg
)
```

Arguments

arg
: An array of any type and dimensionality.

Description

If the input is of type **numeric** then *isnumeric* returns a single scalar **logical** True value. A **numeric** type can be a **double**, **float**, **long**, **integer**, **short**, or **byte**.

ispan

Integer span function.

Synopsis

```
function ispan(
        start[1]:integer,
        finish[1]:integer,
        stride[1]:integer
)
```

Arguments

start
: Single scalar integer value at which the span will start.

finish
: Single scalar integer value at which the span will end.

stride
: Positive single scalar integer which will be the spacing of the span.

Description

Computes a span of integer values beginning at *start* and ending at *finish*. Each element will be separated by the value of *stride*. The stride must be positive but *start* and *finish* can be any valid integers.

---

isproc

Returns True for every elment of the input that is a defined procedure.

Synopsis

```
function isproc(
        proc_names : string
)
```

Arguments

proc_names
    An array of strings of any dimensionality

Description

For each element of the input *isproc* returns True if the element is the name of a defined procedure.

---

isshort

Returns True if input is of type **short**

Synopsis

```
function isshort(
        arg
)
```

Arguments

arg
    An array of any type and dimensionality.

Description

If the input is of type **short** then *isshort* returns a single scalar **logical** True value.

isstring

Returns True if input is of type **string**

Synopsis

```
function isstring(
        arg
)
```

Arguments

arg
    An array of any type and dimensionality.

Description

If the input is of type **string** then *isstring* returns a single scalar **logical** True value.

isvar

This function is used to check if variables are defined before accessing them.

Synopsis

```
function isvar(
        varnames:string
)
```

Arguments

varnames

varnames must be an array of strings, but can have any dimension.

Description

For each element in the *varnames* parameter, **isvar** returns, True if the element is a defined variable and False if not. The output of **isvar** is a logical array with the same dimensions as the *varnames* parameter.

lapsf, lapsg

Given a scalar z, compute its Laplacian via Spherepack.

Synopsis

```
procedure lapsf(
    z    : float,
    zlap : float
)

procedure lapsg(
    z    : float,
    zlap : float
)
```

Arguments

z

scalar function array (input, two or more dimensions, last two dimensions must be *nlat* x *nlon* and input values must be in ascending latitude order).

zlap

Laplacian array (output, same dimensions as *z*, values will be in ascending latitude order)

Description

lapsf and *lapsg* both compute the Laplacian *z* and return it in the array *zlap*. *lapsf* operates on an equal (fixed) grid, and *lapsg* operates on a Gaussian grid.

Note: For the arrays whose last two dimensions are *nlat* x *nlon*, the rest of the dimensions (if any) are collectively referred to as *nt*. If the input/output arrays are just two dimensions, then *nt* can either be considered equal to 1 or nothing at all.

Arrays which have dimensions *nt* x *nlat* x *nlon* should **not** include the cyclic (wraparound) points when invoking the procedures and functions which use spherical harmonics (Spherepack).

For example, if an array *x* has dimensions *nlat* = 64 and *nlon* = 129, where the "129" represents the cyclic points, then the user should pass the data to the procedure/function via:

```
z = sample ( x([...],:,0:nlon-2) )  ; does not include cyclic points
```

Error messages

If *jer* or *ker* is equal to:

1 : error in the specification of *nlat*
2 : error in the specification of *nlon*
4 : error in the specification of *nt* (*jer* only)

---

lapsF, lapsG

Given a scalar z, compute its Laplacian via Spherepack.

Synopsis

```
function lapsF(
    z : float
)

function lapsG(
    z : float
)
```

Arguments

z

: scalar function array (input, two or more dimensions, last two dimensions must be *nlat* x *nlon* and values must be in ascending latitude order).

Description

Given scalar function *z*, *lapsF* and *lapsG* both compute the Laplacian and return a float array with the same dimensions as *z* (values will be in ascending latitude order). *lapsF* operates on an equal (fixed) grid, and *lapsG* operates on a Gaussian grid.

Note: For the arrays whose last two dimensions are *nlat* x *nlon*, the rest of the dimensions (if any) are collectively referred to as *nt*. If the input/output arrays are just two dimensions, then *nt* can either be considered equal to 1 or nothing at all.

Arrays which have dimensions *nt* x *nlat* x *nlon* should **not** include the cyclic (wraparound) points when invoking the procedures and functions which use spherical harmonics (Spherepack).

For example, if an array *x* has dimensions *nlat* = 64 and *nlon* = 129, where the "129" represents the cyclic points, then the user should pass the data to the procedure/function via:

```
z = sample ( x([...],:,0:nlon-2) )  ; does not include cyclic points
```

Error messages

If *jer* or *ker* is equal to:

1 : error in the specification of *nlat*
2 : error in the specification of *nlon*
4 : error in the specification of *nt* (*jer* only)

---

lapvf, lapvg

Given vector quantity (u,v), compute the vector Laplacian via Spherepack.

Synopsis

```
procedure lapvf(
    u    : float,
    v    : float,
    ulap : float,
    vlap : float
)

procedure lapvg(
    u    : float,
    v    : float,
    ulap : float,
    vlap : float
)
```

Arguments

u, v
: vector field arrays (input, two or more dimensions, last two dimensions must be *nlat* x *nlon* and input values must be in ascending latitude order)

ulap, vlap
: vector Laplacian arrays (output, same dimensions as *u, v*, values will be in ascending latitude order)

Description

Given vector field *u* and *v*, *lapvf* and *lapvg* both compute the Laplacian vector and return the results in the arrays *ulap* and *vlap*. *lapvf* operates on an equal (fixed) grid, and *lapvg* operates on a Gaussian grid.

Note: For the arrays whose last two dimensions are *nlat* x *nlon*, the rest of the dimensions (if any) are collectively referred to as *nt*. If the input/output arrays are just two dimensions, then *nt* can either be considered equal to 1 or nothing at all.

Arrays which have dimensions *nt* x *nlat* x *nlon* should **not** include the cyclic (wraparound) points when invoking the procedures and functions which use spherical harmonics (Spherepack).

For example, if an array *x* has dimensions *nlat* = 64 and *nlon* = 129, where the "129" represents the cyclic points, then the user should pass the data to the procedure/function via:

```
z = sample ( x([...],:,0:nlon-2) )  ; does not include cyclic points
```

Error messages

If *jer* or *ker* is equal to:

1 : error in the specification of *nlat*
2 : error in the specification of *nlon*
4 : error in the specification of *nt* (*jer* only)

---

lderuvf, lderuvg

Given vector function (u,v), compute the latitudinal derivatives (uy,vy) via Spherepack.

Synopsis

```
procedure lderuvf(
    u  : float,
    v  : float,
    uy : float,
    vy : float
)

procedure lderuvg(
    u  : float,
    v  : float,
    uy : float,
    vy : float
)
```

Arguments

u, v
: vector function arrays (input, two or more dimensions, last two dimensions must be *nlat* x *nlon* and input values must be in ascending latitude order)

uy, vy
: vector latitudinal derivative arrays (output, same dimensions as *u, v*, values will be in ascending latitude order)

Description

lderuvf and *lderuvg* both compute the latitudinal derivative vector arrays from the vector function *u* and *v* and return the results in the arrays *uy* and *vy*. *lderuvf* operates on an equal (fixed) grid, and *lderuvg* operates on a Gaussian grid.

Note: For the arrays whose last two dimensions are *nlat* x *nlon*, the rest of the dimensions (if any) are collectively referred to as *nt*. If the input/output arrays are just two dimensions, then *nt* can either be considered equal to 1 or nothing at all.

Arrays which have dimensions *nt* x *nlat* x *nlon* should **not** include the cyclic (wraparound) points when invoking the procedures and functions which use spherical harmonics (Spherepack).

For example, if an array *x* has dimensions *nlat* = 64 and *nlon* = 129, where the "129" represents the cyclic points, then the user should pass the data to the procedure/function via:

```
z = sample ( x([...],:,0:nlon-2) )  ; does not include cyclic points
```

Error messages

If *jer* or *ker* is equal to:

1 : error in the specification of *nlat*
2 : error in the specification of *nlon*
4 : error in the specification of *nt* (*jer* only)

---

list_files

This procedure lists all of the variables that reference files.

Synopsis

```
procedure list_files( )
```

Description

Lists all of the current variables containing references to files. These variables were created by the addfile function. If run at the command line, this procedure invokes a pager for scrolling through the list. The pager is either "more" or whatever the user's PAGER environment variable is set to.

---

list_filevars

This procedure lists all of the variables associated with a specific file.

Synopsis

```
procedure list_filevars(
        filevar[1]:file
)
```

Arguments

filevar

A single reference to an open file. **Note** that NCAR Graphics Version 4.*x* does not support arrays of files.

Description

All of the variables referenced by the *filevar* parameter are listed. If run at the command line, this procedure invokes a pager for scrolling through the list. The pager is either "more" or whatever the user's PAGER environment variable is set to. Also see the getfilevarnames function.

---

list_hlus

This procedure is used to list all of the HLU objects currently referenced by NCL variables.

Synopsis

```
procedure list_hlus( )
```

Description

This procedure lists every variable reference to an HLU object currently in the NCL identifier list. If run at the command line, this procedure invokes a pager for scrolling through the list. The pager is either "more" or whatever the user's PAGER environment variable is set to.

---

list_procfuncs

This procedure is used to list all of the currently defined NCL functions and procedures and their parameter lists.

Synopsis

```
procedure list_procfuncs( )
```

Description

Lists all of the currently defined NCL procedures and functions, the type of each parameter, and the dimensionality of each parameter. If run at the command line, this procedure invokes a pager for scrolling through the list. The pager is either "more" or whatever the user's PAGER environment variable is set to.

---

list_vars

This procedure is used to list the currently defined variables that do not reference files or HLU objects.

Synopsis

```
procedure list_vars( )
```

Description

Lists all of the currently defined variables, their types, and their dimensionality. If run at the command line, this procedure invokes a pager for scrolling through the list. The pager is either "more" or whatever the user's PAGER environment variable is set to.

---

log

This function is used to compute the natural log of the input.

Synopsis

```
function log(
        value:numeric
```

```
)
```

Arguments

value
: An array of one or more values of any dimension. These values must be greater than 0.

Description

For each element in the input array, **log** returns the natural log. The return array is dimensioned the same as the input array. Missing values are ignored. The output datatype will be **double** if the input is type **double**.

log10

This function is used to compute the log base 10 of the input.

Synopsis

```
function log10(
        value:numeric
)
```

Arguments

value
: An array of one or more point values of any dimension. These values must be greater than 0.

Description

For each element in the input array, **log10** returns the log base 10. The return array is dimensioned the same as the input array. Missing values are ignored. Output will be of type **double** if input is of type **double**.

mask

Masks one multi-dimensional array against another given a single mask value.

Synopsis

```
function mask(
        array,
        marray,
        mvalue[1]
)
```

Arguments

array
: main array of any type and dimensionality but must be a super set of the dimensionality of *marray*

marray
: mask array of any type and dimensionality but must have at most the same dimensionality as *array* and its dimension sizes must correspond to *array* as defined below.

mvalue
: A single mask value which must be the same type as *marray* or coercible to that type

Description

mask masks the argument *array* against the mask array *marray* at locations where *marray* is equal to the argument *mvalue*. Missing values are placed in the output array in locations where *marray* is NOT equal to the mask value, *mvalue*.

If *array* has **n** dimensions and *marray* has **m** dimensions, then the dimensionality restrictions are as follows:

```
m is less-than-or-equal-to n
The dimension sizes of marray must be equal to
        dimensions n-m through n-1 of array
```

Example

The following demonstrates how *mask* works in addition to showing how to reshape *array* such that the dimensionality restrictions are met. Try running ncl and entering the following statements:

```
;
; Create array a and fill with integer values
;
        a = onedtond(ispan(1,120,1),(/4,5,6/))
        a!0 = "time"
        a!1 = "x"
        a!2 = "y"
;
; Create mask arrays:
; ma0 has the same dimension sizes as dimensions 1 and 2
; of a respectively.
; ma1 has the same dimension sizes as dimensions 2 and 0
; of a respectively.
;
        ma0 = onedtond( (/1,1,2,2,3,3/), (/5,6/))
        ma1 = onedtond( (/1,2,2,3/), (/6,4/))

;
; Simple case mask a at locations where ma0 is equal-to 2
;
        out0 = mask( a, ma0, 2)

;
; More advanced case mask a at locations where ma0 is not-equal-to 2
;
        out1 = mask( a, (ma0.ne.2), True)

;
; Now reshape a to mask a at locations where ma1 is equal-to 2
; using Named Subscripting
;
        out2 = mask( a( x | : , y | :,  time | :), ma1, 2)

;
; Same masking as above but it reshapes output of mask to
; original dimensionality
;
        out2!0 = "x"
        out2!1 = "y"
        out2!2 = "time"

        out2a = out2( time | :, x | : , y | :)
```

max

Returns the maximum value of a multi-dimensional array

Synopsis

```
        function max(
                arg:numeric
        )
```

Arguments

arg

A numeric array of any dimensionality

Description

This function returns the maximum value for an array of any dimensionality. *max* ignores missing values and will return the missing value only if all values are missing.

maxind

Returns the index of the first occurrence of a maximum value.

Synopsis

```
function maxind(
        arg [ * ] : numeric
)
```

Arguments

arg

A singly dimensioned numeric array

Description

Scans an array for the maximum value and returns the index of the first maximum value. *maxind* ignores missing values.

min

Returns the minimum value of a multi-dimensional array

Synopsis

```
function min(
        arg : numeric
)
```

Arguments

arg
: A numeric array of any dimensionality

Description

This function returns the minimum value for an array of any dimensionality. *min* ignores missing values and will return the missing value only if all values are missing.

minind

Returns the index of the first occurrence of a minimum value.

Synopsis

```
function minind(
        arg [ * ] : numeric
)
```

Arguments

arg
: A singly dimensioned numeric array

Description

Scans an array for the minimum value and returns the index of the first minimum value.

natgrids, natgridd

Routines that interpolate from 2D random data to a rectangular output grid. These routines are part of the Natgrid package which implements a natural neighbor interpolation method.

Synopsis

```
function natgrids(
    x[*]  : float,
    y[*]  : float,
    z[*]  : float,
    xo[*] : float,
    yo[*] : float
)

function natgridd(
    x[*]  : double,
    y[*]  : double,
    z[*]  : double,
    xo[*] : double,
    yo[*] : double
)
```

Arguments

x
: 1D array of any length (*npts*) containing the X coordinates of the input data points

y
: 1D array of length *npts* containing the Y coordinates of the input data points

z
: 1D array of length *npts* containing the functional values of the input data points (*z(i)* is the value of the input function at coordinate (*x(i)*, *y(i)*) for i=1,*npts*).

xo
: 1D array of any length (*NumXOut*) containing the X coordinates of the output data grid. The values in *xo* must be increasing, but need not be equally spaced.

yo
: 1D array of any length (*NumYOut*) containing the Y coordinates of the output data grid. The values in *yo* must be increasing, but need not be equally spaced.

Description

Both the `natgrids` and `natgridd` functions use a natural neighbor interpolation method for taking a set of randomly-spaced two-dimensional coordinates with function values at those coordinates and return a set of interpolated function values at coordinates in a user-specified rectangular grid. The results are returned in a 2D float array dimensioned *NumXOut* x *NumYOut*. `natgrids` is the single precision version and `natgridd` is the double precision version.

If an error code is returned, you can look up the meaning of the code in the Natgrid documentation at the URL:

http://ngwww.ucar.edu/ngdoc/ng/ngmath/natgrid/errors.html#ErrorTable

---

Example

```
begin
  NumXOut = 21
  NumYOut = 21
  ISLIM = 6
  x = new((/ISLIM/),double)
  y = new((/ISLIM/),double)
  z = new((/ISLIM/),double)
  xi = new((/NumXOut/),double)
  yi = new((/NumYOut/),double)

  x = (/0.00, 1.00, 0.00, 1.00, 0.40, 0.75/)
  y = (/0.00, 0.00, 1.00, 1.00, 0.20, 0.65/)
  z = (/0.00, 0.00, 0.00, 0.00, 1.25, 0.80/)

  xc = 1./(NumXOut-1.)
  xi = ispan(0,NumXOut-1,1) * xc

  yc = 1./(NumYOut-1.)
  yi = ispan(0,NumYOut-1,1) * yc

  zi = natgridd(x, y, z, xi, yi)
end
```

---

ncargpath

This function is used to retrieve absolute pathnames of various NCAR Graphics directories.

Synopsis

```
        function ncargpath(
                char:string
        )
```

Arguments

char

A string denoting the directory name whose full pathname you want. See the valid strings listed after the description.

Description

This function returns a fully qualified pathname for the requested directory. Below is a list of the valid strings. An invalid string returns NULL.

```
"bin"            Directory where NCAR Graphics binaries are
                 installed.

"config"         Directory where NCAR Graphics
                 configuration files are installed.

"data"           Directory where NCAR Graphics HLU and NCL
                 example data files are installed.

"database"       Directory where NCAR Graphics databases
                 (like the Ezmap database) are installed.

"doc"            Directory where NCAR Graphics PostScript
                 documents are installed.

"examples"       Directory where NCAR Graphics LLU Fortran
                 and C examples are installed.

"fontcap"        The default fontcap being used; only
                 returns a valid fontcap if the environment
                 variable FONTCAP is set.

"fontcaps"       Directory where NCAR Graphics fontcaps are
                 installed.

"gks_output"     The default name of the metafile.

"graphcap"       The default graphcap being used; only
                 returns a valid graphcap if the
                 environment variable GRAPHCAP is set.

"graphcaps"      Directory where NCAR Graphics graphcaps
                 are installed.

"hluex"          Directory where NCAR Graphics HLU Fortran
                 and C examples are installed.

"include"        Directory where NCAR Graphics include
                 files are installed.

"lib"            Directory where NCAR Graphics libraries
                 are installed.
```

"man"	Directory where NCAR Graphics man pages are installed.
"ncarg"	Root directory where NCAR Graphics examples, databases, resource files, etc. are installed.
"nclex"	Directory where NCAR Graphics NCL examples are installed.
"ngwww"	Directory where NCAR Graphics HTML files are installed (if they were installed).
"ngurl"	URL for the NCAR Graphics documentation.
"resfiles"	Directory where resource files for the NCL and HLU examples are installed.
"root"	Parent directory where NCAR Graphics is installed.
"sysappres"	Directory where the NCAR Graphics systems application resource file is installed.
"sysresfile"	Directory where the NCAR Graphics system resource file is installed.
"tests"	Directory where NCAR Graphics LLU Fortran and C test examples are installed.
"tmp"	Directory where NCAR Graphics temporary files will be written.
"tutorial"	Directory where NCAR Graphics LLU tutorial C and Fortran examples are installed.
"usrresfile"	Directory where the NCAR Graphics user resource file is installed.
"xapp"	Directory where NCAR Graphics X application default files are installed.

---

ncargversion

This procedure is used to change the output workstation of one or more HLU View instances.

Synopsis

```
procedure ncargversion(
)
```

Description

This procedure prints out the version number of NCAR Graphics being used.

---

ndtooned

Converts a multi-dimensional array to a singly dimensioned array.

Synopsis

```
function ndtooned(
        val
)
```

Arguments

val

An array of any dimensionality or type.

Description

This function converts any n-dimensional array to a singly dimensioned array that has a size equal to *product(dimsizes(input))*

Example

The following is a simplistic example. Try running the following command in NCL and print out the values.

```
a = (/(/1,2,3/),(/4,5,6/),(/7,8,9/)/)
b = ndtooned(a)
```

---

nngetaspects, nngetaspectd

Routines that retrieve an aspect at a specified coordinate value. These routines are part of the Natgrid package which implements a natural neighbor interpolation method.

Synopsis

```
function nngetaspects(
    i[1] : integer,
    j[1] : integer
)

function nngetaspectd(
    i[1] : integer,
    j[1] : integer
)
```

Arguments

i

a subscript indexing the first dimensioned variable in the output grid of the most recent call to `natgrids(d)`

j

a subscript indexing the second dimensioned variable in the output grid of the most recent call to `natgrids(d)`

Description

Both the `nngetaspects` and `nngetaspectd` functions return an aspect (as a scalar float value) at a specified coordinate value. `nngetaspects` is the single precision version and `nngetaspectd` is the double precision version.

Example

```
begin
;
; Define constants
;
  RAD2DEG = 57.29578
  NUMXOUT = 21
  NUMYOUT = 21
;
;  Coordinate data are defined as random numbers between
;  -0.2 and 1.2. and are explicitly defined here for uniformity
;  across platforms.
;
```

```
  x = (/ 1.16,  0.47,  0.29,  0.72,  0.52,  1.12,  0.33,  0.20,  0.30, \
 0.78,  0.92,  0.52,  0.44,  0.22, -0.10,  0.11,  0.59,  1.13, \
 0.68,  1.11,  0.93,  0.29,  0.74,  0.43,  0.87,  0.87, -0.10, \
 0.26,  0.85,  0.00, -0.02,  1.01, -0.12,  0.65,  0.39,  0.96, \
 0.39,  0.38,  0.94, -0.03, -0.17,  0.00,  0.03,  0.67, -0.06, \
 0.82, -0.03,  1.08,  0.37,  1.02, -0.11, -0.13,  1.03,  0.61, \
 0.26,  0.18,  0.62,  0.42,  1.03,  0.72,  0.97,  0.08,  1.18, \
 0.00,  0.69,  0.10,  0.80,  0.06,  0.82,  0.20,  0.46,  0.37, \
 1.16,  0.93,  1.09,  0.96,  1.00,  0.80,  0.01,  0.12,  1.01, \
 0.48,  0.79,  0.04,  0.42,  0.48, -0.18,  1.16,  0.85,  0.97, \
 0.14,  0.40,  0.78,  1.12,  1.19,  0.68,  0.65,  0.41,  0.90, \
 0.84, -0.11, -0.01, -0.02, -0.10,  1.04,  0.58,  0.61,  0.12, \
-0.02, -0.03,  0.27,  1.17,  1.02,  0.16, -0.17,  1.03,  0.13, \
 0.04, -0.03,  0.15,  0.00, -0.01,  0.91,  1.20,  0.54, -0.14, \
 1.03,  0.93,  0.42,  0.36, -0.10,  0.57,  0.22,  0.74,  1.15, \
 0.40,  0.82,  0.96,  1.09,  0.42,  1.13,  0.24,  0.51,  0.60, \
 0.06,  0.38,  0.15,  0.59,  0.76,  1.16,  0.02,  0.86,  1.14, \
 0.37,  0.38,  0.26,  0.26,  0.07,  0.87,  0.90,  0.83,  0.09, \
 0.03,  0.56, -0.19,  0.51,  1.07, -0.13,  0.99,  0.84,  0.22/)

  y = (/ -0.11,  1.07,  1.11, -0.17,  0.08,  0.09,  0.91,  0.17, -0.02, \
  0.83,  1.08,  0.87,  0.46,  0.66,  0.50, -0.14,  0.78,  1.08, \
  0.65,  0.00,  1.03,  0.06,  0.69, -0.16,  0.02,  0.59,  0.19, \
  0.54,  0.68,  0.95,  0.30,  0.77,  0.94,  0.76,  0.56,  0.12, \
  0.05, -0.07,  1.01,  0.61,  1.04, -0.07,  0.46,  1.07,  0.87, \
  0.11,  0.63,  0.06,  0.53,  0.95,  0.78,  0.48,  0.45,  0.77, \
  0.78,  0.29,  0.38,  0.85, -0.10,  1.17,  0.35,  1.14, -0.04, \
  0.34, -0.18,  0.78,  0.17,  0.63,  0.88, -0.12,  0.58, -0.12, \
  1.00,  0.99,  0.45,  0.86, -0.15,  0.97,  0.99,  0.90,  0.42, \
  0.61,  0.74,  0.41,  0.44,  1.08,  1.06,  1.18,  0.89,  0.74, \
  0.74, -0.06,  0.00,  0.99,  0.03,  1.00, -0.04,  0.24,  0.65, \
  0.12,  0.13, -0.09, -0.05,  1.03,  1.07, -0.02,  1.18,  0.19, \
  0.03, -0.03,  0.86,  1.12,  0.38,  0.72, -0.20, -0.08, -0.18, \
  0.32,  0.13, -0.19,  0.93,  0.81,  0.31,  1.09, -0.03,  1.01, \
 -0.17,  0.84, -0.11,  0.45,  0.18,  0.23,  0.81,  0.39,  1.09, \
 -0.05,  0.58,  0.53,  0.96,  0.43,  0.48,  0.96, -0.03,  1.13, \
  1.16,  0.16,  1.15,  0.57,  0.13,  0.71,  0.35,  1.04,  0.62, \
  1.03,  0.98,  0.31,  0.70,  0.97,  0.87,  1.14,  0.08,  1.19, \
  0.88,  1.00,  0.51,  0.03,  0.17,  1.01,  0.44,  0.17, -0.11/)

  z = (x-0.25)*(x-0.25) + (y-0.50)*(y-0.50)

  u = new((/NUMXOUT,NUMYOUT/),float)
  v = new((/NUMXOUT,NUMYOUT/),float)

  xc = 1./(NUMXOUT-1.)
  xo = ispan(0,NUMXOUT-1,1) * xc

  yc = 1./(NUMYOUT-1.)
  yo = ispan(0,NUMXOUT-1,1) * yc
;
;  Turn on gradient estimate calculations, flag calculation of
;  aspects and slopes, set flag to return aspects and slopes in
;  radians.
;
  nnsetp("SDI - compute slopes and aspects",1)
  nnsetp("IGR - use gradient estimates",1)
  nnsetp("RAD - return results in radians",1)
;
;  Do the interpolation.
;
```

```
  out = natgrids(x, y, z, xo, yo)
;
;  Get the aspects.
;
  do i=0,NUMXOUT-1
    do j=0,NUMYOUT-1
       rtmp = nngetaspects(i,j)
       u(i,j) = sin(rtmp)
       v(i,j) = cos(rtmp)
    end do
  end do

;
;  Get the slopes.
;
  do i = 0,NUMXOUT-1
    do j = 0,NUMYOUT-1
      rtmp = nngetslopes(i, j);
      u(i,j) = RAD2DEG*rtmp
    end do
  end do
end
```

---

nngetslopes, nngetsloped

Routines that retrieve a slope at a specified coordinate value. These routines are part of the Natgrid package which implements a natural neighbor interpolation method.

Synopsis

```
function nngetslopes(
    i[1] : integer,
    j[1] : integer
)

function nngetsloped(
    i[1] : integer,
    j[1] : integer
)
```

Arguments

i

a subscript indexing the first dimensioned variable in the output grid of the most recent call to `natgrids(d)`

j

a subscript indexing the second dimensioned variable in the output grid of the most recent call to `natgrids(d)`

Description

Both the `nngetslopes` and `nngetsloped` functions return a slope (as a scalar float value) at the grid point *z(i,j)* where *z* is the output grid in the most recent call to natgrids(d). `nngetslopes` is the single precision version and `nngetsloped` is the double precision version.

Example

```
begin
;
; Define constants
;
  RAD2DEG = 57.29578
  NUMXOUT = 21
  NUMYOUT = 21
;
;  Coordinate data are defined as random numbers between
;  -0.2 and 1.2. and are explicitly defined here for uniformity
;  across platforms.
;
  x = (/ 1.16,  0.47,  0.29,  0.72,  0.52,  1.12,  0.33,  0.20,  0.30, \
 0.78,  0.92,  0.52,  0.44,  0.22, -0.10,  0.11,  0.59,  1.13, \
 0.68,  1.11,  0.93,  0.29,  0.74,  0.43,  0.87,  0.87, -0.10, \
 0.26,  0.85,  0.00, -0.02,  1.01, -0.12,  0.65,  0.39,  0.96, \
 0.39,  0.38,  0.94, -0.03, -0.17,  0.00,  0.03,  0.67, -0.06, \
 0.82, -0.03,  1.08,  0.37,  1.02, -0.11, -0.13,  1.03,  0.61, \
 0.26,  0.18,  0.62,  0.42,  1.03,  0.72,  0.97,  0.08,  1.18, \
 0.00,  0.69,  0.10,  0.80,  0.06,  0.82,  0.20,  0.46,  0.37, \
 1.16,  0.93,  1.09,  0.96,  1.00,  0.80,  0.01,  0.12,  1.01, \
 0.48,  0.79,  0.04,  0.42,  0.48, -0.18,  1.16,  0.85,  0.97, \
 0.14,  0.40,  0.78,  1.12,  1.19,  0.68,  0.65,  0.41,  0.90, \
 0.84, -0.11, -0.01, -0.02, -0.10,  1.04,  0.58,  0.61,  0.12, \
-0.02, -0.03,  0.27,  1.17,  1.02,  0.16, -0.17,  1.03,  0.13, \
 0.04, -0.03,  0.15,  0.00, -0.01,  0.91,  1.20,  0.54, -0.14, \
 1.03,  0.93,  0.42,  0.36, -0.10,  0.57,  0.22,  0.74,  1.15, \
 0.40,  0.82,  0.96,  1.09,  0.42,  1.13,  0.24,  0.51,  0.60, \
 0.06,  0.38,  0.15,  0.59,  0.76,  1.16,  0.02,  0.86,  1.14, \
 0.37,  0.38,  0.26,  0.26,  0.07,  0.87,  0.90,  0.83,  0.09, \
 0.03,  0.56, -0.19,  0.51,  1.07, -0.13,  0.99,  0.84,  0.22/)

  y = (/ -0.11,  1.07,  1.11, -0.17,  0.08,  0.09,  0.91,  0.17, -0.02, \
  0.83,  1.08,  0.87,  0.46,  0.66,  0.50, -0.14,  0.78,  1.08, \
  0.65,  0.00,  1.03,  0.06,  0.69, -0.16,  0.02,  0.59,  0.19, \
  0.54,  0.68,  0.95,  0.30,  0.77,  0.94,  0.76,  0.56,  0.12, \
  0.05, -0.07,  1.01,  0.61,  1.04, -0.07,  0.46,  1.07,  0.87, \
  0.11,  0.63,  0.06,  0.53,  0.95,  0.78,  0.48,  0.45,  0.77, \
  0.78,  0.29,  0.38,  0.85, -0.10,  1.17,  0.35,  1.14, -0.04, \
  0.34, -0.18,  0.78,  0.17,  0.63,  0.88, -0.12,  0.58, -0.12, \
  1.00,  0.99,  0.45,  0.86, -0.15,  0.97,  0.99,  0.90,  0.42, \
```

```
   0.61,  0.74,  0.41,  0.44,  1.08,  1.06,  1.18,  0.89,  0.74, \
   0.74, -0.06,  0.00,  0.99,  0.03,  1.00, -0.04,  0.24,  0.65, \
   0.12,  0.13, -0.09, -0.05,  1.03,  1.07, -0.02,  1.18,  0.19, \
   0.03, -0.03,  0.86,  1.12,  0.38,  0.72, -0.20, -0.08, -0.18, \
   0.32,  0.13, -0.19,  0.93,  0.81,  0.31,  1.09, -0.03,  1.01, \
  -0.17,  0.84, -0.11,  0.45,  0.18,  0.23,  0.81,  0.39,  1.09, \
  -0.05,  0.58,  0.53,  0.96,  0.43,  0.48,  0.96, -0.03,  1.13, \
   1.16,  0.16,  1.15,  0.57,  0.13,  0.71,  0.35,  1.04,  0.62, \
   1.03,  0.98,  0.31,  0.70,  0.97,  0.87,  1.14,  0.08,  1.19, \
   0.88,  1.00,  0.51,  0.03,  0.17,  1.01,  0.44,  0.17, -0.11/)

  z = (x-0.25)*(x-0.25) + (y-0.50)*(y-0.50)

  u = new((/NUMXOUT,NUMYOUT/),float)
  v = new((/NUMXOUT,NUMYOUT/),float)

  xc = 1./(NUMXOUT-1.)
  xo = ispan(0,NUMXOUT-1,1) * xc

  yc = 1./(NUMYOUT-1.)
  yo = ispan(0,NUMXOUT-1,1) * yc
;
;  Turn on gradient estimate calculations, flag calculation of
;  aspects and slopes, set flag to return aspects and slopes in
;  radians.
;
  nnsetp("SDI - compute slopes and aspects",1)
  nnsetp("IGR - use gradient estimates",1)
  nnsetp("RAD - return results in radians",1)
;
;  Do the interpolation.
;
  out = natgrids(x, y, z, xo, yo)
;
;  Get the aspects.
;
  do i=0,NUMXOUT-1
    do j=0,NUMYOUT-1
       rtmp = nngetaspects(i,j)
       u(i,j) = sin(rtmp)
       v(i,j) = cos(rtmp)
    end do
  end do

;
;  Get the slopes.
;
  do i = 0,NUMXOUT-1
    do j = 0,NUMYOUT-1
      rtmp = nngetslopes(i, j);
      u(i,j) = RAD2DEG*rtmp
    end do
  end do
end
```

---

nnsetp, nngetp

Set and retrieve parameters for Natgrid routines. The ncl language affords a convenient implementation of the parameter setting and retrieval functions.

Synopsis

```
    procedure nnsetp(
        pnam[1] : string,
        pval[1]
    )

    function nngetp(
        pnam[1] : string,
    )
```

Arguments

pnam
: name of the parameter you want to set or retrieve

pval
: value of the parameter you want to set; this value must be of the type appropriate to the parameter being set.

Description

The procedure `nnsetp` is used to set values for natgrid parameters; the function `nngetp` is used to retrieve the current value of the named parameter. `nngetp` returns a value that is of the type appropriate to the parameter type.

Example

```
begin
;
; Define constants
;
  RAD2DEG = 57.29578
  NUMXOUT = 21
  NUMYOUT = 21
;
;  Coordinate data are defined as random numbers between
;  -0.2 and 1.2. and are explicitly defined here for uniformity
;  across platforms.
;
  x = (/ 1.16,  0.47,  0.29,  0.72,  0.52,  1.12,  0.33,  0.20,  0.30, \
```

```
 0.78,  0.92,  0.52,  0.44,  0.22, -0.10,  0.11,  0.59,  1.13, \
 0.68,  1.11,  0.93,  0.29,  0.74,  0.43,  0.87,  0.87, -0.10, \
 0.26,  0.85,  0.00, -0.02,  1.01, -0.12,  0.65,  0.39,  0.96, \
 0.39,  0.38,  0.94, -0.03, -0.17,  0.00,  0.03,  0.67, -0.06, \
 0.82, -0.03,  1.08,  0.37,  1.02, -0.11, -0.13,  1.03,  0.61, \
 0.26,  0.18,  0.62,  0.42,  1.03,  0.72,  0.97,  0.08,  1.18, \
 0.00,  0.69,  0.10,  0.80,  0.06,  0.82,  0.20,  0.46,  0.37, \
 1.16,  0.93,  1.09,  0.96,  1.00,  0.80,  0.01,  0.12,  1.01, \
 0.48,  0.79,  0.04,  0.42,  0.48, -0.18,  1.16,  0.85,  0.97, \
 0.14,  0.40,  0.78,  1.12,  1.19,  0.68,  0.65,  0.41,  0.90, \
 0.84, -0.11, -0.01, -0.02, -0.10,  1.04,  0.58,  0.61,  0.12, \
-0.02, -0.03,  0.27,  1.17,  1.02,  0.16, -0.17,  1.03,  0.13, \
 0.04, -0.03,  0.15,  0.00, -0.01,  0.91,  1.20,  0.54, -0.14, \
 1.03,  0.93,  0.42,  0.36, -0.10,  0.57,  0.22,  0.74,  1.15, \
 0.40,  0.82,  0.96,  1.09,  0.42,  1.13,  0.24,  0.51,  0.60, \
 0.06,  0.38,  0.15,  0.59,  0.76,  1.16,  0.02,  0.86,  1.14, \
 0.37,  0.38,  0.26,  0.26,  0.07,  0.87,  0.90,  0.83,  0.09, \
 0.03,  0.56, -0.19,  0.51,  1.07, -0.13,  0.99,  0.84,  0.22/)

  y = (/ -0.11,  1.07,  1.11, -0.17,  0.08,  0.09,  0.91,  0.17, -0.02, \
  0.83,  1.08,  0.87,  0.46,  0.66,  0.50, -0.14,  0.78,  1.08, \
  0.65,  0.00,  1.03,  0.06,  0.69, -0.16,  0.02,  0.59,  0.19, \
  0.54,  0.68,  0.95,  0.30,  0.77,  0.94,  0.76,  0.56,  0.12, \
  0.05, -0.07,  1.01,  0.61,  1.04, -0.07,  0.46,  1.07,  0.87, \
  0.11,  0.63,  0.06,  0.53,  0.95,  0.78,  0.48,  0.45,  0.77, \
  0.78,  0.29,  0.38,  0.85, -0.10,  1.17,  0.35,  1.14, -0.04, \
  0.34, -0.18,  0.78,  0.17,  0.63,  0.88, -0.12,  0.58, -0.12, \
  1.00,  0.99,  0.45,  0.86, -0.15,  0.97,  0.99,  0.90,  0.42, \
  0.61,  0.74,  0.41,  0.44,  1.08,  1.06,  1.18,  0.89,  0.74, \
  0.74, -0.06,  0.00,  0.99,  0.03,  1.00, -0.04,  0.24,  0.65, \
  0.12,  0.13, -0.09, -0.05,  1.03,  1.07, -0.02,  1.18,  0.19, \
  0.03, -0.03,  0.86,  1.12,  0.38,  0.72, -0.20, -0.08, -0.18, \
  0.32,  0.13, -0.19,  0.93,  0.81,  0.31,  1.09, -0.03,  1.01, \
 -0.17,  0.84, -0.11,  0.45,  0.18,  0.23,  0.81,  0.39,  1.09, \
 -0.05,  0.58,  0.53,  0.96,  0.43,  0.48,  0.96, -0.03,  1.13, \
  1.16,  0.16,  1.15,  0.57,  0.13,  0.71,  0.35,  1.04,  0.62, \
  1.03,  0.98,  0.31,  0.70,  0.97,  0.87,  1.14,  0.08,  1.19, \
  0.88,  1.00,  0.51,  0.03,  0.17,  1.01,  0.44,  0.17, -0.11/)

  z = (x-0.25)*(x-0.25) + (y-0.50)*(y-0.50)

  u = new((/NUMXOUT,NUMYOUT/),float)
  v = new((/NUMXOUT,NUMYOUT/),float)

  xc = 1./(NUMXOUT-1.)
  xo = ispan(0,NUMXOUT-1,1) * xc

  yc = 1./(NUMYOUT-1.)
  yo = ispan(0,NUMXOUT-1,1) * yc
;
;  Turn on gradient estimate calculations, flag calculation of
;  aspects and slopes, set flag to return aspects and slopes in
;  radians.
;
  nnsetp("igr - calculate gradients", 1)
  nnsetp("bI - adjust tautness parameter", 1.5)
;
;  Do the interpolation.
;
  out = natgrids(x, y, z, xo, yo)
;
```

```
; Retrieve current parameter settings and print them.
;
  igrval = nngetp("igr")
  bival = nngetp("bI")
  print(igrval)
  print(bival)
end
```

---

nnpnts, nnpntd

Routines that interpolate at a single point. Before calling these routines, nnpntinits(d) must be called. These routines are part of the Natgrid package which implements a natural neighbor interpolation method.

Synopsis

```
    function nnpnts(
        x[1] : float,
        y[1] : float,
    )

    function nnpntd(
        x[1] : double,
        y[1] : double,
    )
```

Arguments

x
: X coordinate of the point where interpolation is desired

y
: Y coordinate of the point where interpolation is desired

Description

Both the `nnpnts` and `nnpntd` functions interpolate at a specified point; each returns the interpolated function value at the input point. Before calling these procedures, `nnpntinits(d)` must be called.

Example

```
begin
  NUMXOUT = 21
  NUMYOUT = 21
  x = (/ 1.16,  0.47,  0.29,  0.72,  0.52,  1.12,  0.33,  0.20,  0.30, \
 0.78,  0.92,  0.52,  0.44,  0.22, -0.10,  0.11,  0.59,  1.13, \
 0.68,  1.11,  0.93,  0.29,  0.74,  0.43,  0.87,  0.87, -0.10, \
 0.26,  0.85,  0.00, -0.02,  1.01, -0.12,  0.65,  0.39,  0.96, \
 0.39,  0.38,  0.94, -0.03, -0.17,  0.00,  0.03,  0.67, -0.06, \
 0.82, -0.03,  1.08,  0.37,  1.02, -0.11, -0.13,  1.03,  0.61, \
 0.26,  0.18,  0.62,  0.42,  1.03,  0.72,  0.97,  0.08,  1.18, \
 0.00,  0.69,  0.10,  0.80,  0.06,  0.82,  0.20,  0.46,  0.37, \
 1.16,  0.93,  1.09,  0.96,  1.00,  0.80,  0.01,  0.12,  1.01, \
 0.48,  0.79,  0.04,  0.42,  0.48, -0.18,  1.16,  0.85,  0.97, \
 0.14,  0.40,  0.78,  1.12,  1.19,  0.68,  0.65,  0.41,  0.90, \
 0.84, -0.11, -0.01, -0.02, -0.10,  1.04,  0.58,  0.61,  0.12, \
-0.02, -0.03,  0.27,  1.17,  1.02,  0.16, -0.17,  1.03,  0.13, \
 0.04, -0.03,  0.15,  0.00, -0.01,  0.91,  1.20,  0.54, -0.14, \
 1.03,  0.93,  0.42,  0.36, -0.10,  0.57,  0.22,  0.74,  1.15, \
 0.40,  0.82,  0.96,  1.09,  0.42,  1.13,  0.24,  0.51,  0.60, \
 0.06,  0.38,  0.15,  0.59,  0.76,  1.16,  0.02,  0.86,  1.14, \
 0.37,  0.38,  0.26,  0.26,  0.07,  0.87,  0.90,  0.83,  0.09, \
 0.03,  0.56, -0.19,  0.51,  1.07, -0.13,  0.99,  0.84,  0.22/)

  y = (/ -0.11,  1.07,  1.11, -0.17,  0.08,  0.09,  0.91,  0.17, -0.02, \
  0.83,  1.08,  0.87,  0.46,  0.66,  0.50, -0.14,  0.78,  1.08, \
  0.65,  0.00,  1.03,  0.06,  0.69, -0.16,  0.02,  0.59,  0.19, \
  0.54,  0.68,  0.95,  0.30,  0.77,  0.94,  0.76,  0.56,  0.12, \
  0.05, -0.07,  1.01,  0.61,  1.04, -0.07,  0.46,  1.07,  0.87, \
  0.11,  0.63,  0.06,  0.53,  0.95,  0.78,  0.48,  0.45,  0.77, \
  0.78,  0.29,  0.38,  0.85, -0.10,  1.17,  0.35,  1.14, -0.04, \
  0.34, -0.18,  0.78,  0.17,  0.63,  0.88, -0.12,  0.58, -0.12, \
  1.00,  0.99,  0.45,  0.86, -0.15,  0.97,  0.99,  0.90,  0.42, \
  0.61,  0.74,  0.41,  0.44,  1.08,  1.06,  1.18,  0.89,  0.74, \
  0.74, -0.06,  0.00,  0.99,  0.03,  1.00, -0.04,  0.24,  0.65, \
  0.12,  0.13, -0.09, -0.05,  1.03,  1.07, -0.02,  1.18,  0.19, \
  0.03, -0.03,  0.86,  1.12,  0.38,  0.72, -0.20, -0.08, -0.18, \
  0.32,  0.13, -0.19,  0.93,  0.81,  0.31,  1.09, -0.03,  1.01, \
 -0.17,  0.84, -0.11,  0.45,  0.18,  0.23,  0.81,  0.39,  1.09, \
 -0.05,  0.58,  0.53,  0.96,  0.43,  0.48,  0.96, -0.03,  1.13, \
  1.16,  0.16,  1.15,  0.57,  0.13,  0.71,  0.35,  1.04,  0.62, \
  1.03,  0.98,  0.31,  0.70,  0.97,  0.87,  1.14,  0.08,  1.19, \
  0.88,  1.00,  0.51,  0.03,  0.17,  1.01,  0.44,  0.17, -0.11/)

  zo = new((/NUMXOUT,NUMYOUT/),float)

  z = (x-0.25)*(x-0.25) + (y-0.50)*(y-0.50)

  xc = 1./(NUMXOUT-1.)
  xo = ispan(0,NUMXOUT-1,1) * xc

  yc = 1./(NUMYOUT-1.)
  yo = ispan(0,NUMYOUT-1,1) * yc

  nnpntinits(x,y,z)
  do i=0,NUMXOUT-1
    do j=0,NUMYOUT-1
      zo(i,j) = nnpnts(xo(i),yo(j))
    end do
  end do
  nnpntend()
```

```
end
```

nnpntend, nnpntendd

Routines that terminate interpolation at single points. They are called after having made previous calls to `nnpntinits(d)` and `nnpnts(d)`. These routines are part of the Natgrid package which implements a natural neighbor interpolation method.

Synopsis

```
    procedure nnpntend()

    procedure nnpntendd()
```

Arguments

None.

Description

Both the `nnpntend` and `nnpntendd` procedures terminate interpolation at single points. Since these procedures don't take any input, both `nnpntend` and `nnpntendd` basically do the same thing. These procedures are called after `nnpntinits(d)` and `nnpnts(d)` are called to interpolate at individual points.

Example

```
begin
  NUMXOUT = 21
  NUMYOUT = 21
  x = (/ 1.16,  0.47,  0.29,  0.72,  0.52,  1.12,  0.33,  0.20,  0.30, \
 0.78,  0.92,  0.52,  0.44,  0.22, -0.10,  0.11,  0.59,  1.13, \
 0.68,  1.11,  0.93,  0.29,  0.74,  0.43,  0.87,  0.87, -0.10, \
 0.26,  0.85,  0.00, -0.02,  1.01, -0.12,  0.65,  0.39,  0.96, \
 0.39,  0.38,  0.94, -0.03, -0.17,  0.00,  0.03,  0.67, -0.06, \
 0.82, -0.03,  1.08,  0.37,  1.02, -0.11, -0.13,  1.03,  0.61, \
 0.26,  0.18,  0.62,  0.42,  1.03,  0.72,  0.97,  0.08,  1.18, \
 0.00,  0.69,  0.10,  0.80,  0.06,  0.82,  0.20,  0.46,  0.37, \
 1.16,  0.93,  1.09,  0.96,  1.00,  0.80,  0.01,  0.12,  1.01, \
```

```
 0.48,  0.79,  0.04,  0.42,  0.48, -0.18,  1.16,  0.85,  0.97, \
 0.14,  0.40,  0.78,  1.12,  1.19,  0.68,  0.65,  0.41,  0.90, \
 0.84, -0.11, -0.01, -0.02, -0.10,  1.04,  0.58,  0.61,  0.12, \
-0.02, -0.03,  0.27,  1.17,  1.02,  0.16, -0.17,  1.03,  0.13, \
 0.04, -0.03,  0.15,  0.00, -0.01,  0.91,  1.20,  0.54, -0.14, \
 1.03,  0.93,  0.42,  0.36, -0.10,  0.57,  0.22,  0.74,  1.15, \
 0.40,  0.82,  0.96,  1.09,  0.42,  1.13,  0.24,  0.51,  0.60, \
 0.06,  0.38,  0.15,  0.59,  0.76,  1.16,  0.02,  0.86,  1.14, \
 0.37,  0.38,  0.26,  0.26,  0.07,  0.87,  0.90,  0.83,  0.09, \
 0.03,  0.56, -0.19,  0.51,  1.07, -0.13,  0.99,  0.84,  0.22/)

  y = (/ -0.11,  1.07,  1.11, -0.17,  0.08,  0.09,  0.91,  0.17, -0.02, \
  0.83,  1.08,  0.87,  0.46,  0.66,  0.50, -0.14,  0.78,  1.08, \
  0.65,  0.00,  1.03,  0.06,  0.69, -0.16,  0.02,  0.59,  0.19, \
  0.54,  0.68,  0.95,  0.30,  0.77,  0.94,  0.76,  0.56,  0.12, \
  0.05, -0.07,  1.01,  0.61,  1.04, -0.07,  0.46,  1.07,  0.87, \
  0.11,  0.63,  0.06,  0.53,  0.95,  0.78,  0.48,  0.45,  0.77, \
  0.78,  0.29,  0.38,  0.85, -0.10,  1.17,  0.35,  1.14, -0.04, \
  0.34, -0.18,  0.78,  0.17,  0.63,  0.88, -0.12,  0.58, -0.12, \
  1.00,  0.99,  0.45,  0.86, -0.15,  0.97,  0.99,  0.90,  0.42, \
  0.61,  0.74,  0.41,  0.44,  1.08,  1.06,  1.18,  0.89,  0.74, \
  0.74, -0.06,  0.00,  0.99,  0.03,  1.00, -0.04,  0.24,  0.65, \
  0.12,  0.13, -0.09, -0.05,  1.03,  1.07, -0.02,  1.18,  0.19, \
  0.03, -0.03,  0.86,  1.12,  0.38,  0.72, -0.20, -0.08, -0.18, \
  0.32,  0.13, -0.19,  0.93,  0.81,  0.31,  1.09, -0.03,  1.01, \
 -0.17,  0.84, -0.11,  0.45,  0.18,  0.23,  0.81,  0.39,  1.09, \
 -0.05,  0.58,  0.53,  0.96,  0.43,  0.48,  0.96, -0.03,  1.13, \
  1.16,  0.16,  1.15,  0.57,  0.13,  0.71,  0.35,  1.04,  0.62, \
  1.03,  0.98,  0.31,  0.70,  0.97,  0.87,  1.14,  0.08,  1.19, \
  0.88,  1.00,  0.51,  0.03,  0.17,  1.01,  0.44,  0.17, -0.11/)

  zo = new((/NUMXOUT,NUMYOUT/),float)

  z = (x-0.25)*(x-0.25) + (y-0.50)*(y-0.50)

  xc = 1./(NUMXOUT-1.)
  xo = ispan(0,NUMXOUT-1,1) * xc

  yc = 1./(NUMYOUT-1.)
  yo = ispan(0,NUMYOUT-1,1) * yc

  nnpntinits(x,y,z)
  do i=0,NUMXOUT-1
    do j=0,NUMYOUT-1
      zo(i,j) = nnpnts(xo(i),yo(j))
    end do
  end do
  nnpntend()
end
```

---

nnpntinits, nnpntinitd

Routines that calculate all natural neighbor relationships in an input data array and sets some internal parameters so that `nnpnts` can be called to interpolate at individual points. These routines are part of the Natgrid package which implements a natural neighbor interpolation method.

Synopsis

```
procedure nnpntinits(
    x[*] : float,
    y[*] : float,
    z[*] : float
)

procedure nnpntinitd(
    x[*] : double,
    y[*] : double,
    z[*] : double
)
```

Arguments

x
: X coordinates of the input data points (any length)

y
: Y coordinates of the input data points (same length as *x*)

z
: functional values of the input data points (same length as *x*)

Description

Both the `nnpntinits` and `nnpntinitd` procedures calculate all natural neighbor relationships in an input data array and set some internal parameters so that `nnpnts` can be called to interpolate at individual points. `nnpntinits` operates on single precision data and `nnpntinitd` operates on double precision data.

Example

```
begin
  NUMXOUT = 21
  NUMYOUT = 21
  x = (/ 1.16,  0.47,  0.29,  0.72,  0.52,  1.12,  0.33,  0.20,  0.30, \
 0.78,  0.92,  0.52,  0.44,  0.22, -0.10,  0.11,  0.59,  1.13, \
 0.68,  1.11,  0.93,  0.29,  0.74,  0.43,  0.87,  0.87, -0.10, \
 0.26,  0.85,  0.00, -0.02,  1.01, -0.12,  0.65,  0.39,  0.96, \
 0.39,  0.38,  0.94, -0.03, -0.17,  0.00,  0.03,  0.67, -0.06, \
 0.82, -0.03,  1.08,  0.37,  1.02, -0.11, -0.13,  1.03,  0.61, \
 0.26,  0.18,  0.62,  0.42,  1.03,  0.72,  0.97,  0.08,  1.18, \
```

```
 0.00,  0.69,  0.10,  0.80,  0.06,  0.82,  0.20,  0.46,  0.37, \
 1.16,  0.93,  1.09,  0.96,  1.00,  0.80,  0.01,  0.12,  1.01, \
 0.48,  0.79,  0.04,  0.42,  0.48, -0.18,  1.16,  0.85,  0.97, \
 0.14,  0.40,  0.78,  1.12,  1.19,  0.68,  0.65,  0.41,  0.90, \
 0.84, -0.11, -0.01, -0.02, -0.10,  1.04,  0.58,  0.61,  0.12, \
-0.02, -0.03,  0.27,  1.17,  1.02,  0.16, -0.17,  1.03,  0.13, \
 0.04, -0.03,  0.15,  0.00, -0.01,  0.91,  1.20,  0.54, -0.14, \
 1.03,  0.93,  0.42,  0.36, -0.10,  0.57,  0.22,  0.74,  1.15, \
 0.40,  0.82,  0.96,  1.09,  0.42,  1.13,  0.24,  0.51,  0.60, \
 0.06,  0.38,  0.15,  0.59,  0.76,  1.16,  0.02,  0.86,  1.14, \
 0.37,  0.38,  0.26,  0.26,  0.07,  0.87,  0.90,  0.83,  0.09, \
 0.03,  0.56, -0.19,  0.51,  1.07, -0.13,  0.99,  0.84,  0.22/)

  y = (/ -0.11,  1.07,  1.11, -0.17,  0.08,  0.09,  0.91,  0.17, -0.02, \
  0.83,  1.08,  0.87,  0.46,  0.66,  0.50, -0.14,  0.78,  1.08, \
  0.65,  0.00,  1.03,  0.06,  0.69, -0.16,  0.02,  0.59,  0.19, \
  0.54,  0.68,  0.95,  0.30,  0.77,  0.94,  0.76,  0.56,  0.12, \
  0.05, -0.07,  1.01,  0.61,  1.04, -0.07,  0.46,  1.07,  0.87, \
  0.11,  0.63,  0.06,  0.53,  0.95,  0.78,  0.48,  0.45,  0.77, \
  0.78,  0.29,  0.38,  0.85, -0.10,  1.17,  0.35,  1.14, -0.04, \
  0.34, -0.18,  0.78,  0.17,  0.63,  0.88, -0.12,  0.58, -0.12, \
  1.00,  0.99,  0.45,  0.86, -0.15,  0.97,  0.99,  0.90,  0.42, \
  0.61,  0.74,  0.41,  0.44,  1.08,  1.06,  1.18,  0.89,  0.74, \
  0.74, -0.06,  0.00,  0.99,  0.03,  1.00, -0.04,  0.24,  0.65, \
  0.12,  0.13, -0.09, -0.05,  1.03,  1.07, -0.02,  1.18,  0.19, \
  0.03, -0.03,  0.86,  1.12,  0.38,  0.72, -0.20, -0.08, -0.18, \
  0.32,  0.13, -0.19,  0.93,  0.81,  0.31,  1.09, -0.03,  1.01, \
 -0.17,  0.84, -0.11,  0.45,  0.18,  0.23,  0.81,  0.39,  1.09, \
 -0.05,  0.58,  0.53,  0.96,  0.43,  0.48,  0.96, -0.03,  1.13, \
  1.16,  0.16,  1.15,  0.57,  0.13,  0.71,  0.35,  1.04,  0.62, \
  1.03,  0.98,  0.31,  0.70,  0.97,  0.87,  1.14,  0.08,  1.19, \
  0.88,  1.00,  0.51,  0.03,  0.17,  1.01,  0.44,  0.17, -0.11/)

  zo = new((/NUMXOUT,NUMYOUT/),float)

  z = (x-0.25)*(x-0.25) + (y-0.50)*(y-0.50)

  xc = 1./(NUMXOUT-1.)
  xo = ispan(0,NUMXOUT-1,1) * xc

  yc = 1./(NUMYOUT-1.)
  yo = ispan(0,NUMYOUT-1,1) * yc

  nnpntinits(x,y,z)
  do i=0,NUMXOUT-1
    do j=0,NUMYOUT-1
      zo(i,j) = nnpnts(xo(i),yo(j))
    end do
  end do
  nnpntend()
end
```

num

Counts the number of True values in input

Synopsis

```
function num(
        val : logical
)
```

Arguments

val

An array of logical values of any dimensionality

Description

This function counts the number of True values in the input and ignores missing values.

Example

The following is a simplistic example. Try executing NCL with the following commands.

```
a = (/1,2,3,4,5/)
print(num(a.gt.3))
```

---

onedtond

Used to redimension singly dimensioned arrays as well as fill multi-dimensional arrays with singly dimensioned data.

Synopsis

```
function onedtond(
        arg [*] ,
        dims : integer
)
```

Arguments

arg

A singly dimensioned array of any length and type

dims

A singly dimensioned integer array of positive values that represents the desired output dimensionality.

Description

Converts any 1-dimensional array to a multidimensional array. The first parameter is the 1-d array and the second parameter is an array of output dimension sizes. If the product of the output dimension sizes is less than the number of elements in the input array a warning is printed but the output array is filled with as many values from the input as will fit. If the product of the output dimension sizes is greater than the input size then the input is repeatedly copied as many times as will fit. if the fit is uneven an error message is printed.

Example

The following simplistic examples demonstrate all of the possible cases for *onedtond*. Try running these examples and print out the values.

```
        a = (/1,2,3,4,5,6,7,8/)
;
; Basic case where input and output have same number of elements
;
        a0 = onedtond(a,(/2,4/))
;
; Case where there are twice the number of elements in the output
; as the input. In this case the values of a are copied twice
;
        a1 = onedtond(a,(/2,8/))

;
; Case where there are half the number of elements in the output
; as the input. In this case only the first four values of a
; are copied to the output.
; The following case generates a WARNING.
;
        a2 = onedtond(a,(/2,2/))

;
; Case where there are more elements in output than input but
; number_output % number_input is not 0
; The following case generates a WARNING.
;
        a3 = onedtond(a,(/2,7/))
```

overlay

Used to overlay one HLU plot object over a specific style of transformation.

Synopsis

```
procedure overlay (
        base[1] : graphic,
        plot[1] : graphic
)
```

Arguments

base

The bottom element of the overlay chain of plots. The object referenced must support overlays.

plot

Graphical object to be added to the top of the overlay.

Description

Overlay is used to set a plot so it will be drawn over one or more other plots. When a plot is added to an overlay, the new plot will be drawn on top of all of the plots currently in the overlay list. **overlay** calls the HLU function NhlAddOverlay.

print

Prints a value of a variable or expression.

Synopsis

```
procedure print(
        data
)
```

Arguments

data

Any type of data array of any number of dimensions.

Description

The **print** procedure displays either the variable information and value of the data parameter--when it is a variable--or simply displays the values when the parameter is an expression result or literal value. If run at the command line, this procedure invokes a pager for scrolling through the list. The pager is either "more" or whatever the user's PAGER environment variable is set to.

product

Computes the product of the input.

Synopsis

```
function product(
        x : numeric
)
```

Arguments

x

An array of one or more numeric values.

Description

Returns the product of the input regardless of dimensionality. *product* ignores missing values.

prompt

This function is used ...

Synopsis

```
function FUNCNAME(
        ARGUMENTS
)
```

Arguments

ARG1
: ARG1 ...

ARG2
: ARG2 ...

Description

The **FUNCNAME** function ....

qsort

This procedure is used to sort single-dimension arrays of numbers.

Synopsis

```
procedure qsort(
        value[*]:numeric
)
```

Arguments

value
: A single-dimension array of two or more numeric values to be sorted.

Description

The **qsort** procedure sorts its input array *value*. If the input array contains a coordinate variable, then the values of the coordinate variable are repositioned relative to their corresponding array value. Missing values are sorted to either end of the array based on their actual value. The **qsort** algorithm does not support ignoring missing values. See sqsort for sorting arrays of strings.

rand

This function is used to generate pseudo random numbers.

Synopsis

```
        function rand( )
```

Description

The **rand** function returns a pseudo random number. The **srand** procedure can be used to establish a seed for the random number generator. For more information, see the **rand** and **srand** man pages. The **rand** returns an integer value between 0 and 32766 inclusive.

regcoef

Calculates the linear regression coefficient (trend, slope,...) between two series. Missing data are allowed.

Synopsis

```
    function regcoef(
        x     : float,
        y     : float,
        tval  : float,
        nptxy : integer
    )
```

Arguments

x

Input vector array of any dimensionality (last dimension is *npts*). Missing values should be indicated by *x*@_FillValue. If *x*@_FillValue is not set, then the NCL default (-999) will be assumed.

y

Input vector array of any dimensionality. The right-most dimensions of *y* must the same as the dimensions of *x*. Missing values should be indicated by *y*@_FillValue. If *y*@_FillValue is not set, then the NCL default (-999) will be assumed.

tval
: t-statistic (assuming null hypothesis) array (output) of the same dimensionality as *y*, minus the last dimension, *npts*. Space for this must be explicity allocated by the user (see example below).

nptxy
: number of points used (output), same dimensionality as *tval*. Space for this must be explicity allocated by the user (see example below).

Description

`regcoef` computes the regression coefficient (trend, slope,...) and returns it as a float array, for which the space is automatically allocated by NCL.

Example

In the example below, the regression coefficient for the case with no missing data is 0.97, the number of points used (*nptxy*) was 18 which yields 16 degrees of freedom (=*nptxy*-2). A test of the null hypothesis (i.e., that the regression coefficient is zero) yields a t-statistic of 38.7, which is distributed as t(16). Obviously, the null hypothesis would be rejected.

```
begin

 x = (/ 1190.,1455.,1550.,1730.,1745.,1770. \
     ,  1900.,1920.,1960.,2295.,2335.,2490. \
     ,  2720.,2710.,2530.,2900.,2760.,3010. /)

 y = (/ 1115.,1425.,1515.,1795.,1715.,1710. \
     ,  1830.,1920.,1970.,2300.,2280.,2520. \
     ,  2630.,2740.,2390.,2800.,2630.,2970. /)

 x@_FillValue = -999.
 y@_FillValue = -999.

 tval  = 0.
 nptxy = 0
 rc = regcoef(x,y,tval,nptxy)

 print ("NO MSG DATA: rc="+rc+"  tval="+tval+"  nptxy="+nptxy)

 x(2) = x@_FillValue
 y(8) = y@_FillValue

 rc = regcoef(x,y,tval,nptxy)
 print ("MSG DATA: rc="+rc+"  tval="+tval+"  nptxy="+nptxy)

end
```

---

relhum

Calculates relative humidity given temperature, mixing ratio, and pressure.

Synopsis

```
    function relhum(
        t  : float,
        w  : float,
        p  : float
    )
```

Arguments

t

Temperature (K) (any dimensionality)

w

Mixing ratio (kg/kg) (same dimensions as *t*)

p

pressure (Pa) (same dimensions as *p*)

Description

`relhum` returns a float array (of the same dimensionality as its input arrays) where each value represents the relative humidity given the temperature, mixing ratio, and pressure. The input arrays can be of any dimensionality as long as they are all the same dimensionality as each other.

Example

```
begin
  p  =(/ 1008.,  \
         1000.,950.,900.,850.,800.,750.,700.,650.,600., \
               550.,500.,450.,400.,350.,300.,250.,200., \
               175.,150.,125.,100., 80., 70., 60., 50., \
                40., 30., 25., 20. /)
  t  =(/  29.3,  \
          28.1,23.5,20.9,18.4,15.9,13.1,10.1, 6.7, 3.1,   \
          -0.5,-4.5,-9.0,-14.8,-21.5,-29.7,-40.0,-52.4,   \
         -59.2,-66.5,-74.1,-78.5,-76.0,-71.6,-66.7,-61.3, \
         -56.3,-51.7,-50.7,-47.5 /)
```

```
  q  =(/  20.38, \
         19.03,16.14,13.71,11.56,9.80,8.33,6.75,6.06,5.07, \
          3.88, 3.29, 2.39, 1.70,1.00,0.60,0.20,0.00,0.00, \
          0.00, 0.00, 0.00, 0.00,0.00,0.00,0.00,0.00,0.00, \
          0.00, 0.00 /)

  nlvl = dimsizes(p)
  print (nlvl)

  zsfc = 17.0
  q    =  q*0.001
  tk   = t+273.15
  tkv  = tk*(1.+q*0.61)
  tcv  =  tkv-273.15

  zh   = hydro (p*100.,tkv,zsfc)

  rhccm= relhum (tk, q, p*100. )

  expp = 2.7182
  eps = 0.62197

  do n=0,nlvl-1
    ev = q(n)*p(n)/(eps+q(n))
    es = 6.112*expp^(17.67*tcv(n)/(243.5+tcv(n)))
    rh =  (ev/es)*100.       ; in [%]; avoid 0
    print ("HYDRO: "+n+" "+p(n)+" "+zh(n) \
                    +" "+q(n)+" "+tcv(n)\
                    +" rhccm="+rhccm(n) \
                    +" rh="   +rh)
  end do
end
```

---

runave

Calculate an unweighted running average. Missing values are allowed.

Note: This function is only available in version 4.1.1 of NCL. If your site is licensed for version 4.1, then you should receive version 4.1.1 automatically.

Synopsis

```
    function runave(
        x    : float,
        nave : float,
        kopt : integer
    )
```

Arguments

x

An array with one or more dimensions. The rightmost dimension will be the dimension on which the unweighted running average is performed. Missing values should be indicated by *x*@_FillValue. If *x*@_FillValue is not set, then the NCL default (-999) will be assumed.

nave

Number of points to be included in the running average.

kopt

End-point option (*kopt* = 0 is the most frequently used)

In the following, N=last point in the series:

```
kopt < 0 : utilize cyclic conditions
           e.g.,  nave=3
           x(0) = (x(N)  +x(0)+x(1))/nave
           x(N) = (x(N-1)+x(N)+x(0))/nave
           e.g.,  nave=4 at n=1 and n=N
           x(0) = (x(N-1)+x(N)  +x(0)+x(1))/nave
           x(N) = (x(N-2)+x(N-1)+x(N)+x(0))/nave

kopt = 0 : set unsmoothed beginning and end pts to x@_FillValue [most common]
           e.g.,  nave=3 , x(0) = (x@_FillValue and x(N) = (x@_FillValue
                         , x(1) = (x(0)+x(1)+x(2))/nave
           e.g.,  nave=4 , x(0),x(1),x(N-1) and x(N) = (x@_FillValue
                         , x(2) = (x(0)+x(1)+x(2)+x(3))/nave

kopt > 0 : utilize reflective (symmetric) conditions
           e.g.,  nave=3
           x(0) = (x(1)  +x(0)+x(1))/nave
           x(N) = (x(N-1)+x(N)+x(N-1))/nave
           e.g.,  nave=4
           x(0) = (x(2)  +x(1)  +x(0)+x(1))/nave
           x(N) = (x(N-2)+x(N-1)+x(N)+x(N-1))/nave
```

Description

`runave` returns a float array of the same dimensionality as *x* with the last dimension smoothed. The running average is a special case of a filter where all weights are the same. The filter is applied to the *i*-th time of the requested series as follows:

F(i) = SUM{UF(i-(nave/2)+j-1)}/nave from j=0,nave-1

where *F* is the filtered field, *UF* is the unfiltered field, and *nave* is the number of elements in the running average.

If the number of weights is even, the filter's center falls between series elements; in this case, the center is shifted one-half of a time increment towards the latter element.

Example 1

Let *x* be dimensioned *nlat* x *mlon* x *ktimes* (*nlat*=64, *mlon*=128, *ktimes*=1000). Perform a 3 point running average and use *kopt*=0. Return the smoothed value to the original *x* array:

```
x = runave (x,3,0)
```

Example 2

Let *x* be dimensioned *ntimes* x *nlat* x *mlon* with named dimensions "time" , "lat" , "lon". Then:

```
nave = 31
kopt = 0
y    = runave (x(lat|:,lon|:,time|:), nave, kopt)
```

y will be a 3-dimensional array of length *nlat* x *mlon* x *time*.

Example 3

Let *x* be dimensioned *ntimes* x *klev* x *nlat* x *mlon* with named dimensions. Perform a 5 point running average use the cyclic option in the longitude direction:

```
nave = 5
kopt = -1
x = runave (x,nave, kopt)  ; return the series in the original array
```

---

shaec, shagc

Procedures for computing spherical harmonic analyses via Spherepack.

Synopsis

```
procedure shaec(
    g : float,
    a : float,
    b : float
)

procedure shagc(
    g : float,
    a : float,
    b : float
)
```

Arguments

g

discrete function to be analyzed (input, array with two or more dimensions, last two dimensions must be *nlat* x *nlon* and values must be in ascending latitude order, but see note below)

a, b

spherical harmonic coefficients (input/output, same dimensions as *g*, except last two dimensions must be *nlat* x *nlat* and values must/will be in ascending latitude order, but see note below)

Description

> **Note: If you have version 4.1 of NCL, these procedures will not work with arrays greater than two dimensions. In order to use arrays with more than two dimensions, you need to run version 4.1.1 of NCL. If you are licensed for NCAR Graphics software version 4.1, then you should automatically receive version 4.1.1. Typing "ncl" on the command line will tell you which version you are running.**

shaec and *shagc* both perform the spherical harmonic analysis on the array *g* and store the results in the arrays *a* and *b*. *shaec* operates on an equal (fixed) grid, and *shagc* operates on a Gaussian grid.

For the arrays whose last two dimensions are *nlat* x *nlon*, the rest of the dimensions (if any) are collectively referred to as *nt*. If the input/output arrays are just two dimensions, then *nt* can either be considered equal to 1 or nothing at all.

Arrays which have dimensions *nt* x *nlat* x *nlon* should **not** include the cyclic (wraparound) points when invoking the procedures and functions which use spherical harmonics (Spherepack).

For example, if an array *x* has dimensions *nlat* = 64 and *nlon* = 129, where the "129" represents the cyclic points, then the user should pass the data to the procedure/function via:

```
z = sample ( x([...],:,0:nlon-2) )  ; does not include cyclic points
```

Error messages

If *jer* or *ker* is equal to:

1 : error in the specification of *nlat*
2 : error in the specification of *nlon*
4 : error in the specification of *nt* (*jer* only)

shsec, shsgc

Procedures for computing spherical harmonic syntheses via Spherepack.

Synopsis

```
procedure shsec(
    a : float,
    b : float,
    g : float
)

procedure shsgc(
    a : float,
    b : float,
    g : float
)
```

Arguments

a, b

spherical harmonic coefficients (input, array with two or more dimensions, last two dimensions must be *nlat* x *nlat* and values must be in ascending latitude order, but see note below)

g

spherical harmonic synthesis of *a* and *b* (input/output, same dimensions as *a, b*, except last two dimensions must be *nlat* x *nlon* and values must/will be in ascending latitude order, but see note below)

Description

> **Note: If you have version 4.1 of NCL, these procedures will not work with arrays greater than two dimensions. In order to use arrays with more than two dimensions, you need to run version 4.1.1 of NCL. If you are licensed for NCAR Graphics software version 4.1, then you should automatically receive version 4.1.1. Typing "ncl" on the command line will tell you which version you are running.**

shsec and *shsgc* both perform the spherical harmonic synthesis on the arrays *a* and *b* and store the results in the array *g*. *shsec* operates on an equal (fixed) grid, and *shsgc* operates on a Gaussian grid.

For the arrays whose last two dimensions are *nlat* x *nlon*, the rest of the dimensions (if any) are collectively referred to as *nt*. If the input/output arrays are just two dimensions, then *nt* can either be considered equal to 1 or nothing at all.

Arrays which have dimensions *nt* x *nlat* x *nlon* should **not** include the cyclic (wraparound) points when invoking the procedures and functions which use spherical harmonics (Spherepack).

For example, if an array *x* has dimensions *nlat* = 64 and *nlon* = 129, where the "129" represents the cyclic points, then the user should pass the data to the procedure/function via:

```
z = sample ( x([...],:,0:nlon-2) )  ; does not include cyclic points
```

Error messages

If *jer* or *ker* is equal to:

1 : error in the specification of *nlat*
2 : error in the specification of *nlon*
4 : error in the specification of *nt* (*jer* only)

---

sin

This function is computes the sine from radian input.

Synopsis

```
function sin(
        value:numeric
)
```

Arguments

value
An array of one or more values of any dimension, in radians.

Description

The **sin** returns a floating-point array with the same dimensions as the input parameter *value*. The sine is computed for each element of *value*. Missing values are ignored. If the input *value* is of type **double** then output is also of type **double**.

---

sindex_yrmo, snindex_yrmo

Given two series of year-month values, calculate the Southern Oscillation Index.

Synopsis

```
function sindex_yrmo(
    slpt[*][*] : float,
    slpd[*][*] : float,
    iprnt[1] : integer
)

function snindex_yrmo(
    slpt[*][*]      : float,
    slpd[*][*]      : float,
    iprnt[1]        : integer,
    soi_noise[*][*] : float
)
```

Arguments

slpt
: 2D array of monthly data from station/grid pt 1 (*num_years* x *num_months*)

slpd
: 2D array of monthly data from station/grid pt 2 (same dimensions as *slpt*)

iprnt
: print flag (0 means do not print)

soi_noise
: 2D array of noise index (output, same dimension as *slpt, slpd*). Space for this variable must be explicitly allocated by the user (see example below).

Description

Given two series of year-month values (eg: slp) calculate an "index" (eg: Southern Oscillation Index). The overall anomaly standard deviation is used to normalize the anomalies. Both functions return a 2D float array with the dimensions as *slpt* and *slpd*.

Both of these routines return the attributes *long_name*, *short_name*, and *units*.

Example

```
begin
  nmos    = 12                         ; number of months
  nyrstrt = 1880                       ; first year of data
  nyrlast = 1997                       ; last year with data
  nyrs    = (nyrlast-nyrstrt+1)        ; total number of years
  xmsg    = -999.9                     ; missing value
  ncol    = nmos+1                     ; number of columns
  iprnt   = 1                          ; print soi out

; ================================>  ; READ THE ASCII FILES
  filet = asciiread ("tahiti.slp",(/nyrs,ncol/), "float")
  filed = asciiread ("darwin.slp",(/nyrs,ncol/), "float")
                                     ; create vector/arrays
  yr   = filet(:,0 )                   ; vector containing the years
  slpt = filet(:,1:)                   ; tahiti slp
  slpd = filed(:,1:)                   ; darwin slp
  slpt@_FillValue = xmsg
  slpd@_FillValue = xmsg


  soi = sindex_yrmo (slpt,slpd,iprnt)
  print("soi@_FillValue=" + soi@_FillValue)
  print(soi@long_name)
  print(soi@short_name)
  print(soi@units)

  xoi_noise = new ( (/nyrs,nmos/), float  )
  xoi = snindex_yrmo(slpt,slpd,iprnt,xoi_noise)
  print(xoi@long_name)
  print(xoi@short_name)
  print(xoi@units)
end
```

---

sinh

This function computes the hyperbolic sine function from radian input.

Synopsis

```
        function sinh(
                value:numeric
        )
```

Arguments

value

An array of one or more values of any dimension, in radians.

Description

The **sinh** returns a floating-point array with the same dimensions as the input parameter *value*. The hyperbolic sine is computed for each element of *value*. Missing values are ignored. If input *value* is of type **double** the output will be of type **double**.

sizeof

This function returns the total size, in bytes, of its parameter.

Synopsis

```
function sizeof(
        data
)
```

Arguments

data

Any type of data array of any number of dimensions.

Description

The **sizeof** function returns an integer value that is the size of the data pointed to by the parameter *data*. *data* is not constrained to be a specific type, and it has no constraints on its dimensionality. This function does not return the size of string data correctly.

sleep

This procedure pauses execution of NCL scripts for a specified number of seconds.

Synopsis

```
procedure sleep(
        seconds[1]:integer
)
```

Arguments

seconds
: A single positive integer value representing the number of seconds to pause.

Description

sleep pauses for the number of seconds specified in the *seconds* parameter.

---

pslhyp, pslec, pslhor

Functions for computing sea level pressures.

Synopsis

```
function pslhyp(
    pres : float,
    z    : float,
    tv   : float
)

function pslec(
    t    : float,
    phis : float,
    ps   : float,
    pres : float
)

function pslhor(
    z    : float,
    t    : float,
    phis : float,
    ps   : float,
    pres : float,
    lats : float
)
```

Arguments

phis

surface geopotential (m^2/s^2). A 2-dimensional array dimensioned *nlat* x *mlon* where *nlat* is the number of latitudes and *mlon* is the number of longitudes. **Note**: if the utility "ccm2nc" was used to convert the original CCM History Tape file to netCDF, this field may have a time dimension of length one. In this case use "phis(0,:,:)" as the argument to the functions.

pres

pressure (Pascals). For *pslhyp* and *pslec*, an array (corresponding to the lowest vertical level) with at least 2 dimensions. The last 2 dimensions being *nlat* x *mlon* where *nlat* is the number of latitudes and *mlon* is the number of longitudes. It must be at least 3 dimensions for *pslhor* because it requires multilevel fields which start at the bottom of the atmosphere (ie, the lowest vertical level). The last two dimensions must be *nlat* x *mlon*. For *pslhyp*, missing values should be indicated by *pres*@_FillValue. If *pres*@_FillValue is not set, then the NCL default (-999) will be assumed.

ps

surface pressure (Pascals) (last two dimensions must be the same dimensions as *phis*). It must be the same dimensions as *pres* except for in *pslhor* it won't have a multilevel field.

t

temperature (K) (same dimensions as *pres*)

tv

virtual temperature (K) (same dimensions as *pres*)

z

geopotential height (same dimensions as *pres*)

lats

latitudes (degrees). A vector of length *nlat*.

Description

pslhyp computes the sea level pressure via the hypsometric equation and returns the results as a float array with the same dimensions as its input arrays.

pslec computes the sea level pressure using the ECMWF formulation and returns the results as a float array with the same dimensions as its input arrays.

pslhor computes the sea level pressure using the ECMWF formulation and Trenberth's horizontal correction and returns the results as a float array with the same dimensions as its input arrays *ps* and *phis*.

All three of these routines return the attributes *long_name*, *short_name*, and *units*.

Note: For the arrays whose last two dimensions are *nlat* x *nlon*, the rest of the dimensions (if any) are collectively referred to as *nt*. If the input/output arrays are just two dimensions, then *nt* can either be considered equal to 1 or nothing at all.

Examples

Example 1

PSL is required for many time steps. Let's assume that the dimensions are:

t(time, lev, lat, lon), etc. for 4D variables
phis(time,lat,lon), etc. for 3D variables

and that lev(0) is the lowest level, then

PSL = pslhyp (pres(:,0,:,:), z(:,0,:,:), tv(:,0,:,:)

or

PSL = pslec (t(:,0,:,:), phis, ps, pres(:,0,:,:))

or

PSL = pslhor (z,t,phis,ps,pres,lat)

will return PSL(time,nlat,mlon).

Example 2

```
begin
  nlat  =  64              ; T42 gaussian grid
  mlon  = 128

  fname   = "pslecT42.hsl"             ; sequential fortran file
  nrec    = fbinnumrec (fname)         ; number of records

  t       = fbinrecread(fname, 0 ,(/nlat,mlon/), "float" )
  phis    = fbinrecread(fname, 1 ,(/nlat,mlon/), "float" )
  ps      = fbinrecread(fname, 2 ,(/nlat,mlon/), "float" )
  pres    = fbinrecread(fname, 3 ,(/nlat,mlon/), "float" )
  psl     = pslec (t,phis,ps,pres)
  print(psl)
  print(psl@long_name)
  print(psl@short_name)
  print(psl@units)
end
```

---

specx_anal

Calculates spectra quantities of a series.

Note: This function is only available in version 4.1.1 of NCL. If your site is licensed for version 4.1, then you should receive version 4.1.1 automatically.

Synopsis

```
function specx_anal(
    x    : float,
    iopt : integer,
    jave : integer,
    pct  : float
)
```

Arguments

x

A 1-dimensional array of length *N* containing the data. Missing values are not allowed.

iopt

detrending option. iopt = 0 => remove series mean, iopt = 1 => remove the series mean and least squares linear trend.

jave

smoothing to be performed on the periodiogram estimates. This should be an odd number (>= 3). If not, the routine will force it to the next largest odd number.

```
jave=0 : do no smoothing
         spcx contains raw spectra estimates (periodogram)
jave>0 : average jave periodogram estimates together
         utilizing modified daniell smoothing (good
         stability but may lead to large bias ).
         all weights are 1/jave except wgt(1) and wgt(jave)
         which are 1/(2*jave). this is the recommended option.
It is this number which has the most impact on the
degrees of freedom.
```

pct

percent of the series to be tapered (0.0 <= pct <= 1.0). If *pct* =0.0, no tapering will be done. If *pct* = 1.0, the whole series is effected. A *pct* of 0.10 is common (tapering should always be done).

Description

`specx_anal` returns the degrees of freedom as a float scalar. It also returns the following attributes:

- *spcx* - 1-dimensional array of length N/2
  spcx(0) - spectral estimate at frq = (1/N) [N=dimsizes(x)]
  spcx(N/2-1)- spectral estimate at frq = 0.5

  These spectra have been normalized so that SUM{spcx(n)*df}= variance of the detrended series where df=frequency spacing. The units are variance/(unit frequency interval).

- *frq* - frequency (cycles/time). A 1-dimensional array of length *N*/2.
- *bw* - spectral band width (scalar, float)
- *xavei* - average of the x series on input (scalar, float)
- *xvari* - variance of the x series on input (scalar, float)
- *xvaro* - variance of the x series after detrending (scalar, float)
- *xlag1* - lag one auto correlation of the x series after detrending (scalar, float)
- *xslope* - least squares slope per time interval of linear trend (if *iopt* = 1) of the x series.

Example 1

Perform a spectral analysis on series *x*:

```
iopt = 0    ; remove series mean
jave = 0    ; no smoothing; return periodogram estimates
pct  = 0.1  ; taper 10% of the data
dof  = specx_anal (x,iopt,jave,pct)
```

Example 2

Perform a spectral analysis on series *x* but assume *x* has dimensions *nyrs* x *nmos* where *nyrs* represents the number of years and *nmos* the number of months. Use NCL function **ndtooned**:

```
iopt = 0    ; remove series mean
jave = 3    ; Average 3 periodogram estimates using modified Daniel
pct  = 0.1  ; taper 10% of the data
dof  = specx_anal (ndtooned(x),iopt,jave,pct)
```

Example 3

Perform a spectral analysis on series *x* but assume *x* has dimensions *time* x *lev* x *lat* x *lon* and named dimensions "time", "lev", "lat", "lon". Let’s assume we want the spectral analysis across time at a specific point lev=kl, lat=nl, lon=ml. Then, using NCL’s dimension reordering:

```
iopt = 1    ; remove least squares linear trend in x
            ; prior to tapering and computing spectra.
jave = 5    ; Average 5 periodogram estimates using modified Daniel
pct  = 0.1  ; taper 10% of the data
dof  = specx_anal (x(lev|kl:kl,lat|nl:nl,lon|ml:ml,time:),iopt,jave,pct)
```

---

specxy_anal

Calculates cross spectra quantities of a series.

Note: This function is only available in version 4.1.1 of NCL. If your site is licensed for version 4.1, then you should receive version 4.1.1 automatically.

Synopsis

```
function specxy_anal(
    x    : float,
    y    : float,
    iopt : integer,
    jave : integer,
    pct  : float
)
```

Arguments

x, y
: 1-dimensional arrays of length *N* containing the data. *x* and *y* must be the same length. Missing values are not allowed.

iopt
: detrending option. iopt = 0 => remove series mean, iopt = 1 => remove the series mean and least squares linear trend.

jave
: smoothing to be performed on the periodiogram estimates. This should be an odd number (>= 3). If not, the routine will force it to the next largest odd number.

```
jave=0 : do no smoothing
         spcx contains raw spectra estimates (periodogram)
jave>0 : average jave periodogram estimates together
         utilizing modified daniell smoothing (good
         stability but may lead to large bias ).
         all weights are 1/jave except wgt(1) and wgt(jave)
         which are 1/(2*jave). this is the recommended option.
It is this number which has the most impact on the
degrees of freedom.
```

pct

percent of the series to be tapered (0.0 <= pct <= 1.0). If *pct* =0.0, no tapering will be done. If *pct* = 1.0, the whole series is effected. A *pct* of 0.10 is common (tapering should always be done).

Description

`specxy_anal` returns the degrees of freedom as a float scalar. It also returns the following attributes:

- *spcx,spcy* - 1-dimensional arrays of length N/2
  spcx(0) - spectral estimate at frq = (1/N) [N=dimsizes(x)]
  spcx(N/2-1)- spectral estimate at frq = 0.5

  These spectra have been normalized so that SUM{spcx(n)*df}= variance of the detrended series where df=frequency spacing. The units are variance/(unit frequency interval).

- *frq* - frequency (cycles/time). A 1-dimensional array of length *N*/2.

- *cospc* - cospectrum. A 1-dimensional array of length *N*/2. This is the real part of the cross spectrum. It measures the extent to which there are oscillations with the same phase in the two series (or with opposite sign, i.e., with a phase shift of half a cycle). In other words, it measures the contribution of different frequencies to the total cross-covariance at zero lag.

- *quspc* - quadrature spectrum. A 1-dimensional array of length *N*/2. This is the imaginary part of the cross spectrum. it measures the extent to which there are oscillations with a phase difference of a quarter cycle in either direction. i.e., It measures the contribution of different frequencies to the total cross-covariance of the series when all harmonics of one series are delayed a quarter cycle relative to the other relative to the other series.

- *coher* - coherence squared. A 1-dimensional array of length *N*/2. This is analogous to the square of the correlation coef except that the coherence squared is a function of frequency.

- *phase* - phase (in degrees). A 1-dimensioinal array of length *N*/2. A positive phase indicates that *x* leads *y*.

- *bw* - spectral band width (scalar, float)

- *xavei* - average of the x series on input (scalar, float)

- *xvari* - variance of the x series on input (scalar, float)

- *xvaro* - variance of the x series after detrending (scalar, float)

- *xlag1* - lag one auto correlation of the x series after detrending (scalar, float)

- *xslope* - least squares slope per time interval of linear trend (if *iopt* = 1) of the x series.

- *yavei* - average of the y series on input (scalar, float)
- *yvari* - variance of the y series on input (scalar, float)
- *yvaro* - variance of the y series after detrending (scalar, float)
- *ylag1* - lag one auto correlation of the y series after detrending (scalar, float)
- *yslope* - least squares slope per time interval of linear trend (if *iopt* = 1) of the y series.

Example 1

Perform cross-spectral analysis on series *x* and *y*:

```
iopt = 1    ; remove least squares linear trends from each
            ; series prior to tapering and computing spectra.
jave = 7    ; Average 7 periodogram estimates using modified Daniel
pct  = 0.1  ; taper 10% of the data
dof = specxy_anal (x,y,iopt,jave,pct)
```

---

sqrt

This function computes the square root of its input.

Synopsis

```
function sqrt(
        value:numeric
)
```

Arguments

value
: An array of one or more positive values.

Description

For each element of *value*, the square root is returned. Missing values are ignored. The output array is dimensioned the same as the input array. If the input is of type **double** then output is of type **double**.

sqsort

This procedure is used to sort single-dimension arrays of strings.

Synopsis

```
procedure sqsort(
        value[*]:strings
)
```

Arguments

value
A single-dimension array of two or more string values to be sorted.

Description

The **sqsort** procedure sorts its input array *value*. If the input array contains a coordinate variable, then the values of the coordinate variable are repositioned relative to their corresponding array value. Missing values are sorted to either end of the array based on their actual value. The **sqsort** algorithm does not support ignoring missing values. See qsort for sorting arrays of numbers.

srand

This procedure establishes a seed for the **rand** function.

Synopsis

```
procedure srand(
        seed[1]:integer
)
```

Arguments

seed

A positive integer value to use as the seed for the **rand** function.

Description

The **srand** procedure sets a seed for the **rand** function to use when computing random numbers.

rdsstoi

Function for reading weekly/monthly compocp site, and climatology grids.

Synopsis

```
function rdsstoi(
    nyrstrt : integer,
    nyrlast : integer,
    mlon    : integer,
    nlat    : integer,
    info[9] : integer
)
```

Arguments

nyrstrt, nyrlast

Start and end years for selecting data

mlon, nlat

used for error checking

info

date and error information (info(8) has error info)

Description

This function reads weekly/monthly compocp site and climatology grids, and returns a float array dimensioned 180 x 360 (latitude x longitude).

Example

---

stat2, stat_trim, stat4, stat_medrng

For each index of dimensions 0...*n*-2, these procedures calculate estimates, variances, skewnesses, kurtoses, medians, ranges, and mid-ranges on moments of the *n*-1 dimension.

Synopsis

```
procedure stat2(
    x       : float,
    xmean   : float,
    xvar    : float,
    nptused : integer
)

procedure stat_trim(
    x      : float,
    ptrim  : float,
    xmeant : float,
    xsdt   : float
)

procedure stat4(
    x       : float,
    xmean   : float,
    xvar    : float,
    xskew   : float,
    xkurt   : float,
    nptused : integer
)

procedure stat_medrng(
    x       : float,
    xmedian : float,
    xmrange : float,
    xrange  : float,
    nptused : integer
)
```

Arguments

x

Input array of any dimensionality. Missing values should be indicated by *x*@_FillValue. If *x*@_FillValue is not set, then the NCL default (-999) will be assumed.

xmean

Mean of *x* (output, same dimensions as *x*, with last dimension omitted). Space for this variable must be explicitly allocated by the user.

xmeant
: Mean of trimmed *x* (output, same dimensions as *xmean*). Space for this variable must be explicitly allocated by the user.

xvar
: Sample variance of *x* (output, same dimensions as *xmean*). Space for this variable must be explicitly allocated by the user.

xsdt
: Sample standard deviation of trimmed *x* (output, same dimensions as *xmean*). Space for this variable must be explicitly allocated by the user.

xskew
: Coefficient of skewness (output, same dimensions as *xmean*). Space for this variable must be explicitly allocated by the user.

xkurt
: Coefficient of kurtosis (output, same dimensions as *xmean*). Space for this variable must be explicitly allocated by the user.

xmedian
: Median value of *x* (output, same dimensions as *xmean*). Space for this variable must be explicitly allocated by the user.

xmrange
: Mid-range value of *x* (output, same dimensions as *xmean*). Space for this variable must be explicitly allocated by the user.

xrange
: Range of *x* (output, same dimensions as *xmean*). Space for this variable must be explicitly allocated by the user.

nptused
: Number of points used (output, same dimensions as *xmean*). Space for this variable must be explicitly allocated by the user.

ptrim
: Scalar, portion of series to be trimmed (0. <= *ptrim* < 1.). If *ptrim* = 0, then calculate the whole series mean and standard deviation. Otherwise, a minimum of two points will be trimmed.

Description

Note for the procedures below: for dimensions 0 to *n*-2, the calculations are done on the last dimension (*n*-1th dimension) of the input array. The output arrays are the same dimensions as the first 0 to *n*-2 dimensions of the input (if the input is a singly-dimensioned array, then the output will be a scalar). Because the output values are part of the procedure calls, you must allocate the output arrays before the procedure is called. Some examples are shown below.

`stat2` calculates the first two moments of the n-1 dimension of *x* containing missing data. This function returns the sample variance.

`stat_trim` calculates trimmed estimates of the first two moments of the vectors in *x* containing missing data.

`stat4` calculates estimates of the first four moments (mean, variance, skewness, and kurtosis) of the vectors in *x* containing missing data. This function returns the sample variance.

`stat_medrng` calculates median, range, and mid-range of the vectors in *x* containing missing data.

Example

Run the following example and then print out `xmean`, `xvar`, `xskew`, `xkurt`, and `npts` to see how `stat2` and `stat4` work.

```
begin
;
; Define a 3 x 2 x 2 array
;
   x = (/(/(/1.,2./),(/3.,4./)/),(/(/5.,6./),(/-999,8./)/),\
         (/(/9.,10./),(/11.,-999/)/)/)
   x@_FillValue = -999

   xmean   = new((/3,2/),float)
   xvar    = new((/3,2/),float)
   xskew   = new((/3,2/),float)
   xkurt   = new((/3,2/),float)
   npts    = new((/3,2/),integer)

   stat2(x,xmean,xvar,npts)
   print(xmean)
   print(xvar)
   print(npts)

   stat4(x,xmean,xvar,xskew,xkurt,npts)
   print(xmean)
   print(xvar)
   print(xskew)
   print(xkurt)
   print(npts)
end
```

---

stddev

Computes standard deviation

Synopsis

```
        function stddev(
                data : numeric
        )
```

Arguments

data
: a numeric data array of any dimensionality

Description

Returns the standard deviation of the input data. Missing values (*data*@_FillValue) are ignored. To determine the number of data points used to calculate the standard deviation use:

```
N = num(.not.ismissing(data))
```

Technically, this function calculates the population standard deviation.

sum

Sums input.

Synopsis

```
function sum(
        x : numeric
)
```

Arguments

x
: An array of one or more numeric values.

Description

Returns the sum of the input regardless of dimensionality. *sum* ignores missing values.

svdcov, svdstd

Singular value decomposition functions

Synopsis

```
function svdcov(
    x[*][*]      : float,
    y[*][*]      : float,
    nsvd         : integer,
    homlft[*][*] : float,
    hetlft[*][*] : float,
    homrgt[*][*] : float,
    hetrgt[*][*] : float
)

function svdstd(
    x[*][*]      : float,
    y[*][*]      : float,
    nsvd         : integer,
    homlft[*][*] : float,
    hetlft[*][*] : float,
    homrgt[*][*] : float,
    hetrgt[*][*] : float
)
```

Arguments

x

Two-dimensional input array, dimensioned *num_stations* in x by *time*

y

Two-dimensional input array, dimensioned *num_stations* in y by *time*

nsvd

scalar integer that specifies the number of SVD patterns to be calculated

homlft

Left homogeneous array (output), a two dimensional array dimensioned *nsvd* x *num_stations* in x. Space for this must be explicitly allocated by the user.

hetlft

Left heterogeneous array (output), a two dimensional array dimensioned *nsvd* x *num_stations* in x. Space for this must be explicitly allocated by the user.

homrgt

Right homogeneous array (output), a two dimensional array dimensioned *nsvd* x *num_stations* in y. Space for this must be explicitly allocated by the user.

hetrgt

right heterogeneous array (output), a two dimensional array dimensioned *nsvd* x *num_stations* in y. Space for this must be explicitly allocated by the user.

Description

The functions `svdcov` and `svdstd` both compute the singular value decomposition (SVD) of *x* and *y* and return the percent variance explained by the patterns (a float array of length *nsvd*). In addition, `svdstd` standardizes the input data.

Both of these functions return the following attributes:

fnorm (scalar, float)
fnorm

condn (scalar, float)
condition number

rank (scalar, float)
calculated rank

lapack_err (scalar, float)
LAPACK error code

Example

```
begin
  ; ================================>  ; PARAMETERS
  ntime   = 8                         ; # time steps
  ncols   = 3                         ; # columns (stations or grid pts) for S
  ncolz   = 6                         ; # columns (stations or grid pts) for Z
  nsvd    = 3                         ; # svd patterns to calculate
                                      ;   [nsvd <= min(ncols, ncolz) ]
  xmsg    = -999.9                    ; missing value

; ================================>  ; READ THE ASCII FILE
                                    ; open "files" as 2-D
  s     = asciiread ("svdrdm_S.asc",(/ntime,ncols/), "float")
  z     = asciiread ("svdrdm_Z.asc",(/ntime,ncolz/), "float")

  s!0   = "time"                      ; name dimensions for reordering
  s!1   = "col"
  z!0   = "time"
  z!1   = "col"

  homlft = new((/nsvd,ncols/),float)
  hetlft = new((/nsvd,ncols/),float)
  homrgt = new((/nsvd,ncolz/),float)
  hetrgt = new((/nsvd,ncolz/),float)

  x = svdcov(s(col|:,time|:),z(col|:,time|:),nsvd,homlft,hetlft,homrgt,hetrgt)
  print("svdcov: percent variance= " + x)
```

```
  s2 = s(col |:,time |:)                ; for a cleaner look one might do this
  z2 = z(col |:,time |:)

  y = svdstd(s2,z2,nsvd,homlft,hetlft,homrgt,hetrgt)
  print("svdstd: percent variance= " + y)
end
```

---

system

This procedure is used to pass system commands to the shell.

Synopsis

```
        procedure system(
                command[1]: string
        )
```

Arguments

command
A single string shell command.

Description

The string *command* is passed to the shell environment and executed. When finished, control is returned to NCL.

---

systemfunc

This function is used to execute a system call and return the output as a string.

Synopsis

```
        function systemfunc(
                command[1]: string
        )
```

Arguments

command

A single string shell command.

Description

The string *command* is passed to the shell environment and executed. When finished, the output from the command is returned to NCL as a string.

tan

This function computes the tangent from radian input.

Synopsis

```
function tan(
        value:numeric
)
```

Arguments

value

An array of one or more values of any dimension in radians.

Description

The **tan** returns a floating-point array with the same dimensions as the input parameter *value*. The tangent is computed for each element of *value*. Missing values are ignored. If the input is of type **double** then the output is **double**.

tanh

This function computes the hyperbolic tangent from radian input.

Synopsis

```
function tanh(
        value:numeric
)
```

Arguments

value

An array of one or more floating-point values of any dimension in radians.

Description

The **tanh** returns a floating-point array with the same dimensions as the input parameter *value*. The hyperbolic tangent is computed for each element of *value*. Missing values are ignored. If the input is of type **double** then the output will be a **double**.

---

tdez2d

`tdez2d` uses Tdpack entries to draw a surface on a specified workstation.

Synopsis

```
procedure tdez2d(
    wid[1]    : graphic,
    x[*]      : float,
    y[*]      : float,
    z[*][*]   : float,
    rmult[1]  : float,
    theta[1]  : float,
    phi[1]    : float,
    ist[1]    : integer
)
```

Arguments

wid

An NCL workstation identifier for where you want to draw your surface. The identifier wid is the value returned from an NCL call that created a workstation.

x

A 1-dimensional array specifying X-coordinate values.

y

A 1-dimensional array specifying Y-coordinate values.

z

A 2-dimensional array specifying functional values at the X and Y coordinate values in the first two arguments. The values of z are stored with the first dimension varying the fastest, i.e. z(i,j) is the data value at (x(i),y(j)) for i=0, dimsizes(x) and j=0,dimsizes(y).

rmult, theta, phi

Values specifying an eye position (the point from which the surface will be viewed); these values are defined as follows:

- *rmult* is a multiplier of the diagonal length (DL) of the smallest box containing the surface to be drawn.
- *theta* is an angle (in degrees) in the XY plane measured positive counter-clockwise from the X axis
- *phi* is an angle (in degrees) measured from the positive Z axis toward the XY plane.

Thus, the coordinate (*rmult**DL,*theta,phi*) is the spherical coordinate for the eye position. If *rmult* = *theta* = *phi* = 0., a default eye position (2.5,-55.,70.) is chosen.

The point looked *at* is calculated to be the midpoint of the surface.

ist

A style index defining the colors used to shade the surface. The legal values for *ist* are as follows:

ist	**Description**
1	produce a wire-frame surface
2	use gray shades underneath; gray shades on top
3	use gray shades underneath; red shades on top
4	use gray shades underneath; green shades on top
5	use gray shades underneath; blue shades on top
6	use gray shades underneath; cyan shades on top
7	use gray shades underneath; gray shades on top
8	use gray shades underneath; magenta shades on top

If *ist* is positive, then black is used for the background color and white for the foreground color; if *ist* is the negative of any of the above values, then white is used for the background color and black for the foreground color. If *ist* falls outside of the legal range, it is defaulted to 6.

Description

When `tdez2d` is called, a color table is defined for the workstation specified by *wid* in the first argument. *This color table will supersede any color table that has been previously defined.* The color table that is defined is:

Color index	Colors
0	black if IST is positive; white if IST is negative
1	white if IST is positive; black if IST is negative
2	red
3	green
4	blue
5	cyan
6	magenta
7	yellow
8-37	grayscale from white to black
38-67	shades of gray
68-97	shades of red
98-127	shades of green
128-157	shades of blue
158-187	shades of cyan
188-217	shades of magenta
218-247	shades of yellow

`tdez2d` does *not* call `frame`.

Example

The following code fragment:

```
;
;  Create an X11 workstation.
```

```
;
      wid = create "tdez2d_example" xWorkstationClass defaultapp
        "wkPause" : True
      end create

;
;  Draw surface plot.
;
      tdez2d(wid, xo, yo, zo, 0., 0., 0., -6)
      frame(wid)
```

will draw a surface plot of the *zo* array in shades of cyan with a white background using a default eye position.

---

tdez3d

`tdez3d` uses Tdpack entries to draw an isosurface on a specified workstation.

Synopsis

```
procedure tdez3d(
    wid[1]     : graphic,
    x[*]       : float,
    y[*]       : float,
    z[*]       : float,
    u[*][*][*]: float,
    value[1]   : float,
    rmult[1]   : float,
    theta[1]   : float,
    phi[1]     : float,
    ist[1]     : integer
)
```

Arguments

wid
: An NCL workstation identifier for where you want to draw your surface. The identifier wid is the value returned from an NCL call that created a workstation.

x
: A 1-dimensional array specifying X-coordinate values.

y
: A 1-dimensional array specifying X-coordinate values.

z
: A 1-dimensional array specifying Z-coordinate values.

u

A 3-dimensional array specifying functional values at the X, Y, and Z coordinate values in the first three arguments. The values of u are stored with the first dimension varying the fastest, i.e. u(i,j,k) is the data value at (x(i),y(j),z(k)) for i=0, dimsizes(x), j=0,dimsizes(y), and k=0,dimsizes(z).

rmult, theta, phi

Values specifying an eye position (the point from which the isosurface will be viewed); these values are defined as follows:

- *rmult* is a multiplier of the diagonal length (DL) of the smallest box containing the surface to be drawn.
- *theta* is an angle (in degrees) in the XY plane measured positive counter-clockwise from the X axis
- *phi* is an angle (in degrees) measured from the positive Z axis toward the XY plane.

Thus, the coordinate (*rmult**DL,*theta,phi*) is the spherical coordinate for the eye position. If *rmult* = *theta* = *phi* = 0., a default eye position (2.5,-55.,70.) is chosen.

The point looked *at* is calculated to be the midpoint of the box determined by the *x*, *y*, and *z* array limits.

ist

A style index defining the colors used to shade the isosurface. The legal values for *ist* are as follows:

ist	**Description**
1	produce a wire-frame surface
2	use gray shades underneath; gray shades on top
3	use gray shades underneath; red shades on top
4	use gray shades underneath; green shades on top
5	use gray shades underneath; blue shades on top
6	use gray shades underneath; cyan shades on top
7	use gray shades underneath; gray shades on top
8	use gray shades underneath; magenta shades on top

If *ist* is positive, then black is used for the background color and white for the foreground color; if *ist* is the negative of any of the above values, then white is used for the background color and black for the foreground color. If *ist* falls outside of the legal range, it is defaulted to 6.

Description

When `tdez3d` is called, a color table is defined for the workstation specified by *wid* in the first argument. *This color table will supersede any color table that has been previously defined.* The color table that is defined is:

Color index	Colors
0	black if IST is positive; white if IST is negative
1	white if IST is positive; black if IST is negative
2	red
3	green
4	blue
5	cyan
6	magenta
7	yellow
8-37	grayscale from white to black
38-67	shades of gray
68-97	shades of red
98-127	shades of green
128-157	shades of blue
158-187	shades of cyan
188-217	shades of magenta
218-247	shades of yellow

`tdez3d` does *not* call `frame`.

Example

The following code fragment:

```
;
;  Create an X11 workstation.
;

      wid = create "tdez3d_example" xWorkstationClass defaultapp
        "wkPause" : True
```

```
      end create

;
;  Draw isosurface plot.
;
      tdez3d(wid, xo, yo, zo, u, 1., 0., 0., 0., 6)
      frame(wid)
```

will draw an isosurface plot of the *u* array (with isovalue 1.) in shades of cyan with a black background using a default eye position.

---

typeof

Returns the string name of the input type

Synopsis

```
function typeof(
        val
)
```

Arguments

val
: Any type and dimensionality

Description

Returns the string name of the input type.

---

undef

Used to undefine defined NCL symbols.

Synopsis

```
procedure undef(
        names:string
```

```
)
```

Arguments

names

string names of any dimensionality of symbols to be undefined

Description

This function is to be used primarily to undefine functions and procedures although it does work for variables. *undef* will remove from the symbol table any references matching the input. WARNING: once a function, procedure or variable is undefined there is no way to get it back.

NhlUpdateWorkstation

This procedure is used to update HLU workstation objects. (See NhlUpdateWorkstation .)

Synopsis

```
procedure NhlUpdateWorkstation(
        work : graphic
)

or

procedure update(
        work : graphic
)
```

Arguments

work

An array of one or more HLU Workstation instances of any dimensionality.

Description

The **NhlUpdateWorkstation** procedure is used to flush the internal buffers of HLU Workstation instances. It calls the HLU function NhlUpdateWorkstation for each element of the *work* array.

uv2dvf, uv2dvg

Given u and v wind components, compute the divergence via Spherepack.

Synopsis

```
procedure uv2dvf(
    u  : float,
    v  : float,
    dv : float
)

procedure uv2dvg(
    u  : float,
    v  : float,
    dv : float
)
```

Arguments

u, v
: wind components (input, arrays with two or more dimensions, last two dimensions must be *nlat* x *nlon* and input values must be in ascending latitude order)

dv
: divergence (output, same dimensions as *u* and *v*, values will be in ascending latitude order)

Description

Given wind components *u* and *v*, *uv2dvf* and *uv2dvg* both compute the divergence and return it in the array *dv*. *uv2dvf* operates on an equal (fixed) grid, and *uv2dvg* operates on a Gaussian grid.

Note: For the arrays whose last two dimensions are *nlat* x *nlon*, the rest of the dimensions (if any) are collectively referred to as *nt*. If the input/output arrays are just two dimensions, then *nt* can either be considered equal to 1 or nothing at all.

Arrays which have dimensions *nt* x *nlat* x *nlon* should **not** include the cyclic (wraparound) points when invoking the procedures and functions which use spherical harmonics (Spherepack).

For example, if an array *x* has dimensions *nlat* = 64 and *nlon* = 129, where the "129" represents the cyclic points, then the user should pass the data to the procedure/function via:

```
z = sample ( x([...],:,0:nlon-2) )  ; does not include cyclic points
```

Example

```
begin
  nlat  =  64                                        ; dimensions
  mlon  = 128
  mlon1 = mlon+1
  fbfile = "uv300.hs"
                                       ; Generic Workstation setup
  nrec  = fbinnumrec(fbfile)       ; total number of records in the file
  ntim  = nrec/2                       ; number of time steps in dataset

  uvmsg = 1e+36

  work  = new ( (/nlat,mlon1/), float, uvmsg )
  u     = new ( (/nlat,mlon /), float, uvmsg )  ; source u
  v     = new ( (/nlat,mlon /), float, uvmsg )  ; source v
  dv    = new ( (/nlat,mlon /), float, uvmsg )  ; divergence
  vr    = new ( (/nlat,mlon /), float, uvmsg )  ; vorticity  (relative)
  dvx   = new ( (/nlat,mlon /), float, uvmsg )  ; divergence (same as dv)
  vrx   = new ( (/nlat,mlon /), float, uvmsg )  ; vorticity  (same as vr)
  sf    = new ( (/nlat,mlon /), float, uvmsg )  ; stream function
  vp    = new ( (/nlat,mlon /), float, uvmsg )  ; velocity potential
  uy    = new ( (/nlat,mlon /), float, uvmsg )  ; latitudinal derivative (u)
  vy    = new ( (/nlat,mlon /), float, uvmsg )  ; latitudinal derivative (v)
  uu    = new ( (/nlat,mlon /), float, uvmsg )  ; reconstructed u
  vv    = new ( (/nlat,mlon /), float, uvmsg )  ; reconstructed v
  ux    = new ( (/nlat,mlon /), float, uvmsg )  ; reconstructed u
  vx    = new ( (/nlat,mlon /), float, uvmsg )  ; reconstructed v

  do i = 0,nrec-1,2
   month = 1                         ; january
   if (i .ge. 2) then
       month = 7                     ; july
   end if

   work = fbinrecread(fbfile,i  ,(/nlat,mlon1/),"float")
   u    = work(:,0:mlon-1)
   work = fbinrecread(fbfile,i+1,(/nlat,mlon1/),"float")
   v    = work(:,0:mlon-1)

   uv2dvg   (u,v,dv)             ; u,v ==> divergence
   uv2vrg   (u,v,vr)             ; u,v ==> vorticity (rel)
   uv2vrdvg (u,v, vrx,dvx)           ; u,v ==> div and vort
   uv2sfvpg (u,v, sf,vp)             ; u,v ==> stream function + velocity pot
   ud    = new ( (/nlat,mlon /), float, uvmsg )
   vd    = new ( (/nlat,mlon /), float, uvmsg )
   ur    = new ( (/nlat,mlon /), float, uvmsg )

   dv2uvg   (dv,  ud,vd)             ; dv  ==> divergent  wind components
   vr2uvg   (vr,  ur,vr)             ; vr  ==> rotational wind components
   vrdv2uvg (vr,dv,   uu,vv)         ; vr,dv > reconstruct original wind
   vrdv2uvg (vrx,dvx, ux,vx)         ; vr,dv > reconstruct original wind
  end do
```

```
end
```

Error messages

If *jer* or *ker* is equal to:

1 : error in the specification of *nlat*
2 : error in the specification of *nlon*
4 : error in the specification of *nt* (*jer* only)

---

uv2dvF, uv2dvG

Given u and v wind components, compute the divergence via Spherepack.

Synopsis

```
function uv2dvF(
    u : float,
    v : float
)

function uv2dvG(
    u : float,
    v : float
)
```

Arguments

u, v
: wind components (input, arrays with two or more dimensions, last two dimensions must be *nlat* x *nlon* and input values must be in ascending latitude order)

Description

Given wind components *u* and *v*, *uv2dvF* and *uv2dvG* both compute the divergence and return it as an array with the same dimensions as *u* and *v* (values will be in ascending latitude order). *uv2dvF* operates on an equal (fixed) grid, and *uv2dvG* operates on a Gaussian grid.

Note: For the arrays whose last two dimensions are *nlat* x *nlon*, the rest of the dimensions (if any) are collectively referred to as *nt*. If the input/output arrays are just two dimensions, then *nt* can either be considered equal to 1 or nothing at all.

Arrays which have dimensions *nt* x *nlat* x *nlon* should **not** include the cyclic (wraparound) points when invoking the procedures and functions which use spherical harmonics (Spherepack).

For example, if an array *x* has dimensions *nlat* = 64 and *nlon* = 129, where the "129" represents the cyclic points, then the user should pass the data to the procedure/function via:

```
    z = sample ( x([...],:,0:nlon-2) )  ; does not include cyclic points
```

Example

```
begin
  nlat  =  64                                        ; dimensions
  mlon  = 128
  mlon1 = mlon+1
  fbfile = "uv300.hs"
                                        ; Generic Workstation setup
  nrec  = fbinnumrec(fbfile)        ; total number of records in the file
  ntim  = nrec/2                        ; number of time steps in dataset

  uvmsg = 1e+36

  work  = new ( (/nlat,mlon1/), float, uvmsg )
  u     = new ( (/nlat,mlon /), float, uvmsg )  ; source u
  v     = new ( (/nlat,mlon /), float, uvmsg )  ; source v
  dvx   = new ( (/nlat,mlon /), float, uvmsg )  ; divergence (same as dv)
  vrx   = new ( (/nlat,mlon /), float, uvmsg )  ; vorticity  (same as vr)
  sf    = new ( (/nlat,mlon /), float, uvmsg )  ; stream function
  vp    = new ( (/nlat,mlon /), float, uvmsg )  ; velocity potential
  uy    = new ( (/nlat,mlon /), float, uvmsg )  ; latitudinal derivative (u)
  vy    = new ( (/nlat,mlon /), float, uvmsg )  ; latitudinal derivative (v)
  uu    = new ( (/nlat,mlon /), float, uvmsg )  ; reconstructed u
  vv    = new ( (/nlat,mlon /), float, uvmsg )  ; reconstructed v
  ux    = new ( (/nlat,mlon /), float, uvmsg )  ; reconstructed u
  vx    = new ( (/nlat,mlon /), float, uvmsg )  ; reconstructed v

  do i = 0,nrec-1,2
   month = 1                          ; january
   if (i .ge. 2) then
       month = 7                      ; july
   end if

   work = fbinrecread(fbfile,i  ,(/nlat,mlon1/),"float")
   u    = work(:,0:mlon-1)
   work = fbinrecread(fbfile,i+1,(/nlat,mlon1/),"float")
   v    = work(:,0:mlon-1)

   dv = uv2dvG   (u,v)                ; u,v ==> divergence
   vr = uv2vrG   (u,v)                ; u,v ==> vorticity (rel)
   uv2vrdvg (u,v, vrx,dvx)            ; u,v ==> div and vort
   uv2sfvpg (u,v, sf,vp)              ; u,v ==> stream function + velocity pot
   ud    = new ( (/nlat,mlon /), float, uvmsg )
```

```
   vd    = new ( (/nlat,mlon /), float, uvmsg )
   ur    = new ( (/nlat,mlon /), float, uvmsg )

   dv2uvg   (dv,  ud,vd)             ; dv  ==> divergent  wind components
   vr2uvg   (vr,  ur,vr)             ; vr  ==> rotational wind components
   vrdv2uvg (vr,dv,   uu,vv)         ; vr,dv > reconstruct original wind
   vrdv2uvg (vrx,dvx, ux,vx)         ; vr,dv > reconstruct original wind
  end do
end
```

Error messages

If *jer* or *ker* is equal to:

> 1 : error in the specification of *nlat*
> 2 : error in the specification of *nlon*
> 4 : error in the specification of *nt* (*jer* only)

---

uv2sfvpf, uv2sfvpg

Given u and v, compute the stream function and velocity potential via Spherepack.

Synopsis

```
procedure uv2sfvpf(
    u  : float,
    v  : float,
    sf : float,
    vp : float
)

procedure uv2sfvpg(
    u  : float,
    v  : float,
    sf : float,
    vp : float
)
```

Arguments

u, v
: wind components (input, arrays with two or more dimensions, last two dimensions must be *nlat* x *nlon* and input values must be in ascending latitude order)

sf

stream function (output, same dimensions as *u* and *v*, values will be in ascending latitude order)

vp

velocity potential (output, same dimensions as *u* and *v*, values will be in ascending latitude order)

Description

Given wind components *u* and *v*, *uv2sfvpf* and *uv2sfvpg* both compute the stream function and the velocity potential and return the results in the arrays *sf* and *vp*. *uv2sfvpf* operates on an equal (fixed) grid, and *uv2sfvpg* operates on a Gaussian grid.

Note: For the arrays whose last two dimensions are *nlat* x *nlon*, the rest of the dimensions (if any) are collectively referred to as *nt*. If the input/output arrays are just two dimensions, then *nt* can either be considered equal to 1 or nothing at all.

Arrays which have dimensions *nt* x *nlat* x *nlon* should **not** include the cyclic (wraparound) points when invoking the procedures and functions which use spherical harmonics (Spherepack).

For example, if an array *x* has dimensions *nlat* = 64 and *nlon* = 129, where the "129" represents the cyclic points, then the user should pass the data to the procedure/function via:

```
z = sample ( x([...],:,0:nlon-2) )  ; does not include cyclic points
```

Example

```
begin
  nlat  =  64                                       ; dimensions
  mlon  = 128
  mlon1 = mlon+1
  fbfile = "uv300.hs"
                                       ; Generic Workstation setup
  nrec  = fbinnumrec(fbfile)       ; total number of records in the file
  ntim  = nrec/2                       ; number of time steps in dataset

  uvmsg = 1e+36

  work  = new ( (/nlat,mlon1/), float, uvmsg )
  u     = new ( (/nlat,mlon /), float, uvmsg )  ; source u
  v     = new ( (/nlat,mlon /), float, uvmsg )  ; source v
  dvx   = new ( (/nlat,mlon /), float, uvmsg )  ; divergence (same as dv)
  vrx   = new ( (/nlat,mlon /), float, uvmsg )  ; vorticity  (same as vr)
  sf    = new ( (/nlat,mlon /), float, uvmsg )  ; stream function
  vp    = new ( (/nlat,mlon /), float, uvmsg )  ; velocity potential
  uy    = new ( (/nlat,mlon /), float, uvmsg )  ; latitudinal derivative (u)
  vy    = new ( (/nlat,mlon /), float, uvmsg )  ; latitudinal derivative (v)
  uu    = new ( (/nlat,mlon /), float, uvmsg )  ; reconstructed u
  vv    = new ( (/nlat,mlon /), float, uvmsg )  ; reconstructed v
  ux    = new ( (/nlat,mlon /), float, uvmsg )  ; reconstructed u
  vx    = new ( (/nlat,mlon /), float, uvmsg )  ; reconstructed v
```

```
  do i = 0,nrec-1,2
   month = 1                          ; january
   if (i .ge. 2) then
       month = 7                      ; july
   end if

   work = fbinrecread(fbfile,i  ,(/nlat,mlon1/),"float")
   u    = work(:,0:mlon-1)
   work = fbinrecread(fbfile,i+1,(/nlat,mlon1/),"float")
   v    = work(:,0:mlon-1)

   dv = uv2dvG   (u,v)                ; u,v ==> divergence
   vr = uv2vrG   (u,v)                ; u,v ==> vorticity (rel)
   uv2vrdvg (u,v, vrx,dvx)            ; u,v ==> div and vort
   uv2sfvpg (u,v, sf,vp)              ; u,v ==> stream function + velocity pot
   ud    = new ( (/nlat,mlon /), float, uvmsg )
   vd    = new ( (/nlat,mlon /), float, uvmsg )
   ur    = new ( (/nlat,mlon /), float, uvmsg )

   dv2uvg   (dv,  ud,vd)              ; dv  ==> divergent  wind components
   vr2uvg   (vr,  ur,vr)              ; vr  ==> rotational wind components
   vrdv2uvg (vr,dv,   uu,vv)          ; vr,dv > reconstruct original wind
   vrdv2uvg (vrx,dvx, ux,vx)          ; vr,dv > reconstruct original wind
  end do
end
```

Error messages

If *jer* or *ker* is equal to:

1 : error in the specification of *nlat*
2 : error in the specification of *nlon*
4 : error in the specification of *nt* (*jer* only)

---

uv2vrf, uv2vrg

Given u and v wind components, compute the vorticity via Spherepack.

Synopsis

```
    procedure uv2vrf(
        u    : float,
        v    : float,
        vort : float
    )

    procedure uv2vrg(
```

```
        u    : float,
        v    : float,
        vort : float
    )
```

Arguments

u, v

wind components (input, arrays with two or more dimensions, last two dimensions must be *nlat* x *nlon* and input values must be in ascending latitude order)

vort

vorticity (output, same dimensions as *u* and *v*, values will be in ascending latitude order)

Description

Given wind components *u* and *v*, *uv2vrf* and *uv2vrg* both compute the vorticity and return it in the array *vort*. *uv2vrf* operates on an equal (fixed) grid, and *uv2vrg* operates on a Gaussian grid.

Note: For the arrays whose last two dimensions are *nlat* x *nlon*, the rest of the dimensions (if any) are collectively referred to as *nt*. If the input/output arrays are just two dimensions, then *nt* can either be considered equal to 1 or nothing at all.

Arrays which have dimensions *nt* x *nlat* x *nlon* should **not** include the cyclic (wraparound) points when invoking the procedures and functions which use spherical harmonics (Spherepack).

For example, if an array *x* has dimensions *nlat* = 64 and *nlon* = 129, where the "129" represents the cyclic points, then the user should pass the data to the procedure/function via:

```
    z = sample ( x([...],:,0:nlon-2) )  ; does not include cyclic points
```

Example

```
begin
  nlat  =  64                                       ; dimensions
  mlon  = 128
  mlon1 = mlon+1
  fbfile = "uv300.hs"
                                         ; Generic Workstation setup
  nrec  = fbinnumrec(fbfile)        ; total number of records in the file
  ntim  = nrec/2                         ; number of time steps in dataset

  uvmsg = 1e+36

  work  = new ( (/nlat,mlon1/), float, uvmsg )
  u     = new ( (/nlat,mlon /), float, uvmsg )  ; source u
```

```
  v     = new ( (/nlat,mlon /), float, uvmsg )  ; source v
  dv    = new ( (/nlat,mlon /), float, uvmsg )  ; divergence
  vr    = new ( (/nlat,mlon /), float, uvmsg )  ; vorticity  (relative)
  dvx   = new ( (/nlat,mlon /), float, uvmsg )  ; divergence (same as dv)
  vrx   = new ( (/nlat,mlon /), float, uvmsg )  ; vorticity  (same as vr)
  sf    = new ( (/nlat,mlon /), float, uvmsg )  ; stream function
  vp    = new ( (/nlat,mlon /), float, uvmsg )  ; velocity potential
  uy    = new ( (/nlat,mlon /), float, uvmsg )  ; latitudinal derivative (u)
  vy    = new ( (/nlat,mlon /), float, uvmsg )  ; latitudinal derivative (v)
  uu    = new ( (/nlat,mlon /), float, uvmsg )  ; reconstructed u
  vv    = new ( (/nlat,mlon /), float, uvmsg )  ; reconstructed v
  ux    = new ( (/nlat,mlon /), float, uvmsg )  ; reconstructed u
  vx    = new ( (/nlat,mlon /), float, uvmsg )  ; reconstructed v

  do i = 0,nrec-1,2
   month = 1                        ; january
   if (i .ge. 2) then
       month = 7                    ; july
   end if

   work = fbinrecread(fbfile,i  ,(/nlat,mlon1/),"float")
   u    = work(:,0:mlon-1)
   work = fbinrecread(fbfile,i+1,(/nlat,mlon1/),"float")
   v    = work(:,0:mlon-1)

   uv2dvg   (u,v,dv)             ; u,v ==> divergence
   uv2vrg   (u,v,vr)             ; u,v ==> vorticity (rel)
   uv2vrdvg (u,v, vrx,dvx)            ; u,v ==> div and vort
   uv2sfvpg (u,v, sf,vp)              ; u,v ==> stream function + velocity pot
   ud    = new ( (/nlat,mlon /), float, uvmsg )
   vd    = new ( (/nlat,mlon /), float, uvmsg )
   ur    = new ( (/nlat,mlon /), float, uvmsg )

   dv2uvg   (dv,  ud,vd)              ; dv  ==> divergent  wind components
   vr2uvg   (vr,  ur,vr)              ; vr  ==> rotational wind components
   vrdv2uvg (vr,dv,   uu,vv)          ; vr,dv > reconstruct original wind
   vrdv2uvg (vrx,dvx, ux,vx)          ; vr,dv > reconstruct original wind
  end do
end
```

Error messages

If *jer* or *ker* is equal to:

> 1 : error in the specification of *nlat*
> 2 : error in the specification of *nlon*
> 4 : error in the specification of *nt* (*jer* only)

---

uv2vrF, uv2vrG

Given u and v wind components, compute the vorticity via Spherepack.

Synopsis

```
function uv2vrF(
    u : float,
    v : float
)

function uv2vrG(
    u : float,
    v : float
)
```

Arguments

u, v

: wind components (input, arrays with two or more dimensions, last two dimensions must be *nlat* x *nlon* and input values must be in ascending latitude order)

Description

Given wind components *u* and *v*, *uv2vrF* and *uv2vrG* both compute the vorticity and return it as an array with the same dimensions as *u* and *v* (values will be in ascending latitude order). *uv2vrF* operates on an equal (fixed) grid, and *uv2vrG* operates on a Gaussian grid.

Note: For the arrays whose last two dimensions are *nlat* x *nlon*, the rest of the dimensions (if any) are collectively referred to as *nt*. If the input/output arrays are just two dimensions, then *nt* can either be considered equal to 1 or nothing at all.

Arrays which have dimensions *nt* x *nlat* x *nlon* should **not** include the cyclic (wraparound) points when invoking the procedures and functions which use spherical harmonics (Spherepack).

For example, if an array *x* has dimensions *nlat* = 64 and *nlon* = 129, where the "129" represents the cyclic points, then the user should pass the data to the procedure/function via:

```
z = sample ( x([...],:,0:nlon-2) )  ; does not include cyclic points
```

Example

```
begin
  nlat  =  64                                       ; dimensions
  mlon  = 128
```

```
  mlon1 = mlon+1
  fbfile = "uv300.hs"
                                     ; Generic Workstation setup
  nrec  = fbinnumrec(fbfile)       ; total number of records in the file
  ntim  = nrec/2                     ; number of time steps in dataset

  uvmsg = 1e+36

  work  = new ( (/nlat,mlon1/), float, uvmsg )
  u     = new ( (/nlat,mlon /), float, uvmsg )  ; source u
  v     = new ( (/nlat,mlon /), float, uvmsg )  ; source v
  dvx   = new ( (/nlat,mlon /), float, uvmsg )  ; divergence (same as dv)
  vrx   = new ( (/nlat,mlon /), float, uvmsg )  ; vorticity  (same as vr)
  sf    = new ( (/nlat,mlon /), float, uvmsg )  ; stream function
  vp    = new ( (/nlat,mlon /), float, uvmsg )  ; velocity potential
  uy    = new ( (/nlat,mlon /), float, uvmsg )  ; latitudinal derivative (u)
  vy    = new ( (/nlat,mlon /), float, uvmsg )  ; latitudinal derivative (v)
  uu    = new ( (/nlat,mlon /), float, uvmsg )  ; reconstructed u
  vv    = new ( (/nlat,mlon /), float, uvmsg )  ; reconstructed v
  ux    = new ( (/nlat,mlon /), float, uvmsg )  ; reconstructed u
  vx    = new ( (/nlat,mlon /), float, uvmsg )  ; reconstructed v

  do i = 0,nrec-1,2
   month = 1                         ; january
   if (i .ge. 2) then
       month = 7                     ; july
   end if

   work = fbinrecread(fbfile,i  ,(/nlat,mlon1/),"float")
   u    = work(:,0:mlon-1)
   work = fbinrecread(fbfile,i+1,(/nlat,mlon1/),"float")
   v    = work(:,0:mlon-1)

   dv = uv2dvG   (u,v)             ; u,v ==> divergence
   vr = uv2vrG   (u,v)             ; u,v ==> vorticity (rel)
   uv2vrdvg (u,v, vrx,dvx)         ; u,v ==> div and vort
   uv2sfvpg (u,v, sf,vp)           ; u,v ==> stream function + velocity pot
   ud    = new ( (/nlat,mlon /), float, uvmsg )
   vd    = new ( (/nlat,mlon /), float, uvmsg )
   ur    = new ( (/nlat,mlon /), float, uvmsg )

   dv2uvg   (dv,  ud,vd)             ; dv  ==> divergent  wind components
   vr2uvg   (vr,  ur,vr)             ; vr  ==> rotational wind components
   vrdv2uvg (vr,dv,   uu,vv)         ; vr,dv > reconstruct original wind
   vrdv2uvg (vrx,dvx, ux,vx)         ; vr,dv > reconstruct original wind
  end do
end
```

Error messages

If *jer* or *ker* is equal to:

> 1 : error in the specification of *nlat*
> 2 : error in the specification of *nlon*
> 4 : error in the specification of *nt* (*jer* only)

uv2vrdvf, uv2vrdvg

Given (u,v) compute the vorticity and divergence via Spherepack.

Synopsis

```
procedure uv2vrdvf(
    u  : float,
    v  : float,
    vr : float,
    dv : float
)

procedure uv2vrdvg(
    u  : float,
    v  : float,
    vr : float,
    dv : float
)
```

Arguments

u, v
: input arrays with two or more dimensions, last two dimensions must be *nlat* x *nlon* and input values must be in ascending latitude order

vr
: vorticity array (output, same dimensions as *u* and *v*, values will be in ascending latitude order)

dv
: divergence array (output, same dimensions as *u* and *v*, values will be in ascending latitude order)

Description

uv2vrdvf and *uv2vrdvg* take as input *u* and *v* and return the vorticity and divergence in the arrays *vr* and *dv*. *uv2vrdvf* operates on an equal (fixed) grid, and *uv2vrdvg* operates on a Gaussian grid.

Note: For the arrays whose last two dimensions are *nlat* x *nlon*, the rest of the dimensions (if any) are collectively referred to as *nt*. If the input/output arrays are just two dimensions, then *nt* can either be considered equal to 1 or nothing at all.

Arrays which have dimensions *nt* x *nlat* x *nlon* should **not** include the cyclic (wraparound) points when invoking the procedures and functions which use spherical harmonics (Spherepack).

For example, if an array *x* has dimensions *nlat* = 64 and *nlon* = 129, where the "129" represents the cyclic points, then the user should pass the data to the procedure/function via:

```
z = sample ( x([...],:,0:nlon-2) )  ; does not include cyclic points
```

Example

```
begin
  nlat  =  64                                        ; dimensions
  mlon  = 128
  mlon1 = mlon+1
  fbfile = "uv300.hs"
                                               ; Generic Workstation setup
  nrec  = fbinnumrec(fbfile)       ; total number of records in the file
  ntim  = nrec/2                       ; number of time steps in dataset

  uvmsg = 1e+36

  work  = new ( (/nlat,mlon1/), float, uvmsg )
  u     = new ( (/nlat,mlon /), float, uvmsg )  ; source u
  v     = new ( (/nlat,mlon /), float, uvmsg )  ; source v
  dvx   = new ( (/nlat,mlon /), float, uvmsg )  ; divergence (same as dv)
  vrx   = new ( (/nlat,mlon /), float, uvmsg )  ; vorticity  (same as vr)
  sf    = new ( (/nlat,mlon /), float, uvmsg )  ; stream function
  vp    = new ( (/nlat,mlon /), float, uvmsg )  ; velocity potential
  uy    = new ( (/nlat,mlon /), float, uvmsg )  ; latitudinal derivative (u)
  vy    = new ( (/nlat,mlon /), float, uvmsg )  ; latitudinal derivative (v)
  uu    = new ( (/nlat,mlon /), float, uvmsg )  ; reconstructed u
  vv    = new ( (/nlat,mlon /), float, uvmsg )  ; reconstructed v
  ux    = new ( (/nlat,mlon /), float, uvmsg )  ; reconstructed u
  vx    = new ( (/nlat,mlon /), float, uvmsg )  ; reconstructed v

  do i = 0,nrec-1,2
   month = 1                         ; january
   if (i .ge. 2) then
       month = 7                     ; july
   end if

   work = fbinrecread(fbfile,i  ,(/nlat,mlon1/),"float")
   u    = work(:,0:mlon-1)
   work = fbinrecread(fbfile,i+1,(/nlat,mlon1/),"float")
   v    = work(:,0:mlon-1)

   dv = uv2dvG   (u,v)               ; u,v ==> divergence
   vr = uv2vrG   (u,v)               ; u,v ==> vorticity (rel)
   uv2vrdvg (u,v, vrx,dvx)           ; u,v ==> div and vort
   uv2sfvpg (u,v, sf,vp)             ; u,v ==> stream function + velocity pot
   ud    = new ( (/nlat,mlon /), float, uvmsg )
   vd    = new ( (/nlat,mlon /), float, uvmsg )
   ur    = new ( (/nlat,mlon /), float, uvmsg )

   dv2uvg   (dv,  ud,vd)             ; dv  ==> divergent  wind components
   vr2uvg   (vr,  ur,vr)             ; vr  ==> rotational wind components
   vrdv2uvg (vr,dv,   uu,vv)         ; vr,dv > reconstruct original wind
   vrdv2uvg (vrx,dvx, ux,vx)         ; vr,dv > reconstruct original wind
  end do
```

```
end
```

Error messages

If *jer* or *ker* is equal to:

> 1 : error in the specification of *nlat*
> 2 : error in the specification of *nlon*
> 4 : error in the specification of *nt* (*jer* only)

variance

The function computes the variance of all input points regardless of dimensionality.

Synopsis

```
function variance(
        x : numeric
)
```

Arguments

x

An array of any dimensionality.

Description

Computes the variance for all input points.

vhaec, vhagc

Procedures for computing vector harmonic analyses via Spherepack.

Synopsis

```
procedure vhaec(
    u  : float
    v  : float,
    br : float,
    bi : float,
    cr : float,
    ci : float
)

procedure vhagc(
    u  : float,
    v  : float,
    br : float,
    bi : float,
    cr : float,
    ci : float
)
```

Arguments

u, v
: vector function to be analyzed (input, array with two or more dimensions, last two dimensions must be *nlat* x *nlon* and values must be in ascending latitude order, but see note below)

br, bi, cr, ci
: vector spherical harmonic coefficients (input/output, same dimensions as *u* and *v*, except the last two dimensions must be *nlat* x *nlat* and values must/will be in ascending latitude order, but see note below)

Description

> **Note: If you have version 4.1 of NCL, these procedures will not work with arrays greater than two dimensions. In order to use arrays with more than two dimensions, you need to run version 4.1.1 of NCL. If you are licensed for NCAR Graphics software version 4.1, then you should automatically receive version 4.1.1. Typing "ncl" on the command line will tell you which version you are running.**

vhaec and *vhagc* both perform the vector spherical harmonic analysis on the vector field *u* and *v* and store the results in the arrays *br*, *bi*, *cr*, and *ci*. *vhaec* operates on an equal (fixed) grid, and *vhagc* operates on a Gaussian grid.

For the arrays whose last two dimensions are *nlat* x *nlon*, the rest of the dimensions (if any) are collectively referred to as *nt*. If the input/output arrays are just two dimensions, then *nt* can either be considered equal to 1 or nothing at all.

Arrays which have dimensions *nt* x *nlat* x *nlon* should **not** include the cyclic (wraparound) points when invoking the procedures and functions which use spherical harmonics (Spherepack).

For example, if an array *x* has dimensions *nlat* = 64 and *nlon* = 129, where the "129" represents the cyclic points, then the user should pass the data to the procedure/function via:

```
z = sample ( x([...],:,0:nlon-2) )  ; does not include cyclic points
```

Error messages

If *jer* or *ker* is equal to:

1 : error in the specification of *nlat*
2 : error in the specification of *nlon*
4 : error in the specification of *nt* (*jer* only)

---

vhsec, vhsgc

Procedures for computing vector spherical harmonic syntheses via Spherepack.

Synopsis

```
procedure vhsec(
    br : float,
    bi : float,
    cr : float,
    ci : float,
    u  : float,
    v  : float
)

procedure vhsgc(
    br : float,
    bi : float,
    cr : float,
    ci : float,
    u  : float,
    v  : float
)
```

Arguments

br, bi, cr, ci

vector spherical harmonic coefficients (input, array with two or more dimensions, last two dimensions must be *nlat* x *nlat* and values must be in ascending latitude order, but see note below)

u, v

vector spherical harmonic synthesis arrays (input/output, same dimensions as *br, bi, cr, ci*, except the last two dimensions must be *nlat* x *nlon* and values must/will be in ascending latitude order, but see note below)

Description

> **Note: If you have version 4.1 of NCL, these procedures will not work with arrays greater than two dimensions. In order to use arrays with more than two dimensions, you need to run version 4.1.1 of NCL. If you are licensed for NCAR Graphics software version 4.1, then you should automatically receive version 4.1.1. Typing "ncl" on the command line will tell you which version you are running.**

vhsec and *vhsgc* both perform the vector spherical harmonic synthesis on the arrays *br*, *bi*, *cr*, and *ci* and store the results in the arrays *u* and *v*. *vhsec* operates on an equal (fixed) grid, and *vhsgc* operates on a Gaussian grid.

For the arrays whose last two dimensions are *nlat* x *nlon*, the rest of the dimensions (if any) are collectively referred to as *nt*. If the input/output arrays are just two dimensions, then *nt* can either be considered equal to 1 or nothing at all.

Arrays which have dimensions *nt* x *nlat* x *nlon* should **not** include the cyclic (wraparound) points when invoking the procedures and functions which use spherical harmonics (Spherepack).

For example, if an array *x* has dimensions *nlat* = 64 and *nlon* = 129, where the "129" represents the cyclic points, then the user should pass the data to the procedure/function via:

```
z = sample ( x([...],:,0:nlon-2) )  ; does not include cyclic points
```

Error messages

If *jer* or *ker* is equal to:

> 1 : error in the specification of *nlat*
> 2 : error in the specification of *nlon*
> 4 : error in the specification of *nt* (*jer* only)

---

vibeta

Function for doing vertical integration using beta factors.

Synopsis

```
    function vibeta(
        p[*]      : float,
        x         : float,
        linlog[1] : integer,
        psfc      : float,
        pbot[1]   : float,
        ptop[1]   : float
    )
```

Arguments

p
: pressure levels (input, 1d array of at least length 3)

x
: quantity to be integrated (input, array of any dimensionality, last dimension must be same length as *p*). Missing values should be indicated by *x*@_FillValue. If *x*@_FillValue is not set, then the NCL default (-999) will be assumed.

linlog
: scalar, linear (1) or log (2) interpolation

psfc
: surface pressure (must be same dimensions as *x*, minus the last dimension)

pbot, ptop
: scalar, lower and upper limits of integration

Description

Function for doing vertical integration using Beta factors (references: Boer (1982) Mon. Wea. Rev pp. 1801-20 and Trenberth (1992) NCAR Tech Note 373). The return value is the same size as *psfc*.

Example

```
begin
```

```
  nlvl   = 17
  linlog = 1

  p      = (/ 1000.,925.,850.,700.,600.,500., \
                 400.,300.,250.,200.,150.,100., \
                  70.,50.,30.,20.,10. /)
  psfc   = 1013.
  pbot   = 1100.
  ptop   = p(nlvl-1)
  plvcrt = p(nlvl-1)

  x      = new ( nlvl ,float)
  do nl=0,nlvl-1
     x(nl) = nlvl-nl
  end do

  vint = vibeta (p,x,linlog,psfc,pbot,ptop)
end
```

---

vinth2p

This function interpolates CCM2/3 hybrid coordinate data to pressure coordinates using pressure surfaces as the coordinate surface where the interpolation is done.

Synopsis

```
        function vinth2p(
                datai[*][*][*] : float,
                hbcofa[*] : float,
                hbcofb[*] : float,
                plevo[*] : float,
                psfc[*][*] : float,
                intyp[1] : integer,
                p0[1] : float,
                ilev[1] : integer,
                kxtrp[1] : logical
        )
```

Arguments

datai
: 3 dimensional data with the vertical level as the 0-th (leftmost) dimension

hbcofa
: Hybrid coefficients A, a single vector dimensioned the same as the 0'th dimension of *datai*

hbcofb
: Hybrid coefficients B, a single vector dimensioned the same as the 0'th dimension of *datai*

plevo

An array of desired output pressure levels (mb)

psfc

2 dimensional surface pressure data (Pa). These two dimensions should agree with the two rightmost dimensions of *datai*

intyp

Single scalar value to determine interpolation type 1 - LINEAR, 2 - LOG, 3 - LOG LOG

p0

Single scalar float value that is the surface reference pressure (mb)

ilev

Single scalar integer value that specifies whether the data exist on level midpoints or level interfaces, 1 - interface levels, 2 - midpoint levels

kxtrp

If False no extrapolation is done when the pressure level is ouside of the range of psfc.

Description

This function interpolates CCM2/3 hybrid coordinate data to pressure coordinates using pressure surfaces as the coordinate surface where the interpolation is done. The type of interpolation is currently a variant of transformed pressure coordinates with the interpolation type specified by *intyp*. All hybrid coordinate values are transformed to pressure values.

The output variable from *vinth2p* will be a 3-dimensional field with the 0-th (leftmost dimension) being the same dimension as *plevo* [dimsizes(plevo)] and the next two dimensions being the same as in "datai." The function automatically creates the 3D space for the returned variable (the user need not pre-allocate space via the "new" statement). If there are more than 3 dimensions, then the user must explicitly allocate the required memory (see example 3 below).

Note: This is the *exact* routine used within the CCM Processor. The mixture of Pa for *psfc* and mb for *plevo* and *p0* is specified by the source routine.

Examples

Example 1:

The models which produce files in CCM History Tape format write the variables in the following order ([time],lat,lev,lon). This example has no time dimension. The function "vinth2p" requires the "lev" dimension to be the slowest varying (leftmost) dimension. Thus, the following example uses NCL's dimension reordering to put the dimensions into the required order. The surface pressure (PS) has two dimensions (lat,lon).

```
fccm = addfile ("dummy.ccm" , "r")  ; a CCM file
pnew = (/ 850.0,200.0 /)            ; desired output levels [hPa/mb]
hyam = fccm->hyam                   ; read to memory [optional]
hybm = fccm->hybm
```

```
T    = fccm->T                          ; say "T" is (lat,lev,lon)
P0mb = 1000.                            ; reference pressure [mb]

Tnew = vinth2p (T(lev|:, lat|:, lon|:) \      ; dimension reorder
                ,hyam, hybm, pnew ,fccm->PS, 1 ,P0mb, 2, True)
```

`Tnew` is a 3D array returned by "vinth2p" with dimensions:

Tnew(dimsizes(pnew),lat,lon) ==> Tnew(2,lat,lon)

Example 2:

Let's assume that the above file had been converted to netCDF format with the following dimension order (lev,lat,lon). This is typically the case when the "ccm2nc" tool is used. (See http://www.cgd.ucar.edu/pubsoft/.) In this case, the "lev" dimension is as required by "vint2p" and dimension reordering is not required. Also, "ccm2nc" explicitly adds the reference pressure [P0; Pa] to the file so it may be retrieved from the file (fncf->P0). Thus,

```
fncf = addfile ("dummy.nc" , "r")  ; a CCM file converted to netCDF via "ccm2nc"
P0mb = 0.01*fncf->P0               ; change Pa to mb (as required)
      .
      .
Tnew = vinth2p (T ,hyam, hybm, pnew ,fncf->PS, 1 ,P0mb, 2, True)
```

Example 3:

A variable with a time dimension is being read directly from a CCMHT file. Each variable's dimension order is (time,lat,lev,lon). We want to interpolate each variable with four dimensions to the specified pressure levels and we want the output to be in (time,lev_new,lat,lon) order. The program will then proceed to operate on each variable. Naming dimensions and creating coordinate variables is also demonstrated.

```
fccm = addfile ("dummy.ccm" , "r")  ; a CCM file
pnew = (/ 850., 500., 200. /)       ; desired output levels
hyam = fccm->hyam                   ; read to memory (optional)
hybm = fccm->hybm
time = fccm->time
P0mb = 1000.                        ; reference pressure [mb]

vname= getfilevarnames(fccm)        ; get all variable names on input file
                                    ; preallocate memory
Vnew = new ((/dimsizes(time),dimsizes(pnew),dimsizes(lat),dimsizes(lon) )
Vnew!0 = "time"                     ; name dimensions (not required)
Vnew!1 = "lev_new"
Vnew!2 = "lat"
Vnew!3 = "lon"

lev_new   = pnew                    ; create a new coordinate variable
lev_new!0 = "lev_new"               ; variable and dimension name the same

do i=0,dimsizes(vnames)-1
   dims  = getfilevardims(fccm,vname(i))
   nrank = dimsizes(dims)           ; determine the number of dimensions
   if (nrank.eq.4) then             ; only interp 4D variables
       Vold = fccm->$vname(i)$      ; read each variable to memory
                                    ; (faster dimension reordering)
```

```
        do nt=0, dimsizes(time)-1    ; interpolate at each time step
           Vnew(nt,:,:,:) = vinth2p (Vold(nt,lev|:, lat|:, lon|:) \ ; dimension re
                                    ,hyam, hybm, pnew ,fccm->PS(nt,:,:), 1    \
                                    ,P0, 2, True)
        end do

    end if
;   do something with Vnew
 end do
```

Example 4:

Again, let's assume that "ccm2nc" had been used to create a netCDF file with the following dimension ordering (time,lev,lat,lon). Thus dimension reordering is not required. Either of the following approaches could be used:

```
 fncf = addfile ("dummy.nc" , "r")  ; a CCM file created by "ccm2nc"

    [snip]
        Vold = fncf->$vname(i)$      ; read to memory [could be large]

        do nt=0, dimsizes(time)-1    ; interpolate at each time step
           Vnew(nt,:,:,:) = vinth2p (Vold(nt,:,:,:) \
                                    ,hyam, hybm, pnew ,fncf->PS(nt,:,:), 1    \
                                    ,P0mb, 2, True)
        end do
```

or:

Do NOT read all of `Vold` to memory (in other words, don't use `Vold = fncf->$vname(i)$`). Just read each time step from the file. This is a bit slower but uses minimum memory.

```
do nt=0, dimsizes(time)-1    ; interpolate at each time step
   Vnew(nt,:,:,:) = vinth2p (fncf->$vname(i)$(nt,:,:,:) \
                            ,hyam, hybm, pnew ,fncf->PS(nt,:,:), 1 \
                            ,P0mb, 2, True)
end do
```

---

vr2uvf, vr2uvg

Given relative vorticity, compute the non-divergent (rotational) wind components via Spherepack.

Synopsis

```
procedure vr2uvf(
    vort : float,
    ur   : float,
    vr   : float
)
```

```
procedure vr2uvg(
    vort : float,
    ur   : float,
    vr   : float
)
```

Arguments

vort
: relative vorticity array (input, array with two or more dimensions, last two dimensions must be *nlat* x *nlon* and input values must be in ascending latitude order)

ur, vr
: non-divergent wind components (output, same dimensions as *vort*, values will be in ascending latitude order)

Description

vr2uvf and *vr2uvg* both compute the non-divergent wind components given vorticity array *vort* and store the results in the arrays *ur, vr*. *nt* can be any number of dimensions. *vr2uvf* operates on an equal (fixed) grid, and *vr2uvg* operates on a Gaussian grid.

Note: For the arrays whose last two dimensions are *nlat* x *nlon*, the rest of the dimensions (if any) are collectively referred to as *nt*. If the input/output arrays are just two dimensions, then *nt* can either be considered equal to 1 or nothing at all.

Arrays which have dimensions *nt* x *nlat* x *nlon* should **not** include the cyclic (wraparound) points when invoking the procedures and functions which use spherical harmonics (Spherepack).

For example, if an array *x* has dimensions *nlat* = 64 and *nlon* = 129, where the "129" represents the cyclic points, then the user should pass the data to the procedure/function via:

```
z = sample ( x([...],:,0:nlon-2) )  ; does not include cyclic points
```

Example

```
begin
  nlat  =  64                                       ; dimensions
  mlon  = 128
  mlon1 = mlon+1
  fbfile = "uv300.hs"
                                        ; Generic Workstation setup
  nrec  = fbinnumrec(fbfile)        ; total number of records in the file
  ntim  = nrec/2                        ; number of time steps in dataset
```

```
  uvmsg = 1e+36

  work  = new ( (/nlat,mlon1/), float, uvmsg )
  u     = new ( (/nlat,mlon /), float, uvmsg )  ; source u
  v     = new ( (/nlat,mlon /), float, uvmsg )  ; source v
  dvx   = new ( (/nlat,mlon /), float, uvmsg )  ; divergence (same as dv)
  vrx   = new ( (/nlat,mlon /), float, uvmsg )  ; vorticity  (same as vr)
  sf    = new ( (/nlat,mlon /), float, uvmsg )  ; stream function
  vp    = new ( (/nlat,mlon /), float, uvmsg )  ; velocity potential
  uy    = new ( (/nlat,mlon /), float, uvmsg )  ; latitudinal derivative (u)
  vy    = new ( (/nlat,mlon /), float, uvmsg )  ; latitudinal derivative (v)
  uu    = new ( (/nlat,mlon /), float, uvmsg )  ; reconstructed u
  vv    = new ( (/nlat,mlon /), float, uvmsg )  ; reconstructed v
  ux    = new ( (/nlat,mlon /), float, uvmsg )  ; reconstructed u
  vx    = new ( (/nlat,mlon /), float, uvmsg )  ; reconstructed v

  do i = 0,nrec-1,2
   month = 1                        ; january
   if (i .ge. 2) then
       month = 7                    ; july
   end if

   work = fbinrecread(fbfile,i  ,(/nlat,mlon1/),"float")
   u    = work(:,0:mlon-1)
   work = fbinrecread(fbfile,i+1,(/nlat,mlon1/),"float")
   v    = work(:,0:mlon-1)

   dv = uv2dvG   (u,v)              ; u,v ==> divergence
   vr = uv2vrG   (u,v)              ; u,v ==> vorticity (rel)
   uv2vrdvg (u,v, vrx,dvx)          ; u,v ==> div and vort
   uv2sfvpg (u,v, sf,vp)            ; u,v ==> stream function + velocity pot
   ud    = new ( (/nlat,mlon /), float, uvmsg )
   vd    = new ( (/nlat,mlon /), float, uvmsg )
   ur    = new ( (/nlat,mlon /), float, uvmsg )

   dv2uvg   (dv,  ud,vd)            ; dv  ==> divergent  wind components
   vr2uvg   (vr,  ur,vr)            ; vr  ==> rotational wind components
   vrdv2uvg (vr,dv,   uu,vv)        ; vr,dv > reconstruct original wind
   vrdv2uvg (vrx,dvx, ux,vx)        ; vr,dv > reconstruct original wind
  end do
end
```

Error messages

If *jer* or *ker* is equal to:

> 1 : error in the specification of *nlat*
> 2 : error in the specification of *nlon*
> 4 : error in the specification of *nt* (*jer* only)

---

vrdv2uvf, vrdv2uvg

Given vorticity and divergence, compute the wind components via Spherepack.

Synopsis

```
procedure vrdv2uvf(
    vr : float,
    dv : float,
    ud : float,
    vd : float
)

procedure vrdv2uvg(
    vr : float,
    dv : float,
    ud : float,
    vd : float
)
```

Arguments

vr

vorticity array (input, array with two or more dimensions, last two dimensions must be *nlat* x *nlon* and input values must be in ascending latitude order)

dv

divergence array (input, same dimensions as *dv*, values will be in ascending latitude order)

ud, vd

wind components (output, same dimensions as *vr* and *dv*, values will be in ascending latitude order)

Description

vrdv2uvf and *vrdv2uvg* both compute the wind components given vorticity and divergent arrays *vr* and *dv* and store the results in the arrays *ud* and *vd*. *vrdv2uvf* operates on an equal (fixed) grid, and *vrdv2uvg* operates on a Gaussian grid.

Note: For the arrays whose last two dimensions are *nlat* x *nlon*, the rest of the dimensions (if any) are collectively referred to as *nt*. If the input/output arrays are just two dimensions, then *nt* can either be considered equal to 1 or nothing at all.

Arrays which have dimensions *nt* x *nlat* x *nlon* should **not** include the cyclic (wraparound) points when invoking the procedures and functions which use spherical harmonics (Spherepack).

For example, if an array *x* has dimensions *nlat* = 64 and *nlon* = 129, where the "129" represents the cyclic points, then the user should pass the data to the procedure/function via:

```
    z = sample ( x([...],:,0:nlon-2) )  ; does not include cyclic points
```

Example

```
begin
  nlat  =  64                                       ; dimensions
  mlon  = 128
  mlon1 = mlon+1
  fbfile = "uv300.hs"
                                       ; Generic Workstation setup
  nrec  = fbinnumrec(fbfile)       ; total number of records in the file
  ntim  = nrec/2                       ; number of time steps in dataset

  uvmsg = 1e+36

  work  = new ( (/nlat,mlon1/), float, uvmsg )
  u     = new ( (/nlat,mlon /), float, uvmsg )  ; source u
  v     = new ( (/nlat,mlon /), float, uvmsg )  ; source v
  dvx   = new ( (/nlat,mlon /), float, uvmsg )  ; divergence (same as dv)
  vrx   = new ( (/nlat,mlon /), float, uvmsg )  ; vorticity  (same as vr)
  sf    = new ( (/nlat,mlon /), float, uvmsg )  ; stream function
  vp    = new ( (/nlat,mlon /), float, uvmsg )  ; velocity potential
  uy    = new ( (/nlat,mlon /), float, uvmsg )  ; latitudinal derivative (u)
  vy    = new ( (/nlat,mlon /), float, uvmsg )  ; latitudinal derivative (v)
  uu    = new ( (/nlat,mlon /), float, uvmsg )  ; reconstructed u
  vv    = new ( (/nlat,mlon /), float, uvmsg )  ; reconstructed v
  ux    = new ( (/nlat,mlon /), float, uvmsg )  ; reconstructed u
  vx    = new ( (/nlat,mlon /), float, uvmsg )  ; reconstructed v

  do i = 0,nrec-1,2
   month = 1                          ; january
   if (i .ge. 2) then
       month = 7                      ; july
   end if

   work = fbinrecread(fbfile,i  ,(/nlat,mlon1/),"float")
   u    = work(:,0:mlon-1)
   work = fbinrecread(fbfile,i+1,(/nlat,mlon1/),"float")
   v    = work(:,0:mlon-1)

   dv = uv2dvG   (u,v)                ; u,v ==> divergence
   vr = uv2vrG   (u,v)                ; u,v ==> vorticity (rel)
   uv2vrdvg (u,v, vrx,dvx)            ; u,v ==> div and vort
   uv2sfvpg (u,v, sf,vp)              ; u,v ==> stream function + velocity pot
   ud    = new ( (/nlat,mlon /), float, uvmsg )
   vd    = new ( (/nlat,mlon /), float, uvmsg )
   ur    = new ( (/nlat,mlon /), float, uvmsg )

   dv2uvg   (dv,  ud,vd)              ; dv  ==> divergent  wind components
   vr2uvg   (vr,  ur,vr)              ; vr  ==> rotational wind components
   vrdv2uvg (vr,dv,   uu,vv)          ; vr,dv > reconstruct original wind
   vrdv2uvg (vrx,dvx, ux,vx)          ; vr,dv > reconstruct original wind
  end do
end
```

Error messages

If *jer* or *ker* is equal to:

> 1 : error in the specification of *nlat*
> 2 : error in the specification of *nlon*
> 4 : error in the specification of *nt* (*jer* only)

wmbarb

The procedure `wmbarb` uses the wind barb functions from the Wmap package to draw wind barbs. **Note:** This procedure is only available in version 4.1.1 of NCL. If your site is licensed for version 4.1, then you should receive version 4.1.1 automatically.

Synopsis

```
procedure wmbarb(
    wid[1]    : graphic,
    x[*]      : float,
    y[*]      : float,
    u[*]      : float,
    v[*]      : float,
)
```

Arguments

wid
: An NCL workstation identifier specifying where you want to draw your wind barbs. The identifier wid is the value returned from an NCL call that created a workstation.

x
: A 1-dimensional array specifying X-coordinate values.

y
: A 1-dimensional array specifying Y-coordinate values.

u
: A 1-dimensional array specifying X-components of wind vectors at the associated (x,y) coordinates specified in arguments two and three of this procedure.

u
: A 1-dimensional array specifying Y-components of wind vectors at the associated (x,y) coordinates specified in arguments two and three of this procedure.

Description

The procedure `wmbarb` draws wind barbs at the (x,y) coordinates specified in arguments two and three of the procedure. The (x,y) coordinates specify the center position of the base of the barb and u and v specify the components of the wind vector. The angle at which the wind barb is drawn is determined by the vector (u,v) and the magnitude of that vector determines the wind speed in knots.

Certain parameters may be set to control the appearance of a wind barb, such as its size, color, and so forth. The procedure `wmsetp` is used to set parameter values, and the function `wmgetp` is used to retrieve parameter values. The parameters applicable to `wmbarb` are: COL, WBF, WBA, WBC, WBD, WBR, WBS, WBT.

The procedure `wmbarb` does *not* call `frame`.

Example

The following code:

```
begin
;
;  Create an X11 workstation.
;

      wid = create "wmbarb_example" xWorkstationClass defaultapp
        "wkPause" : True
      end create
;
;  Draw wind barbs.
;
      x = (/0.25, 0.75, 0.75, 0.25/)
      y = (/0.25, 0.25, 0.75, 0.75/)
      u = (/50., -50., -50.,  50./)
      v = (/50.,  50., -50., -50./)
      wmsetp("wbs",0.2)
      wmbarb(wid, x, y, u, v)
      frame(wid)
;
;  Get parameter value.
;
      size = wmgetp("wbs")
      print(size)
end
```

will draw four wind barbs of the same magnitude, but at different locations and in different directions.

wmsetp, wmgetp

Set and retrieve parameters for selected Wmap routines. The NCL language affords a convenient implementation of the parameter setting and retrieval functions. The parameters currently valid for wmsetp and wmgetp are those that apply to wmbarb: COL, WBF, WBA, WBC, WBD, WBR, WBS, WBT.

Note: These functions and procedures are only available in version 4.1.1 of NCL. If your site is licensed for version 4.1, then you should receive version 4.1.1 automatically.

Synopsis

```
    procedure wmsetp(
        pnam[1] : string,
        pval[1]
    )

    function wmgetp(
        pnam[1] : string,
    )
```

Arguments

pnam
: name of the parameter you want to set or retrieve

pval
: value of the parameter you want to set; this value must be of the type appropriate to the parameter being set.

Description

The procedure `wmsetp` is used to set values for wmap parameters; the function `wmgetp` is used to retrieve the current value of the named parameter. `wmgetp` returns a value that is of the type appropriate to the parameter type.

Example

```
begin
;
```

```
;  Create an X11 workstation.
;

      wid = create "wmbarb_example" xWorkstationClass defaultapp
        "wkPause" : True
      end create
;
;  Draw wind barbs.
;
      x = (/0.25, 0.75, 0.75, 0.25/)
      y = (/0.25, 0.25, 0.75, 0.75/)
      u = (/50., -50., -50.,  50./)
      v = (/50.,  50., -50., -50./)
      wmsetp("wbs",0.2)
      wmbarb(wid, x, y, u, v)
      frame(wid)
;
;  Get parameter value.
;
      size = wmgetp("wbs")
      print(size)
end
```

---

wgt_runave

Calculate a weighted running average. Missing values are allowed

Note: This function is only available in version 4.1.1 of NCL. If your site is licensed for version 4.1, then you should receive version 4.1.1 automatically.

Synopsis

```
    function wgt_runave(
        x    : float,
        wgt  : float,
        kopt : integer
    )
```

Arguments

x

An array with one or more dimensions. The rightmost dimension will be the dimension on which the weighted running average is performed. Missing values should be indicated by *x*@_FillValue. If *x*@_FillValue is not set, then the NCL default (-999) will be assumed.

wgt

Vector containing all the weights

kopt

End-point option (*kopt* = 0 is the most frequently used)

In the following, N=last point in the series:

```
kopt < 0 : utilize cyclic conditions
           e.g.,  nwgt=3
           x(0)  = w(0)*x(N)   +w(1)*x(0)+w(2)*x(1)
           x(N)  = w(0)*x(N-1)+w(1)*x(N)+w(2)*x(0)
           e.g.,  nwgt=4 at n=1 and n=N
           x(0)  = w(0)*x(N-1)+w(1)*x(N)   +w(2)*x(0)+w(3)*x(1)
           x(N)  = w(0)*x(N-2)+w(1)*x(N-1)+w(2)*x(N)+w(3)*x(0)

kopt = 0 : set unsmoothed beginning and end pts to x@_FillValue [most common]
           e.g.,  nwgt=3 , x(0) = x@_FillValue and x(N) = x@_FillValue
                         , x(1) = w(0)*x(0)+w(1)*x(1)+w(2)*x(2)
           e.g.,  nwgt=4 , x(0),x(1),x(N-1) and x(N) = x@_FillValue
                         , x(2) = w(0)*x(0)+w(1)*x(1)+w(2)*x(2)+w(3)*x(3)

kopt > 0 : utilize reflective (symmetric) conditions
           e.g.,  nwgt=3
           x(0)  = w(0)*x(1)   +w(1)*x(0)+w(2)*x(1)
           x(N)  = w(0)*x(N-1)+w(1)*x(N)+w(2)*x(N-1)
           e.g.,  nwgt=4
           x(0)  = w(0)*x(2)   +w(1)*x(1)   +w(2)*x(0)+w(3)*x(1)
           x(N)  = w(0)*x(N-2)+w(1)*x(N-1)+w(2)*x(N)+w(3)*x(N-1)
```

Description

`wgt_runave` returns a float array of the same dimensionality as *x* with the last dimension smoothed. The running average is a special case of a filter where all weights are the same. The filter is applied to the *i*-th time of the requested series as follows:

F(i) = SUM{w(j)*UF(i-(nwgt/2)+j-1)} from j=0,nwgt-1

where *F* is the filtered field, *UF* is the unfiltered field, *w(j)* is the j-th filter weight, and *nwgt* is the number of weights.

With the proper choice of weights, this "filter" can also be used to compute time differences and derivatives.

If the number of weights is even, the filter's center falls between series elements. In this case, the center is shifted one-half of a time increment towards the latter element.

Example 1

Let *x* be dimensioned *nlat* x *mlon* x *ktimes* where *nlat*=64, *mlon*=128, *ktimes*=1000. Perform a 3 point (1-2-1) average and use *kopt*=0. Return the *nlat* x *mlon* smoothed series to the original *x* array:

```
x = wgt_runave (x, (/ 0.25, 0.50, 0.25 /), 0)
```

Example 2

Let *x* be dimensioned *ntimes* x *nlat* x *mlon* with named dimensions "time" , "lat" , "lon". (Assume the "time" dimension contains monthly data.) Use the 11-point Trenberth filter on the monthly data [Trenberth, K.E.: Mon. Wea. Review, February 1984]. The Trenberth filter effectively removes fluctuations with periods of less than 8 months but includes all others. At 24 months 80% of the variance is retained.

```
kopt = 0
wgt  = (/ 0.0270, 0.05856, 0.09030, 0.11742, 0.13567, \
          0.1421, 0.13567, 0.11742, 0.09030, 0.05856, 0.027 /)
y    = wgt_runave (x(lat|:,lon|:,time|:), wgt, kopt)
```

y will be a 3-dimensional array of length *nlat* x *mlon* x *time*.

Example 3

Let *x* be dimensioned *ntimes* x *klev* x *nlat* x *mlon* with named dimensions "time" , "lev" , "lat" , "lon". (Assume the "time" dimension contains twice daily data.) Use the low, medium and high pass 31-point Blackmon filters on the twice daily data. [Blackmon, M.B.: J . Atmos. Sci., August, 1976, p1609]. The low-pass filter allows most of the variance from fluctuations of 10 days or more; the medium pass filter allows variance of 2.5 to 6 days; and the high pass filter allows variance 1 to 2 days.

```
kopt  = 0
wlow  = (/-0.0059591606, -0.0074864031, -0.0081766107, -0.0074769889 \
        ,-0.0048789007,  0.0         ,  0.0073407153,  0.0170670479 \
        , 0.0288191067,  0.0419626250,  0.0556356004,  0.0688283154 \
        , 0.0804876892,  0.0896329731,  0.0954676702 \
        , 0.0974726419 \                              ; central wgt
        , 0.0954676702,  0.0896329731,  0.0804876892 \
        , 0.0688283154,  0.0556356004,  0.0419626250,  0.0288191067 \
        , 0.0170670479,  0.0073407153,  0.0         , -0.0048789007 \
        ,-0.0074769889,  -0.0081766107,-0.0074864031, -0.0059591606 /)
ylow  = wgt_runave (x(lev|:,lat|:,lon|:,time|:), wlow, kopt)

wmed  = (/-0.0030384857, -0.0001341773, -0.0096723016, -0.0191709641 \
        ,-0.0020017146,  0.0304306715,  0.0328072034,  0.0041075557 \
        , 0.0033466748,  0.0419335015,  0.0283041151, -0.0923257264 \
        ,-0.1947701551, -0.1020097578,  0.1433496840 \
        , 0.2776877534 \                              ; central weight
        , 0.1433496840, -0.1020097578, -0.1947701551 \
        ,-0.0923257264,  0.0283041151,  0.0419335015,  0.0033466748 \
        , 0.0041075557,  0.0328072034,  0.0304306715, -0.0020017146 \
        ,-0.0191709641, -0.0096723016, -0.0001341773, -0.0030384857 /)
ymed  = wgt_runave (x(lev|:,lat|:,lon|:,time|:), wmed, kopt)

whigh = (/ 0.0036193743,  0.0062672457, -0.0071490567, -0.0089978990 \
        , 0.0125704103,  0.0117924147, -0.0207251125, -0.0144542141 \
        , 0.0333056699,  0.0167834343, -0.0546750050, -0.0185972473 \
        , 0.1009810886,  0.0197489990, -0.3186000638 \
        , 0.4762599236 \  ; central weight
        ,-0.3186000638,  0.0197489990,  0.1009810886 \
```

```
          ,-0.0185972473, -0.0546750050,  0.0167834343,  0.0333056699 \
          ,-0.0144542141, -0.0207251125,  0.0117924147,  0.0125704103 \
          ,-0.0089978990, -0.0071490567,  0.0062672457,  0.0036193743 /)
   yhigh = wgt_runave (x(lev|:,lat|:,lon|:,time|:), whigh, kopt)
```

y will be a 4-dimensional array of length *klev* x *nlat* x *mlon* x *time*.

In the above example it might be better to create a new variable *xnew* rather than reorder the dimensions each time. For example, if you are going to apply this filter for low, medium, and high filters, it will be more efficient to create a new variable *tnew*:

```
  tnew = t(lev|:,lat|:,lon|:,time|:)
```

and then use *tnew* in the calls (be sure to delete *tnew* after finishing).

Example 4

Let *x* be dimensioned *ntimes* x *klev* x *nlat* x *mlon*. Perform a 5 point running average use the cyclic option in the longitude direction:

```
     kopt = -1
     wgt  = (/ 1., 3., 5., 3., 1. /)
     wgt  = wgt/sum(wgt)              ; normalize
     x    = wgt_runave (x,wgt, kopt)
```

Source code NCL functions and procedures

All of the following functions are written in NCL source code. To use these you must download the NCL script and use the *load* command to access them.

General NCL functions and procedures

Procedures

- copy_filevaratts - copies file variable attributes to a variable
- copy_filevarcoords - copies file variable coordinate variables to a variable
- copy_filevardims - copies file variable dimensions to a variable
- copy_varcoords - copies coordinate variables from one variable to another
- copy_varatts - copies variable attributes from one variable to another
- copy_vardims - copies variable dimension names from one variable to another

Functions

NCL math functions

Functions

- sm11tr - 11 point Trenbreth filter.
- smooth92d - 9 point 2D smoothing function
- vor - computes kinetic vorticity

Plotting functions and procedures

Functions

- hsv2rgb - converts from HSV color model to RGB color model.

Procedures

- bigger - makes a plot bigger
- centerat - centers a plot at a given point
- map_corners - returns the lat/lon coordinate of the corners of a map projection
- mapzoom - given a lat/lon center point, zooms in on that point
- move - moves a plot by NDC offsets
- moveto - moves a plot to a specific location in NDC units.
- setaspect - sets the aspect ratio of a plot
- plotcolors - plots the current color table
- zoom - zooms in on a plot

NCL source code functions and procedures

bigger.ncl

Contains a single procedure to make plots bigger.

Source: bigger.ncl

Synopsis

```
procedure bigger (
        plot[1]:graphic,
        frac[1]:numeric
)
```

Arguments

plot
: A reference to an HLU plot object

frac

A numeric value given in %

Description

Makes a plot bigger by a given fraction.

---

centerat.ncl

Contains a single procedure that will center a plot at a give NDC point.

Source: centerat.ncl

Synopsis

```
procedure centerat (
        plot[1]:graphic,
        x[1]:numeric,
        y[1]:numeric
)
```

Arguments

plot
: A reference to an HLU plot object

x
: x coordinate of center point in NDC units

y
: y coordinate of center point in NDC units

Description

Resets a plots view port so it is centered at the input point (/*x*,*y*/)

---

copy.ncl

Contains 6 procedures for copying attributes, dimension names and coordinate variables from variables to variables and from file variables to variables

---

Source: copy.ncl

Synopsis

```
procedure copy_varcoords (
        var_from ,
        var_to
)
```

Arguments

var_from
: Variable from which coordinates will be copied.

var_to
: Variable to which coordinates will be copied.

Description

Copies all the dimension names and ascotiated coordinate variables from *var_from* to *var_to* if and only if the dimension sizes match.

Source: copy.ncl

Synopsis

```
procedure copy_varatts (
        var_from,
        var_to
)
```

Arguments

var_from
: Variable from which attributes will be copied

var_to
: Variable to which attributes will be copied

Description

Copies all the atttributes from *var_from* to *var_to*

Source: copy.ncl

Synopsis

```
procedure copy_vardims (
        var_from,
        var_to
)
```

Arguments

var_from
    Variable to copy dimension names from
var_to
    Variable to copy dimension names to

Description

Copies dimension names from one variable to another. The variables must have the same number of dimensions.

Source: copy.ncl

Synopsis

```
procedure copy_filevaratts (
        fromfile[1]:file,
        fromvar[1]:string,
        var_to
)
```

Arguments

fromfile
    A reference to the file containing the variable *fromvar*
fromvar
    The string name of the variable from which the attributes will be read
var_to
    The variable towhich the attributes will be copied

Description

Reads attributes of a file variable and assigns them to *var_to*

Source: copy.ncl

Synopsis

```
procedure copy_filevarcoords (
        fromfile[1]:file,
        fromvar[1]:string,
        var_to
)
```

Arguments

fromfile
    A reference to the file containing the variable *fromvar*
fromvar
    The string name of the variable from which the coordinate variables will be read
var_to
    The variable to which the coordinate variables will be assigned

Description

Copies coordinate variables from the variable *fromvar* in the file *fromfile* to *var_to*. If the dimension sizes or number of dimensions don't match then an error is printed and no changes are made to *var_to*.

Source: copy.ncl

Synopsis

```
procedure copy_filevardims (
        fromfile[1]:file,
        fromvar[1]:string,
        var_to
)
```

Arguments

fromfile
    A reference to the file containing the variable *fromvar*
fromvar
    The string name of the variable from which the dimension names will be read
var_to
    The variable to which the dimension names will be assigned

Description

Copies dimension names from the variable *fromvar* in the file *fromfile* to *var_to*. If the number of dimensions don't match then an error is printed and no changes are made to *var_to*.

---

hsv2rgb.ncl

Converts from the HSV color model to the RGB color model required by the HLU workstation object

---

Source: hsv2rgb.ncl

Synopsis

```
function hsv2rgb (
        h[*]:float,
        v[*]:float,
        s[*]:float
)
```

Arguments

h
: A singly dimensioned array of values from [0.0..360.0]

v
: A singly dimensioned array the same size as *h* with values from [0.0..1.0]

s
: A singly dimensioned array the same size as *h* with values from [0.0..1.0]

Description

This function maps values from the HSV color model to the RGB color model. HSV is a good model for generating smooth color maps. See (Computer Graphics: Principles and Practice by Foley). The return value is a 2 dimensional array of rgb color triplets. The return value from this function can be directly assigned to the "wkColorMap" resource of an HLU workstation object.

---

map_corners.ncl

Contains a single function for returning the lat/lon coordinates at the corners of the viewport.

Source: map_corners.ncl

Synopsis

```
function  map_corners (
        mp[1]:graphic
)
```

Arguments

mp
: A reference to an HLU mapPlotClass object

Description

Returns the lat/lon coordinates of the map plot at the corners of the viewport. This function only works when "mpLimitMode" is not equal to "MAXIMALAREA".

mapzoom.ncl

Contains a single procedure that zooms in on a specific lat/lon location.

Source: mapzoom.ncl

Synopsis

```
procedure mapzoom(
        mp[1]:graphic,
        lat[1]:float,
        lon[1]:float,
        scale[1]:integer
)
```

Arguments

mp
: A reference to an HLU mapPlotClass object

lat
: Lattitude about which the map will be centered

lon
: Longitude about which the map will be centered

scale
: Number of times to zoom in (i.e. 2 means 2X zoom)

Description

Configures the mapPlotClass object *mp* such that it is centered at (*lat*,*lon*) and is zoomed in by *scale*. A value of 2 for *scale* means a 2X zoom is applied.

move.ncl

Contains a single routine to move a plot by a x and y NDC value

Source: move.ncl

Synopsis

```
procedure move (
        plot[1]:graphic,
        dx[1]:float,
        dy[1]:float
)
```

Arguments

plot
: A reference to an HLU plot object

dx
: A offset in NDC units to move the plot in the x direction

dy
: A offset in NDC units to move the plot in the y direction

Description

Given NDC offset, this procedure moves the input plot by the offset (*dx*,*dy*).

---

moveto.ncl

Contains a single routine to move a plot to a specific location in NDC units.

Source: moveto.ncl

Synopsis

```
procedure moveto (
        plot[1]:graphic,
        vpx[1]:float,
        vpy[1]:float
)
```

Arguments

plot
: A reference to an HLU plot object

vpx
: New x coordinate of upper left corner of plot in NDC units

vpy
: New y coordinate of upper left corner of plot in NDC units

Description

Assigns *vpx* and *vpy* to the resources "vpXF" and "vpYF" respectively

---

plot_utils.ncl

Source: plot_utils.ncl

Synopsis

```
procedure FUNCNAME (
)
```

Arguments

frac
: A numeric value given in %

Description

Makes a plot bigger by a given fraction.

---

plotcolors.ncl

Source: plotcolors.ncl

Synopsis

```
procedure FUNCNAME (
)
```

Arguments

frac
: A numeric value given in %

Description

Makes a plot bigger by a given fraction.

---

save.ncl

Source: save.ncl

Synopsis

```
procedure FUNCNAME (
```

```
)
```

Arguments

frac
: A numeric value given in %

Description

Makes a plot bigger by a given fraction.

---

setaspect.ncl

Source: setaspect.ncl

Synopsis

```
procedure FUNCNAME (
)
```

Arguments

frac
: A numeric value given in %

Description

Makes a plot bigger by a given fraction.

---

sm11tr.ncl

Source: sm11tr.ncl

Synopsis

```
procedure FUNCNAME (
)
```

Arguments

frac

A numeric value given in %

Description

Makes a plot bigger by a given fraction.

---

smaller.ncl

Source: smaller.ncl

```
procedure FUNCNAME (
)
```

Arguments

frac
A numeric value given in %

Description

Makes a plot bigger by a given fraction.

---

smooth92d.ncl

Source: smooth92d.ncl

```
procedure FUNCNAME (
)
```

Arguments

frac
A numeric value given in %

Description

Makes a plot bigger by a given fraction.

---

vor.ncl

Source: vor.ncl

Synopsis

```
procedure FUNCNAME (
)
```

Arguments

frac
: A numeric value given in %

Description

Makes a plot bigger by a given fraction.

zoom.ncl

Source: zoom.ncl

Synopsis

```
procedure FUNCNAME (
)
```

Arguments

frac
: A numeric value given in %

Description

Makes a plot bigger by a given fraction.

Information on supported data formats

This page provides more detailed information about each of the supported data formats in NCL. Information about referencing files and file variables is provided in the Files and file variables section of the reference guide. Everything discussed in that section applies to all supported data formats regardless of type. The discussion here focuses on specific conventions and limitations of each format and its NCL implementation.

NCL's supported data formats are:

- netCDF - Network Common Data Format (**.nc** and **.cdf** extensions)
- GRIB - Grids In Binary (**.grb** extension)
- HDF - Hierarchical Data Format - Scientific Data Sets (SDS) only (**.hdf** extension)
- CCM - Community Climate Model History Tape Format (**.ccm** extension)

netCDF - network common data format

Online documentation for netCDF is available at http://www.unidata.ucar.edu/packages/netcdf/index.html.

Nearly all the netCDF features are supported by NCL because NCL's data model is patterned after netCDF. The procedure filedimdef can be used to create file dimensions that are unlimited, and the procedure filevardef can be used to pre-define variables.

Strings are not a supported netCDF type. NCL maps all character attributes into type string for convenience. The conversion function stringtochar must be used to write string data into netCDF files. Also errors can be generated if an attempt to write a longer string to an existing attribute is attempted.

Also since netCDF does not support 64-bit integers, these are not supported either.

Global file attributes can be written to netCDF files by assigning attributes to the variable that references the file, in the same way that attributes are assigned to normal NCL variables. Doing this causes NCL to write the attribute into the file.

GRIB - Grids In Binary

Online documentation for GRIB is available at ftp://nic.fb4.noaa.gov/pub/nws/nmc/docs/gribed1.

Support for GRIB from NCL was added with release 4.0.1. It is mostly complete with a few exceptions covered here.

GRIB is very problematic file format. There is no official API for writing GRIB, so **very often** GRIB files are written incorrectly. This can cause minor to severe problems when reading these data with NCL. NCL has been tested on NCEP, FSL, and ECMWF GRIB data. ECMWF data to date have caused the most problems. ECMWF changes their parameter table regularly and without notice. This can cause some parameters to be mislabeled in NCL. If using ECMWF data, please contact them for information on their data. NCEP and FSL data typically do not cause NCL to behave badly. Data from other centers have been tested, but a comprehensive list is not available. Quite frequently though data from other centers have contained problems in the GDS and PDS sections. If data from other centers use an extended parameter table, NCL may not be able to correctly provide names and unit information for all the variables in the file. If having problems reading GRIB files, please contact ncargfx@ncar.ucar.edu

First and most important, GRIB is a **read only** format. GRIB files cannot be created using NCL. The main reason for this is that GRIB files must contain the ids of the meteorological center writing the data and the model used to generate the data. Since NCL is not a model and could be used at locations that don't have a center id, NCL has been designed to only read GRIB data.

Another important item to understand about GRIB with respect to reading GRIB from within NCL is that GRIB often stores grids in various projection and computational grids that do not adhere to NCL's coordinate variable constraints. For example, one common GRIB grid type is a rectilinear grid of points sampled from a stereographic projection, commonly referred to as a tangent grid. In this situation, a single monotonic coordinate variable is not possible because the coordinates of each grid point are defined to be a function of the index (i.e. lat = f(i,j) and lon = g(i,j)). NCL handles this by providing two 2D arrays for grids like this. One array contains all of the lat values for every index, and the other contains all the lon values.

NCL presents GRIB variables as having up to five dimensions. The dimensions are ordered [*initial_time*]x[*forecast_time*]x[*level*]x[*gridx*]x[*gridy*]. GRIB files contain single 2D slabs of data with headers that state the initial time, forecast time, and level to which the slab belongs. NCL, when opening a GRIB file, scans through all the records in the file and sorts them by variable type, initial time, forecast time, level type, and finally grid type. This allows NCL to present GRIB data in a fashion analogous to netCDF. If only one record for a specific variable is in the file, the dimension of the variable will only be [*gridx*]x[*gridy*].

Since GRIB files can have the same variable written with different grids, different time range indicators, and different pressure level indicators, the NCL implementation has to use some unique naming conventions for variables, dimension, and grids so that there would be unique variable names. For example, consider the variable TMP (temperature). One GRIB file could contain the variable with many different variations. One record could be the average temperature, another could be the difference in temperature from one time to the next, and yet another could be the temperature at tropopause. Clearly these are each different variables in the file, but GRIB identifies them as TMP. Therefore, NCL has conventions for distinguishing these different variables.

Before continuing with NCL naming conventions, note that the NCAR Graphics distribution comes with a sample GRIB file. The following NCL statement will access this sample file if your site administrator has installed the data.

```
gribfile = addfile(ncargpath("data") + "/grb/ced1.lf00.t00z.eta.grb","r")
```

Further discussions of NCL naming conventions will refer to variables in this sample data file.

The GRIB file variable names are constructed from information in the GRIB record's "product description section." The first portion of any variable name is its abbreviation taken directly from NCEP's GRIB specification document. If the variable is unrecognized -- which could happen because centers often change their GRIB files without telling anyone -- the variable is given a name "VAR" plus its variable identifier number. The variable name is followed by an underscore ("_") followed by a number that represents the GRIB grid number the data are written in. After this, there is another underscore followed by an abbreviation of the type of level coordinates the data exist in. After this, there is an optional string that communicates what type of time range indicator is used with the data. Just as with the variable name, if the time indicator is unrecognized, then the time range indicator number is concatenated to the end of the variable name.

The best way to put this all together is to look at some examples from the file above. The following are the temperature variables defined in this GRIB file. There are six different temperature variables. In each case the variable name starts with its abbreviation from the GRIB document TABLE 2, in this case "TMP". This set of variables is defined over two grids, grid 6 is a 2385-point 53x45 N. Hemisphere polar stereographic grid, and grid 101 is a 10283-point 113x91 N. Hemisphere polar stereographic grid. The final strings here are the level abbreviations also taken from the GRIB document in TABLE 3 or TABLE 3a. "TRO" stands for "tropopause level," "ISBL" stands for "isobaric level," "GPML" stands for "fixed height level," "SIGL" stands for "sigma level," and "SIGY" stands for "layer between two sigma levels."

```
gribfile->TMP_6_TRO
gribfile->TMP_6_ISBL
gribfile->TMP_6_GPML
gribfile->TMP_6_SIGL
gribfile->TMP_6_SIGY
gribfile->TMP_101_SIGL
```

Each variable also has attributes defined from the GRIB product description set. These typically are "center," "units," and a "long_name." None of these variables has a special time range indicator, meaning each value is a "snapshot" of the data at the given time. In the example data file, only a few variables have a time range indicator; one variable is named "A_PCP_105_SFC_acc." This variable references total precipitation on grid 105 at the surface. The "_acc" component implies that each value in the data is an accumulation. Currently in NCL only "_ave" for average, "_acc" for accumulation, and "_dif" for difference have string values; any other time range indicators have an underscore followed by the time range indicator number from the product description section of the variable's GRIB record.

Dimension names and coordinate variables also have a naming convention. The most important naming convention is how the grid coordinate variables and dimensions are named. If a grid is defined on a type of grid where the coordinates for the grid points can be represented with two monotonic coordinate variables, the dimension and grid coordinate variables are named the same. These kinds of grids include Mercator and Cylindrical Equidistant lat/lon grids.

Dimension names are named "lat" and "lon" with an underscore followed by the GRIB grid number and can be used as normal NCL coordinate variables. Grid types that do not fit into the NCL coordinate variable convention are named differently than their respective dimensions. In this second case, the dimension is named "gridx" and "gridy" followed by the GRIB grid number. The coordinates for each grid point are provided by NCL as variables. The are two 2D arrays containing either all the latitude points or all the longitude points for a specific grid type. These variables are named "gridlat" and "gridlon" followed by an underscore and the GRIB grid number. These variables have attributes that describe the type of grid and the coordinates at the corners of the grid. The following is an excerpt from the print of the GRIB file "/grb/ced1.lf00.t00z.eta.grb" showing what these coordinate variables look like.

```
Variable: gribfile (file variable)

filename:       ced1.lf00.t00z.eta
path:   /cyclone/ncargd/dev/IRIS_IRIX_5_2_/lib/ncarg/data/grb/ced1.lf00.t00z.eta
   file global attributes:
   dimensions:
      gridx_6 = 45
```

```
        gridy_6 = 53

    ...

        float TMP_6_ISBL ( lv_ISBL7, gridx_6, gridy_6 )
           forecast_time :        0
           initial_time : 10/24/1995 (00:00)
           center :       US Weather Service - National Met. Center
           _FillValue :   -999
           units :        K
           long_name :    Temperature

    ...

        integer lv_ISBL7 ( lv_ISBL7 )

    ...
        float gridlat_6 ( gridx_6, gridy_6 )
           corners :
           grid_description :     2385-point (45x53) N. Hemisphere polar stereogra
           units :        degrees_north
           long_name :    latitude

        float gridlon_6 ( gridx_6, gridy_6 )
           corners :
           grid_description :     2385-point (45x53) N. Hemisphere polar stereogra
           units :        degrees_east
           long_name :    longitude
```

Level coordinate variables and dimensions begin with the prefix "lv_" followed by the level abbreviation from the GRIB document followed by the dimension number. The dimension number is needed because often the same coordinate variable with the same units will exist for different variables with different values. The addition of the dimension number following the dimension name creates a unique dimension and coordinate variable combination.

The forecast_time coordinate is in hours, and the initial_time coordinate variable is a string with the month, day, year, hour, and minute of the initial time in the following format: MM/DD/YYYY (HH:MM).

HDF - Hierarchical Data Format - Scientific Data Sets (SDS) only

Online documentation for HDF is available at http://hdf.ncsa.uiuc.edu/.

Note: NCL currently does not handle HDF-EOS data. Support for the conventions used in HDF-EOS data is under development

HDF is somewhat limited from NCL. NCL only reads data written using the SDS interface. Importing HDF files is entirely analogous to importing netCDF files as described above, with some minor exceptions. First, if variable names contain spaces or non-alphanumeric characters, these characters are replaced with the '_' character when listed from NCL and are referenced from NCL in this fashion.

Writing variables to HDF files from NCL is somewhat limited currently. Only the values of variables and their names are written to files in the HDF format. Attributes and dimension names cannot currently be written from NCL. Attributes and dimension names will generate warning messages when attempting to write an NCL variable to an HDF file.

There is currently no way to access 8-bit and 24-bit HDF images from NCL. There is also no way to access VGROUP or VDATA HDF data classes.

CCM - Community Climate Model History Tape Format

The CCM format is a format, originally in CRAY COS blocked form, written by the NCAR CCM1, CCM2, and CCM3 global climate models. It is also possible to have IEEE CCM files. Currently, NCL does not support IEEE CCM files due to lack of documentation. It is possible to use the public domain tool called "ccm2nc" (available on almost all SCD computers; "man ccm2nc") to convert these files to netCDF. NCL can then reference the netCDF file(s). If not on SCD machines then the "ccm2nc" software can be downloaded from http://neit.cgd.ucar.edu/cms/ccm3/tools/

CCM files are pretty straightforward (no special naming convention is needed as with GRIB files); the variable names and unit information are stored as character data in the CCM files. When a CCM file is opened, NCL scans the file and creates an index of all the data in the files. This can be expensive for large files, but it facilitates quickly accessing individual variables of the file. Because this can be expensive, you should avoid repeatedly calling addfile on the same file whenever possible.

For more information on the CCM model and CCM file format, see the CCM3 User's Guide.

Creating Graphical Output

NCL uses the NCAR Graphics High Level Utilities Library (HLU) to produce its graphical output. Complete documentation on the HLUs is available from the HLU User Guide and the HLU Reference Guide. In addition, NCL source code examples using the HLU library can be obtained from the Quick Start Guide Examples documentation.

This section of the NCL reference guide covers some basics of using the HLU libraries and NCL's interface to the HLUs. This is not an exhaustive manual for the HLUs. This section is intended to be a basic overview of useful features.

- HLU Basics
- Changing a plot's size
- Transformations
- Changing a plot's data
- Modifying a plot
    - Contour Plots
    - Xy Plots
    - Vector Plots
    - Streamline Plots
    - Legends, Label Bars, TickMarks, and Titles
- Using maps

- Using Resource Files for plot defaults
- Using annotations
- Drawing lines and polygons

HLU Basics

Plot creation can get very complex depending on how you want to look at your data. NCL's interface for creating graphics is called the High Level Utilities (HLUs). There are five basic steps needed to create a simple plot. Issues such as combining plots, multiple plot frames, adding maps, and annotations are covered later.

- Open a workstation

  This is probably the simplest decision to make. X11 output will create an X11 window on the display specified by the DISPLAY environment variable. NCGM is a compact, efficient, and portable vector description of the graphical output. NCGM can be translated and viewed using the application ctrans and animated using the idt application. NCGM is recommended for the efficient storing of NCAR Graphics-generated output. PostScript can be output directly. The PostScript generated by the HLUs is encapsulated so it can be directly imported into documents or PostScript-compatible software and hardware devices. The workstation is responsible for managing all the colors used by plots. Each workstation has "wkColorMap" resources that can be set to a nx3 array of red, green, and blue colors. In addition, several default color maps can be set by using their names instead of an array of color combinations.

  The following are the Class names needed to create each of the output types using the *create* statement:

  - xWorkstationClass - X11 output
  - ncgmWorkstationClass - NCGM output
  - psWorkstationClass - PostScript output

- Describe your data

  There are HLU objects whose primary purposes are to describe the data in a flexible fashion. The HLU plot object needs to know what the dimensionality of the data is, what the coordinates of the data are, whether the data contains missing values, and finally what data type the data is. If it weren't for these data objects, you'd have to convert your data into a common form before visualizing it. The data objects provide the flexibility to represent your data in the fashion it is represented. There are three basic types of data objects. There is the scalarFieldClass that describes scalar fields which are used by the contourPlotClass. There is the coordArraysClass which is used by the xyPlotClass, and finally there is the vectorFieldClass which is used by the vectorPlotClass and the streamlinePlotClass. A data object must be defined before the next step.

- Create a plot object and add your data

  Producing plots using the HLUs is a different a paradigm than most people are used to. In NCL, you create an instance of a plot which can then be drawn and/or manipulated, rather than setting up a single graphics state or calling a single procedural interface with a lot of parameters. HLU plots exist and can be drawn independently from each other. This provides a great deal of flexibility, but can be a little confusing for beginners. Just as you create workstations and data objects, you can create plot objects. There are many types of plots which are listed in the HLU reference manual

- Draw the plot
  The draw procedure is called to draw a plot. Once a plot is created, draw can be called at any time, a feature unique to NCL.

- Call the frame command
  Once a plot has been drawn to an output frame and no other plots are going to be drawn on the same frame, a call to the frame procedure is needed. This call, when using either Postscript or CGM workstations, causes a new page to be inserted into the output file. If X11 output is being used, a frame call will wait for a button click (unless "wkPause" is set to False) and then clear the screen.

The following is a short script demonstrating these steps. The available plot types are listed in the HLU reference manual. The following creates a simple filled contour plot from a data file included with the NCAR Graphics release. The following script should run without modification on your system if the NCAR Graphics data directory was installed.

```
begin
;
; Step 1)
;
; Open an X11 output window
;
        xwks = create "xwks" xWorkstationClass defaultapp
                "wkColorMap" : "temp1"
        end create

;
; Step 2)
;
; Open the data file and describe the data
;

        data_file = addfile(ncargpath("data") + "/cdf/Pstorm.cdf","r")

;
; Describe data by creating a scalarField object
;

        data = create "data" scalarFieldClass defaultapp
;
; Select the first time step
;
                "sfDataArray" : data_file->p(0,:,:)
                "sfMissingValueV" : data_file->p@_FillValue
                "sfXArray" : data_file->p&lon
                "sfYArray" : data_file->p&lat
        end create

;
; Step 3)
;
; Create a contourPlot with the above data
;
```

```
        contour = create "contour" contourPlotClass xwks
                "cnScalarFieldData" : data
                "cnFillOn" : True
        end create

;
; Step 4)
;
; Call draw
;
        draw(contour)

;
; Step 4)
;
; Call frame
;

        frame(xwks)

end
```

Setting the size of a plot

The resources ***vpXF, vpYF, vpWidthF,i*** and ***vpHeightF*** are used to specify the size of the output plot. The values of these resources can be set to values between 0.0 and 1.0. The output frame, regardless of which output device is open, has coordinates called Normalized Device Coordinates (NDC). These coordinates range from 0.0 to 1.0 along each axis of the output frame. The *setvalues* statement is used to assign values to each of these resources. Every object that can be drawn accepts these four resources for setting size.

Transformations

The transformation is defined as how the data coordinates are mapped to the screen. Contour plots, vector plots, and streamline plots all work the same way, xy plots are slightly different.

Conceptually you must think of two transformations. One is the natural transformation of the data which is defined by the data's coordinates. The second is the view transformation (how you want to look at the data). If the data's coordinates are evenly spaced, then the data's natural transformation is a Linear uniformly spaced transformation. In order to view the data in a different transformation you must create an additional plot, either a mapPlot, an irregularPlot, or a logLinearPlot and call the overlay procedure. This will allow you to select how the data are displayed.

A note about IrregularPlots. These are NOT randomly spaced plots, they are plots where the coordinates for each axis are not evenly spaced, but are still rectilinear, meaning all adjacent points along the X axis have the same Y coordinate and vice versa. This type of data's natural transformation is irregular; when the data is viewed, it is neither a linear projection nor a log projection. The tick marks will be irregularly spaced.

So to sum it up for contour plots and similar plots, if no coordinate values are set for the plot's scalar field, the transformation will be integer grid coordinates which represent the integer indexes of each grid point. If coordinates are set using the ***sfYCStartV, sfYCEndV, sfXCStartV,*** and ***sfXCEndV,*** then the HLUs assume that the grid points are equally spaced starting at the beginning and end of each coordinate range. In this case the default view will be use linear axis to display the contour. The resources ***sfYArray*** and ***sfXArray*** are used if the grid points are not equally spaced (this happens with the latitude coordinate for Gaussian grids). In this situation the output display is not a linear axis, it is called an irregular axis. This transformation essentially displays the grid with each point being equally spaced but marks the axis by the function defined by the sequence of points in the coordinate arrays. This type of transformation is especially useful for coordinate systems which are neither log base 10 nor linear. Example cn03 demonstrates this type of transformation.

Changing the data of a plot

Since the HLUs are object oriented, it is very simple to change the data for a plot. Changing the data does not require recreating plots. The *setvalues* statement is used to reset the data array in the plot's data object. For example, consider changing the above plot from using the first time step to using the second. The following is all that is needed to update the plot and redraw it:

```
setvalues data
        "sfDataArray" : data_file->p(1,:,:)
end setvalues
draw(contour)
frame(xwks)
```

That's it. No need to do anything more. The contour plot is automatically informed that the data have changed, and it updates itself.

In XyPlot, it is possible to have multiple curves and data objects in a single plot. Because of this, a special function to add new data and a special procedure to remove old data can be used without affecting the data already in the plot. These routines are NhlAddData and NhlRemoveData. The return value from NhlAddData is called a *Data Specific Object*. It is used to allow individual customization of how a piece of data is displayed. For instance, it is used in the XyPlot to send line styles, labeling, and colors for each data object; see the XyPlot section on modifying a plot.

Modifying a plot

The setvalues statement can be used to modify a plot once it has been created. Modifications could be anything from resizing, to changing the data, to changing the colors.

Contour plots

The following are frequently used resources for configuring the ContourPlot object. For a full list of these resources and more descriptions, see the HLU reference guide section for ContourPlot.

- "cnScalarFieldData": This resource must be set to create a contour plot. It accepts a single reference to a ScalarField object. The scalar field describes the data, their coordinates, and missing values to the contour object.
- "cnLevelSelectionMode": This resource determines how the contour intervals will be chosen. There are four options: The default is "AutomaticLevels" which means each time the

"cnScalarField" resource is set, the contour plot will choose "nice" values for the contour intervals. "ManualLevels" allows you to set the minimum, maximum, and spacing for the intervals through the resources "cnMaxLevelValF", "cnMinLevelValF", and "cnLevelSpacingF". If these are set, then the contour plot will not reset itself when new data are set. "ExplicitLevels" allows you to set each interval to a specific value using the "cnLevels" resource. Finally, the option "EqualSpacedLevels" simply divides up the range of data into equally spaced levels.

- "cnFillOn": Causes the contour plot to draw filled contours when set to "True". The colors for the filled contours are determined by "cnFillColors".
- "cnRasterModeOn": When set to "True" causes the contour plot to fill each grid point with a color rather than interpolate contour areas. This is very useful if the data being viewed are categorical data, meaning they do not represent a function. Examples of this type of data could be vegetation types, demographic data, etc. This mode is also **much faster** for large grids.
- "cnLinesOn": Turns the contour lines on and off.
- "cnLineLabelsOn": Turns the contour line labels on and off. Sometimes drawing the contour line labels can be expensive, so turning them off can improve performance.

Xy plots

The following are frequently used resources for configuring the XyPlot object. For a full list of these resources and more descriptions, see the HLU reference guide section for XyPlot. The most important thing to know about the XyPlot that makes it different from other plots is that it accepts multiple data objects. The data objects can be added and removed at will. Each data object can represent one or more curves. Since these can be added and deleted, it is difficult to specify colors and curve-specific attributes, like line styles and labels, in the XyPlot itself. Because of this the XyPlot uses a special data specific object for configuring colors and line styles that is associated with each of the XyPlot's data objects. In this way it is not confusing as to which color belongs to which piece of data.

- "xyCoordData": XyPlot accepts instances of the CoordArrays object. These can be set either using setvalues, create, or by using NhlAddData to add individual pieces of data to the plot.
- "xyCoordDataSpec": Used to retrieve the XyPlot's data-specific objects at any time. You can use getvalues and this resource to retrieve all of the XyPlot's current data specific objects.
- "xyXStyle" and "xyYStyle": These resources control how each axis of the xyPlot is viewed. The valid options are: LOG, LINEAR, and IRREGULAR. If IRREGULAR is used, then "xyXIrregularPoints" or "xyYIrregularPoints" must be specified. The default is LINEAR.

The following are resources that apply to the XyPlot data specific object, objects referenced by the resource "xyCoordDataSpec":

- "xyLineColors": Sets the line colors for each curve represented by the corresponding CoordArrays object.
- "xyDashPatterns": Sets the dash patterns for each curve represented by the corresponding CoordArrays object. The values 1-16 are valid patterns.
- "xyMarkLineModes": Sets whether markers, lines, or both should be used when drawing the curves represented by the corresponding CoordArrays object. The options are "Lines", "Markers", or "MarkLines".
- "xyMarkers": If xyMarkLineModes is set to "Markers", then this resource is used to select which marker will be drawn for each curve. A list of markers appears on the Workstation reference page.
- "xyLabelModes": Lines can be labeled with either custom labels or letters. This resource

determines how each line is drawn. The valid options are "noLabels", "lettered", and "custom". If "custom" is selected, then the resource "xyExplicitLabels" is used to provide the strings for the custom labels.
- "xyLineThicknesses": Changes the thickness of each curve represented by the corresponding CoordArrays object.

Vector plots

The following are frequently used resources for configuring the VectorPlot object. For a full list of these resources and more descriptions, see the HLU reference guide section for VectorPlot. VectorPlot can display vectors in a variety of fashions. There are line vectors, filled vectors, and wind barbs. A variety of "looks" can be obtained by adjusting arrow head and tail sizes as well as shape.

- "vcVectorFieldData": This resource is used to assign an instance of a VectorField object to the VectorPlot.
- "vcScalarFieldData": Optionally, vectors can be colored by an additional scalar value. This resource is used to assign the ScalarField. To display the plot using these data, you must set "vcUseScalarArray" to True.
- "vcGlyphStyle": Selects the style of vector. "LineArrow", "FillArrow", and "WindBarb" are the options.
- "vcRefMagnitudeF" and "vcRefLengthF": These resources are key resources for setting the sizes of output vectors when using "LineArrow" and "FillArrow" glyph styles. "vcRefMagnitudeF" is a data value that represents what is considered a reference vector. If not set, the maximum vector magnitude is used as the reference vector. The resource "vcRefLengthF" sets the size vector for this reference value. All other vectors are scaled relative to the value of "vcRefMagnitudeF". So if "vcRefMagnitudeF" is the maximum magnitude, all vectors will be smaller. If "vcRefMagnitudeF" is a mid-range value, then some vectors will be bigger and some smaller. If "WindBarb" glyphs are used, then "vcRefLengthF" sets the length of all the windbarbs.
- "vcMinFracLengthF": This resource can be used to assure that vectors have a visible minimum size. Sometimes vectors become too small to see, especially if the magnitudes vary greatly, this resource establishes a minimum size.
- "vcLevelSelectionMode": This resource is used when the "vcScalarFieldData" resource is set. It specifies how the levels for the input scalar field should be determined and colored. This resource works just like ContourPlot’s "cnLevelSelectionMode".

Streamline plots

The following are frequently used resources for configuring the StreamLinePlot object. For a full list of these resources and more descriptions, see the HLU reference guide section for StreamLinePlot.

- "stVectorFieldData": This resource is used to assign an instance of a VectorField object to the StreamLine plot.
- "stArrowLengthF": sets the size of the streamline arrows.
- "stMinArrowSpacingF": Sets a minimum spacing for arrows along a single streamline.
- "stMinLineSpacingF": Sets the minimum distance that a streamline in progress is allowed to approach existing streamlines before being terminated.

Legends, Label Bars, TickMarks, and Titles

Legends, Label Bars, TickMarks, and Titles are special types of annotations that are controlled by an object called a PlotManager. ContourPlot, VectorPlot, StreamLinePlot, XyPlot, and MapPlot are all PlotManagers. This means that they can accept overlayed plots, they accept PlotManager resources, and finally that they manage Legends, Label Bars, TickMarks, and Titles. When a PlotManager is "managing" one of these types of annotations, it means the user does not have to create and configure these objects. For example, when ContourPlot displays a LabelBar, it configures each of the LabelBar's filled boxes and sets the valid label resources. If users were to create a LabelBar manually, they'd have to set all the applicable LabelBar resources, including its position. The PlotManager takes care of these details. Also when an annotation like LabelBar is managed by a PlotManager, the LabelBar's resources can be directly set through the managing object (ContourPlot, XyPlot...) rather than through the LabelBar object itself.

The following resources are PlotManager resources that control the display of Legends, LabelBars, TickMarks, and Titles:

- "pmLabelBarDisplayMode": Turns on and off the plot's LabelBar. Valid options are "NoCreate", "Never", "Always", and "Conditional". "Always" and "Never" are self explanatory. "NoCreate" can be used when creating a new plot to prevent the creation of a LabelBar. Otherwise a LabelBar object is created but just not drawn unless "Always" is set. "Conditional" leaves the display of the LabelBar up to the PlotManager and depends on the plots overlayed on each other.
- "pmTickMarkDisplayMode": Options are the same as pmLabelBarDisplayMode.
- "pmTitleDisplayMode": Options are the same as pmLabelBarDisplayMode.
- "pmLegendDisplayMode": Options are the same as pmLabelBarDisplayMode.
- "pmLabelBarSide": Sets which side of the plot the LabelBar will appear. Make sure to set "lbOrientation".
- "pmLegendSide": Sets which side of the plot the Legend will appear. Make sure to set "lgOrientation".

For specific resources, see the HLU reference guide sections for the LabelBar, Legend, Title, and TickMark objects.

Using maps

MapPlots are a special type of plot. Although instances of MapPlot don't accept data, they do provide a transformation from lat/lon coordinates to screen coordinates. Furthermore, one or more of the other types of plots can be overlayed using the overlay procedure. One important item to note is that MapPlot is really two objects: the MapPlot and the MapTransformation. The only reason to be aware of this is that some MapPlot resources -- those that configure map outlines and colors -- are documented in the MapPlot reference page, and others -- those that configure the transformation -- are documented in the MapTransformation section of the reference guide. Both sets of resources are set through a single MapPlot instance.

The following are important resources for configuring MapPlot:

- "mpDataBaseVersion": Chooses between the low-resolution and high-resolution map outline datasets. To use low resolution, the default, set this resource to "Ncarg4_0", and to use the high-resolution map outline set, set this resource to "Ncarg4_1". Using the high-resolution set on

"MAXIMALAREA" plots is not recommended as this map outline set is very large, and the resolution of most displays is not enough to be of any benefit. For non-"MAXIMALAREA" plots, the high-resolution set provides exceptional resolution.

- "mpProjection": Sets the map projection.
- "mpCenterLatF": Establishes the latitude that will be at the center of the projection plane. This resource is not valid with the LambertConformal projection.
- "mpCenterLonF": Establishes the longitude that will be at the center of the projection plane.
- "mpLimitMode": This is the resource to use to create a MapPlot which is not a maximal projection (i.e. zoom in on a portion). There are many options for this resource, more than can be explained here. The options include:
    - "MaximalArea" - The default, causes entire projection to be rendered.
    - "LatLon" - Limits the projection to a minimum and maximum latitude and longitude. Probably the most straightforward, although some odd aspect ratio plots are possible when using non-rectangular projections. Works great with the "CYLINDRICALEQUIDISTANT" and "MERCATOR" projections. This option uses the resources "mpMinLatF", "mpMaxLatF", "mpMaxLonF", and "mpMinLonF" to set the map limits.
    - "NPC" - Stands for "Normalized Projection Coordinates". This is most useful when using NCL interactively. However, NPC values can be set when creating a map plot. The area defined by the viewport (**vpXF, vpYF, vpWidthF,** and **vpHeightF**) is referenced as values from 0.0-1.0 for each axis starting at the lower left corner. So if you want to zoom in on a specific section of the "MaximalArea" projection, for example say the upper right corner of a plot, you can set "mpLeftNPCF", "mpRightNPCF", "mpTopNPCF", and "mpBottomNPCF" to .5 , 1.0, 1.0, .5 respectively.
    - "NDC" - Stands for "Normalized Device Coordinates" or screen coordinates. This option allows you to set screen coordinates as the reference points for setting the map limits. NDC units run from 0.0 to 1.0 from the lower left corner of the output frame. Use the resources "mpLeftNDCF", "mpRightNDCF", "mpBottomNDCF", and "mpTopNDCF" to set these normalized device coordinate limits.
    - "Corners" - This limit mode is suggested for non-rectangular projections. It allows you to set the corners, in lat/lon coordinates, that you want to be the corners of the new projection.
    - "Points" - Points is the analog of "Corners" using NDC units.
    - "Angles" - Mainly useful with the Satellite projection.
- "mpFillOn": Turns map filling on and off.
- "mpOutlineBoundarySets": Determines which outline set is used. The options are: "NoBoundaries", "Geophysical", "National", "USStates", "GeophysicalAndUSStates", and "AllBoundaries".
- "mpPerimOn": Turns on a line that is drawn around the perimeter of the map.
- "mpGridAndLimbOn": Turns on the grid.
- "mpGridLatSpacingF": Sets the spacing in degrees for the latitude grid.
- "mpGridLonSpacingF": Sets the spacing in degrees for the longitude grid.

Using Resource Files

Resource files are external text files that can be used to create default templates for resources. There are two user-editable default resource files. One can be put in your home directory and is named ".hluresfile", the other is named "ncl.res" and goes in the directory you're running NCL. Different directories can have different "ncl.res" files, which means different directories can have different default plot configurations. There are other more advanced uses of resource files which are not covered here.

See the HLU user guide section on Understanding resources and the section on Using resource files. The file ".hluresfile" is used to supply resources globally, meaning every time you run NCL the defaults set in ".hluresfile" can be used. If the "ncl.res" file exists in NCL's current working directory, then the resources in "ncl.res" can be used. It is important to note that resources in an "ncl.res" file will override those in ".hluresfile".

Every time you create a plot, a name must be given to that plot. This name can be used to determine which resources should be applied to the plot being created. By using the name of the plot, specific resources can be directed to specific plots. For example, consider the following small resource file. This example shows how to set defaults for two different projections. By using this resource file, the NCL script is greatly shortened. To get an idea of how resource files work, run the following example on your own. Save the resources in a file called "ncl.res", and run the script in the same directory. This should be enough to convince you of the utility of resource files and of the convenience that carefully designed resource files can provide.

```
begin
        x = create "x" xWorkstationClass defaultapp end create
        mp0 = create "Pacific Ocean" mapPlotClass x
                "vpXF" : .01
                "vpYF" : .99
                "vpWidthF" : .9
                "vpHeightF" : .5
        end create
        mp1 = create "Atlantic Ocean" mapPlotClass x
                "vpXF" : .01
                "vpYF" : .49
                "vpWidthF" : .9
                "vpHeightF" : .5
        end create
        draw((/mp0,mp1/))
        update(x)
end
```

```
*Pacific Ocean*mpCenterLatF : 0.0
*Pacific Ocean*mpCenterLonF : 180.0
*Pacific Ocean*mpProjection : CYLINDRICALEQUIDISTANT
*Pacific Ocean*mpLimitMode : LATLON
*Pacific Ocean*mpMaxLatF : 67.67
*Pacific Ocean*mpMinLatF : -34.0
*Pacific Ocean*mpMaxLonF : 291.00
*Pacific Ocean*mpMinLonF : 100.00
*Pacific Ocean*vpYF : .92
*Pacific Ocean*mpFillOn : True
*Pacific Ocean*mpMonoFillColor : True
*Pacific Ocean*mpMonoFillPattern : False
*Pacific Ocean*mpMonoFillScale : False
*Pacific Ocean*mpOceanFillPattern : -1
*Pacific Ocean*mpLandFillPattern : 17
*Pacific Ocean*mpLandFillScaleF : .70

*Atlantic Ocean*mpCenterLatF : 0.0
*Atlantic Ocean*mpCenterLonF : 0.0
*Atlantic Ocean*mpProjection : CYLINDRICALEQUIDISTANT
*Atlantic Ocean*mpLimitMode : LATLON
*Atlantic Ocean*mpMaxLatF : 67.67
```

```
*Atlantic Ocean*mpMinLatF : -34.0
*Atlantic Ocean*mpMaxLonF : 17.
*Atlantic Ocean*mpMinLonF : -98.0
*Atlantic Ocean*vpYF : .92
*Atlantic Ocean*mpFillOn : True
*Atlantic Ocean*mpMonoFillColor : True
*Atlantic Ocean*mpMonoFillPattern : False
*Atlantic Ocean*mpMonoFillScale : False
*Atlantic Ocean*mpOceanFillPattern : -1
*Atlantic Ocean*mpLandFillPattern : 17
*Atlantic Ocean*mpLandFillScaleF : .70
```

Using annotations

Annotations are an advanced topic. Essentially Annotations are ancillary plots that can be "attached" to plot that is a PlotManager. This means that the ancillary plot can be redrawn, moved, and even resized by drawing or changing the position of the PlotManager to which it is assigned. These ancillary plots can be TextItems, LabelBars, Legends, TickMarks, or other Plots. For each plot added to a PlotManager as an annotation, an object called an AnnoManager is created. This additional object contains resources for positioning the annotation. The following are key AnnoManager resources:

- "amZone": This specifies an ordering that is used to draw annotations. However, this ordering is spatial rather than sequential. Annotations are drawn from lower number zones out to larger number zones. If two objects occupy zones 2 and 3 respectively, then the plot in zone 3 will be drawn after the plot in zone 2, and it will be drawn such that the two objects don’t overlap. The concept is basically that objects in different zones are guaranteed not to overlap. For more information on how zones work, please read the PlotManager Location Control Model in the HLU reference manual.
- "amSide": In addition to the above zone concept, a side to place the annotation on can be specified. The options are "TOP", "BOTTOM", "LEFT", and "RIGHT".
- "amOn": When True, the annotation will be drawn every time the main plot is drawn. When False, the annotation is not drawn.
- "amTrackData": When True, the annotation will "track" a specific data location. This is useful for positioning a piece of text over a specific data location. This resource must be used with "amDataXF" and "amDataYF".

Drawing lines and polygons

Since lines and polygons need to have colors, dash patterns, and fill patterns associated with them, a special object is needed to specify these options. This object is called a GraphicStyle object. The GraphicStyle object can be modified just like other objects using setvalues. A custom GraphicStyle object can be created using the create command, or the default one can be retrieved from the current workstation using the resource "wkDefGraphicStyleId". The graphic style has resources for changing line styles, colors, polygon fill patterns, markers, and line labels. Also needed to draw lines and primitives is either a plot on which the line or polygon will be drawn, or a workstation on which the line or polygon will be drawn. If a plot is provided, then the polygon or line is clipped to the viewport of the plot. If a workstation is provided, then the polygon or line is clipped to the workstation frame.

There are six procedures that use the GraphicStyle object; three use data coordinates for points, and three use NDC or screen coordinates. Each takes a plot or workstation, a graphic style, and two vectors of coordinates either in NDC units or data coordinates. They are:

- NhlDataPolygon - draws a polygon clipped to a plot's viewport in data coordinates
- NhlDataPolyline - draws a line clipped to a plot's viewport in data coordinates
- NhlDataPolymarker - draws markers in a plot's viewport in data coordinates
- NhlNDCPolygon - draws a polygon clipped to a plot's viewport in NDC units
- NhlNDCPolyline - draws a line clipped to a plot's viewport in NDC units
- NhlNDCPolymarker - draws markers in a plot's viewport in NDC units

NCL command line editing and history

When running NCL interactively at the command line, certain keys can be used to navigate through and edit the 25 most recent commands. The following is a list of the key sequences and what they do.

```
The following control characters are accepted:
     ^A       Move to the beginning of the line
     ^B       Move left (backward) [n]
     ^D       Delete character [n]
     ^E       Move to end of line
     ^F       Move right (forward) [n]
     ^G       Ring the bell
     ^H       Delete character before cursor (backspace key) [n]
     ^I       Complete filename (tab key); see below
     ^J       Done with line (return key)
     ^K       Kill to end of line (or column [n])
     ^L       Redisplay line
     ^M       Done with line (alternate return key)
     ^N       Get next line from history [n]
     ^P       Get previous line from history [n]
     ^R       Search backward (forward if [n]) through history for text;
              must start line if text begins with an up arrow
     ^T       Transpose characters
     ^V       Insert next character, even if it is an edit command
     ^W       Wipe to the mark
     ^X^X     Exchange current location and mark
     ^Y       Yank back last killed text
     ^[       Start an escape sequence (escape key)
     ^]c      Move forward to next character ``c''
     ^?       Delete character before cursor (delete key) [n]

The following escape sequences are provided.
     ESC ^H   Delete previous word (backspace key) [n]
     ESC DEL  Delete previous word (delete key) [n]
     ESC SP   Set the mark (space key); see ^X^X and ^Y above
     ESC .    Get the last (or [n]'th) word from previous line
     ESC ?    Show possible completions; see below
     ESC <    Move to start of history
     ESC >    Move to end of history
     ESC b    Move backward a word [n]
     ESC d    Delete word under cursor [n]
     ESC f    Move forward a word [n]
     ESC l    Make word lowercase [n]
     ESC u    Make word uppercase [n]
     ESC y    Yank back last killed text
     ESC v    Show library version
     ESC w    Make area up to mark yankable
```

```
ESC nn   Set repeat count to the number nn
ESC C    Read from environment variable ``_C_'', where C is
         an uppercase letter
```

Loading Default scripts and shared libraries

There are two environment variables which can be set in the user's shell environment which can allow users to customize their NCL environment. The first environment variable **NCL_DEF_SCRIPTS_DIR** can be used to point to a directory containing scripts the user want to be loaded every time NCL is run by that user. The second environment variable **NCL_DEF_LIB_DIR** can be used to point to a directory containing specially built shared libraries. These shared libraries can contain the user's custom built C or FORTRAN functions and procedures which will appear upon invocation to be part of NCL's intrinsic function set. Instructions for building custom shared libraries to use with NCL can be found in the Extending NCL's function set. The two main sections that apply are Writing your own wrapper as a shared library and Using wrapit77.

Use of NCL_DEF_SCRIPTS_DIR

When using this environment parameter, NCL will attempt to load all files found within the directory pointed to by **NCL_DEF_SCRIPTS_DIR**. Only files ending with the file extension ".ncl" will be loaded. NCL will not follow any subdirectories. It is recommended that the scripts loaded are functions, procedures and perhaps constant initializations. Loading scripts that load other scripts can be done but if a script defines a function or procedure that is already defined error messages may be generated. The undef() procedure can be used to guard against multiple definitions if placed in scripts before function and procedure definitions. It is also important to note that scripts should not depend on a loading order. The scripts are loaded in the order in which they appear in the UNIX directory entries structure.

Use of NCL_DEF_LIB_DIR

This environment parameter will cause NCL to load all shared libraries in the directory. These shared libraries must contain a C function called **Init** which will call either NclRegisterProc or NclRegisterFunc one or more times to register a set of wrapper functions. Information on creating wrapper functions as well as building the shared libraries on a variety of architectures can be found in the NCL Reference Guide section on Extending NCL's function set. The advantage of using this environment parameter over the external statement is that the loaded functions become part of NCL intrinsic function set and can be called without a dynamic library name.

NCL keyword listing

Keywords are reserved by NCL. They cannot be used as names for variables or functions. All keywords in NCL are case sensitive and must appear exactly as they are listed here.

```
begin
```

```
break
byte
character
continue
create
defaultapp
do
double
else
end
external
False
file
float
function
getvalues
graphic
if
integer
load
local
logical
long
new
noparent
numeric
procedure
quit
QUIT
Quit
record
return
setvalues
short
stop
string
then
True
while
```

##

NCL data types overview

Basic numeric types

In NCL, basic data types are all of the standard types found in nearly all programming languages. Types classified as numeric types support all of the algebraic functions available in NCL (see Expressions). The following is a list of the basic numeric types currently supported; they are listed by their keywords, default size for most systems, and valid numerical ranges:

CAUTION: Currently, arithmetic overflow and underflow are not reported as errors to the user. Assignment of out-of-range values may cause errors.

double	64bits	+/- (2.22507e-308) to (8.98846e+307)
float	32bits	+/- (1.175494e-38) to (1.701411e+38)
long	32bits	+/- (2.147483e+09)
integer	32bits	+/- (2.147483e+09)
short	16bits	+/- (32767)
byte	8bits	(0) - (255)

Non-numeric types

Non-numeric types are types for which there is no numeric value and that cannot be coerced into a numeric type. Also, in general, non-numeric types only support the *.ne.* and *.eq.* operations, with the exception of the string and logical types. Strings use the '+' operation to concatenate one or more strings. The logical type supports *.and.*, *.or.*, *.not.*, and *.xor.*. The logical types can have three values: the keywords *True* and *False* or a missing value.

The graphic type is a reference to an instance of an HLU object. The file type is a reference to a file in a supported file format. The logical type is generated from relational expressions. Logical values are either *True* or *False*. Graphic values are returned by the *create* statement, HLU functions or the getvalues statement, and file values are returned by the *addfile* intrinsic function.

string	N/A
character	8bits
graphic	N/A
file	N/A
logical	N/A

Coercion of types

Coercion is basically the implicit conversion of data from one data type to another. This occurs when two values of different types are operands to the same operator. Operands must be of a compatible type to perform the requested operation. For example, when a float value is multiplied by an integer value, the integer value can easily, without losing or corrupting the value, be converted to a float value. Because there is no loss of information, NCL does this automatically. The following table lists the automatic conversion possibilities for all of the basic types. The data types on the left side can automatically be converted to values of the types on the right side. If a coercion to a common type doesn't exist for both sides of an expression, then a type mismatch error is generated and the script will exit.

Coercion is discussed again in the section of this document covering expressions.

One very important item, which is covered in the functions and the procedures sections, has to do with how parameters work in functions and procedures. First, parameters in NCL are pass-by-reference, meaning that a change to the value of a parameter inside a function changes the value of a variable passed in to the function or procedure. This is important because when a parameter is expected to be a certain type, say *float*, and an *integer* variable is passed as the parameter, the *integer* parameter must be coerced to a *float*. Since there is no automatic conversion possible from *float* to *integer*, changes to the parameter within the function or procedure can **not** be propagated back to the calling variable. A warning message is given when this occurs.

Type	Coercible to

`character`	`string`
`byte`	`character` `string` `short` `integer` `long` `logical` `float` `double`
`short`	`character` `string` `integer` `long` `logical` `float` `double`
`integer`	`string` `long` `logical` `float` `double`
`long`	`logical` `string` `float` `double`
`float`	`logical` `string` `double`
`double`	`logical` `string`

A special set of functions exists for forcing the coercion in the reverse direction. For example, the function doubletointeger can be used to convert a double precision number into an integer. This operation will cause the decimal portion to be truncated, and information will be lost. See NCL functions and procedures for other similar functions.

Creating data

Data exist in NCL either as a single scalar value or a multi-dimensional array of scalar elements. The term *value* in NCL can refer to either a single scalar value or a multi-dimensional array of a specific data type.

There are three ways data can be created:

- By entering constant values.
- By using the *new* command to allocate space for a multi-dimensional value.
- By using functions like cbinread, fbinrecread, and addfile.

Constants and Arrays of constants

Constant values are values that are either entered at the command line or are written textually in an NCL script. There are **only three data types** in which constant values are expressed: *float*, *integer*, and *string*. Floating point constants are entered either in scientific notation or as numbers with a decimal point. It is not possible to specify a double precision constant. However it is possible to assign a floating point constant to a double precision variable. Integer constants are entered without decimal points. String constants' values contain characters enclosed in quotes (").

The following are examples of single scalar constant values:

```
Floating Point Constants:

        3.141592        1e-12   2.

Integer Constants:

        100     1

String Constants:

        "a"     "Hello World"
```

NOTE: There is currently no way to specify characters literally. The *stringtochar* function can be used to convert a string to an array of characters though.

Arrays of constants can be constructed using the array designator characters *"(/"* and *"/)"* with constant values separated by commas. Multi-dimensional constant arrays can be created by nesting the array designator characters. The following are examples of constant arrays. Each example uses the assignment statement to assign the constant arrays to a variable.

```
1D Floating Point Constant Array:

        var0 = (/ 1.2, 2.3, 3.4, 4.5, 5.6, 6.7 /)

[2]x[6] 2D Floating Point Constant Array:

        var1 = (/ (/ 1.2, 2.3, 3.4, 4.5, 5.6, 6.7 /), (/ 7.8, 8.9, 9.1, 10.2, 1

1D Integer Constant Array:

        var2 = (/ -5, -4, -3, -2, -1, 0, 1, 2, 3, 4, 5 /)

1D String Constant Array:

        var3 = (/ "one", "two", "three", "four", "five" /)
```

Creating new data arrays

All numeric data, string, character, logical, and graphic types can be created using the *new* statement. It is important to note that *new* is **not a function**; it is a statement. Because *new* is a statement, there are two ways to use it, and it is possible to use a type keyword as an argument to *new*. The two possible ways to use *new* are:

- Create an array of data with a specific missing value assigned to each element.
- Create the array of data using the default missing value.

The *new* statement takes, as parameters, an array of integer dimension sizes, the keyword for the data type, and optionally a missing value to assign to each element of the new data array. The following creates a [5]x[6]x[7] three-dimensional float array filled with the default missing value for *float* types and the attribute *_FillValue* is also set:

```
a = new( (/ 5, 6, 7 /), float)
print(a)


Variable: a
Type: float
Total Size: 840 bytes
        210 values
Number of Dimensions: 3
Dimensions and sizes:   [5] x [6] x [7]
Coordinates:
Number Of Attributes: 1
        _FillValue :  -999
(0,0,0) -999
(0,0,1) -999
(0,0,2) -999
 . . .
```

The default missing values are:

```
logical   : -1
byte      :  0377 (octal)
short     : -99
integer   : -999
long      : -9999
float     : -999
double    : -9999
graphic   : -9999
file      : -9999
character : 0 or '\0' for those who know C
string    : "missing"
```

The following is an example of how to assign a specific missing value. The result is an array filled with -1e12 at every index.

```
a = new( (/ 5, 6, 7 /), float, -1e12)
```

Importing data arrays and files

Data can be read in to NCL in a variety of ways. If data exist as a UNIX file in either ASCII, C, or Fortran binary data, the data can be read in to NCL with one of the following NCL intrinsic functions respectively, asciiread, cbinread, or fbinrecread. Each of these functions requires the user to enter a string file name, a dimension size array similar to the *new* command, and finally a string denoting what type the data are stored in. These functions return a single array of a single data type per call. Currently, support for binary or ASCII files containing more than one array is not officially supported. The NCL scientific tutorial contains some examples of how to read multiple records from a Fortran binary file.

Another way to import data is to import from a supported file format. Supported file formats are specially recognized data formats that store variables and other information. Supported formats allow direct access to variables and other information in data files through NCL. This access is very different than the abovementioned methods of reading files. NCL has a special syntax for referencing variables in files that simplifies the importation of external data. The addfile function is used to open external files that are in a supported format. The **addfile** function returns a reference to a file that is used to access data within that file. This is all covered in the variables section of this reference guide.

For specific information on a supported file format, see the Supported data format information section of the reference guide. Currently, with version 4.1, these formats are CCM history tape, netCDF, GRIB, and HDF.

NCL variables overview

Properties of variables

Variable names must begin with an alphabetic character, but they can contain any mix of numeric and alphabetic characters. The underscore ('_') is also allowed. Variable names are also case sensitive. The maximum variable name length is currently 256 characters. Variables reference arrays of multi-dimensional data. These data can be described by variable attributes, named dimensions, and coordinate variables. Variables can also reference files and graphical objects.

The following are examples of unique variable names:

```
a
A
forecast_time
s092389
__t__
```

Variables, like in other programming languages, are textual names that reference data. In NCL, somewhat like Fortran, variables can be created without previously defining them. This is call-implicit instantiation. Unlike Fortran though, the type of a variable is based on what type of value is assigned to it, not the name. Unlike other languages, variables in NCL can be deleted, or changed from defined to undefined.

NCL has been designed with special features that allow ancillary information to be "attached" to a variable programmatically. NCL provides a unique syntax for storing and retrieving these ancillary data values. These ancillary data are divided up into three categories: variable attributes, dimensions, and coordinates. Variables can have an unlimited number of attributes assigned to them. Each dimension of a variable can have a name associated with it and optionally a coordinate variable. Variables become defined when an undefined name appears on the left side of an assignment statement.

There are three types of variables referenced in this document. The first and most obvious is the term *variable* which is defined as a textual reference to a multi-dimensional or scalar value. Second, there is the term *variable that references a file*, which is defined as a variable that is assigned the return value of the addfile function. These variables provide references to open files. Finally, the term *file variable* is defined to be a variable in a file that references a multi-dimensional data value. These terms will be used throughout this section of the reference guide to distinguish between different kinds of variables.

Attributes

Attributes are descriptive pieces of information that can be associated with an already existing variable, or file variable. They are very useful for communicating information to the user about specific data. Attributes can be assigned single-dimensioned arrays, but not files. Variable attributes are referenced by entering the variable name, followed by '@', followed by the name to be used to reference the attribute. If the attribute is not defined, then an error message is displayed. An attribute is created by referencing it on the left side of an assignment statement and assigning a value to it. The following are examples of creating the attributes *units, long_name,* and *_FillValue* in the variable named *temperature*:

```
temperature@units = "Degrees C"
temperature@long_name = "Temperature at Tropopause"
temperature@_FillValue = -9999.0
```

Attributes can be used in expressions and subscripted in the same fashion as variables. Common uses of attributes are to store the units that data are stored in, and to store names and text that could be useful.

Missing values

The attribute *_FillValue* is a special reserved attribute name that denotes what values stored in a variable should be considered missing values. Whenever the *_FillValue* attribute is assigned a new value, every occurrence of the previous value in the variable temperature, in the above example, is replaced with the new *_FillValue*. The *_FillValue* attribute must be the same type as the data type referenced by the variable. The procedure delete is used to remove the missing value attribute. Once removed, all of the previous elements of the variable that were treated as missing values are treated as normal values.

The *_FillValue* attribute has many important uses in NCL. First and most important is how missing values are handled in expressions. When missing values appear in terms of an NCL expression, they are ignored for the specific element that contains the missing value. Consider the following example:

```
a = (/27.2, -10.0/)
a@_FillValue = -10.0
b = a * 9.0/5.0 + 32.0
print(b)

Variable: b
Type: float
Total Size: 8 bytes
            2 values
```

```
Number of Dimensions: 1
Dimensions and sizes:   [2]
Coordinates:
Number Of Attributes: 1
  _FillValue :  -10
(0)     80.96
(1)     -10
```

In this example, the constant array *(/ 27.2, -10.0/)* is assigned to the variable *a*. Next the value of -10.0 is assigned as the *_FillValue* attribute. When the expression is evaluated, the element equal to -10.0 in the array is ignored, and the result is the array referenced by *b* has -10.0 as its _FillValue. See Expressions and missing values for more discussion.

The default missing values for the various types are:

```
logical   : -1
byte      :  0
short     : -99
integer   : -999
long      : -9999
float     : -999
double    : -9999
graphic   : -9999
file      : -9999
character : 0 or '\0' for those who know C
string    : "missing"
```

Dimensions

Dimensions define the shape and size of the data referenced by variables. In NCL, dimensions are ordered using *row x column* ordering, which is identical to the C programming language. By convention, dimensions are numbered from 0 to n-1 where n is the number of dimensions of the data referenced. The dimension numbers are significant because NCL allows names to be associated with dimensions. This in turn facilitates coordinate subscripting and named subscripting. A variable dimension is referenced by entering the variable name, followed by the '!' character, followed by the dimension number being referenced. If the dimension has been assigned a name, then this reference returns the name. To assign or change a name to a dimension, simply assign a string to the dimension number in the following fashion:

```
temperature!0 = "frtime"
temperature!1 = "lat"
temperature!2 = "lon"
```

The previous example is valid only if temperature has three or more dimensions.

Coordinate variables

Coordinate variables are single-dimension arrays that have the same name and size as the dimension they are assigned to. These arrays represent the data coordinates for each index in the named dimension. When the values in these arrays are monotonically increasing or decreasing, they can be used in coordinate subscripting. If they are not monotonic or contain missing values, then coordinate subscripting will not work. The '&' operator is used to reference and assign coordinate variables. In order to assign a coordinate variable to a dimension, the dimension must have a name. These examples show assignment of variables to coordinate variables:

```
temperature&frtime = forecast_times
temperature&lat = lat_points
temperature&lon = lon_points
```

String references

Sometimes it is impossible to know the names of the attributes and coordinates before writing a script, or these names may vary from variable to variable. To solve this problem, string variables can be used to reference attributes and coordinates by enclosing the variable reference within dollar signs ’$’. The following are examples of this:

```
        dimnames = (/"frtime","lat","lon"/)
        attnames = (/"_FillValue", "long_name"/)

;
; access to attribute
;
        att0 = temperature@$attnames(0)$

;
; Example of referencing a coordinate variable
; without knowing the dimension name
;
        if(iscoord(dimnames(0))
                coord0 = temperature&$temperature!0$
        end if
```

Variables used as parameters to functions and procedures

When functions and procedures are called, in NCL, the parameter passing mechanism is called **pass-by-reference**. This means, just like FORTRAN, that changes made within a function or procedure to a parameter are also applied to the variable in the calling environment. With NCL, however, there are some differences which must be considered when passing variables as parameters. In NCL changes to named dimensions, coordinate variables and attributes also affect the variable in the calling environment. Furthermore, if the variable is subscripted prior to calling a function or procedure the values are remapped back into the original variable once the function or procedure is terminated.

In the following example the variable **a** is subscripted note how the assignments in the procedure **set** are propagated back to the calling environment:

```
procedure set(x)
begin
        x = 1
        x!0 = "dim1"
        x@_FillValue = 1
end

a = (/(/1,2,3/),(/4,5,6/),(/7,8,9/)/)

set(a(1,:))
```

```
print(a)
```

It is important to consider this functionality whenever writing an NCL function or procedure that makes any kind of assignments to the input parameters.

Subscripts

There are three types of subscripting in NCL. Standard is similar to the array subscripting available in Fortran 90. One very important item to note is that NCL dimension indexes start at **0** and end at **n-1**. Second, coordinate subscripting uses the data in the coordinate variables for determining subsections of the array to select. Third, named subscripting uses the names of the dimensions to allow for array reordering operations. All three types of subscripting can be used in a single variable selection.

Standard subscripts

Standard subscripting provides the capability of selecting ranges and strides in addition to the ability to select data using a vector of integer indexes. All of this functionality is more or less duplicated from Fortran 90. The following is a simple example of a set of single subscripts for a three-dimensional variable with dimensions 5x6x7. Unlike Fortran, the array indexes begin at 0. Standard subscript indexes must always be integer; floating point numbers and strings are not accepted.

Note: Individual subscripts are separated by commas ',' and the entire subscript list is enclosed in parentheses.

```
temperature(0,5,6)
```

A range subscript selection accepts a beginning and ending index separated by a colon ':'. Both the beginning index and the ending index are included in the selection, therefore the range is inclusive. Furthermore, if the start is greater than the end, then the selection reverses the ordering of the array. Some examples are:

```
temperature(1:3,5,6)
temperature(1:3,4:5,5:6)
```

The first selection selects a 3x1x1 subsection of the array temperature, and the second selection selects a 3x2x2 subsection. In addition to the above style of selection, a stride can also be specified that causes the selection to skip over a given number of elements. For example, a value of 1 means that every element from the beginning of the range and the end of the range will be selected. With values greater than 1, the first index of the subscript is followed by the stride plus the current index. Therefore a value of 3 selects the first, fourth, seventh, and so on.

```
temperature(0:4:2,0:5:3,0:6:4)
```

The above selection uses strides to produce a 3x2x2 array.

There is no restriction on having the start of a subscript range be less than the end of the subscript range. When the start is greater than the end, a reverse selection is done, meaning the order output selection is reversed from the original variable. For example:

```
temperature(3:1,5,6)
temperature(3:1,4:5,5:6)
```

Another option for selection is to leave out the start, end, or both. This means that the start or end will default to the beginning or end respectively.

```
temperature(:2,:1,5:)
temperature(:,:,:)
```

The first selection selects from the beginning to index 2 for the first dimension, the beginning to index 1 for the second dimension, and the final subscript selects from index 5 to the end for the 3rd dimension. The second example shows how the entire array can be selected.

The following uses the default range to reverse the ordering of each dimension using a negative stride:

```
temperature(:2:-1,:1:-1,5::-1)
temperature(::-1,::-1,::-1)
```

Finally, a vector of integer indexes can be used as a subscript. As long as all of the entries in the vector are within the bounds of the given dimension, the vector could be any size. Vector subscripting allows a single index to be selected more than once. For example, consider the following array and its use of vector subscripting on the variable temperature:

```
(/1,1,1,2,2,2/)

temperature((/1,1,1,2,2,2,/),:,:)
```

This selection creates an array 6x6x7 which is actually bigger than the original. The first, second, and third indexes of the first dimension contain identical arrays. The vector must always be a single-dimensioned array of integers.

Coordinate subscripts

Coordinate subscripts use the coordinate variables associated with a variable to determine which indexes are used in the selection. When specifying a coordinate subscript, braces '{' and '}' indicate the start and end values of the coordinate variable that will be used to select the indexes. Essentially, the start and end values are "looked" up in the coordinate variable, and the indexes are used to make the subselection. The following are examples of coordinate subscripts. Note that coordinate and standard subscripting can be mixed in the same variable subscript. Also, stride is still specified as an integer stride. If the coordinate values used in the subscript do not exactly match values in the coordinate variable, all coordinate values that fall within the coordinate subscript range are selected. If the values do match, then they are selected in an inclusive fashion.

```
temperature(0,{20:60},{-95:-120})
temperature(0,{20},{-95})
temperature(0,{:20:2},{:-95:2})
```

Coordinate subscripting only works when the coordinate variables assigned to the variables are monotonically increasing or decreasing. If an attempt is made to subscript a coordinate variable that is not monotonic, an error message is generated.

Named subscripting

Named subscripting is a means by which arrays can be reordered. Named subscripting requires that each dimension of the variable being subscripted is named. If one or more are not named, an error messages is printed. The following are examples of named subscripting. The dimension names of the variable temperature are "time", "lat", and "lon" for dimensions 0 through 2 respectively.

```
temperature( time | 0, lon | :, lat | :)
temperature( time | :, {lon | 20 : 60}, {lat | -95 : -120})
```

The first example "swaps" the lat and lon dimensions. The second example shows a similar dimension reordering but utilizes coordinate subscripting.

Using string reference with named subscripting

It is not necessary to "hard-code" the names of dimensions when using named subscripting. Alternatively, dollar signs '$' can be placed around a string variable. This causes NCL to use the variable's string value as the dimension name. The following shows how an array can be reordered without knowing the names of the dimensions:

```
dims = getfilevardims(file1,"T")
T = file1->T( $dims(1)$ | :,  $dims(0)$ | :, $dims(2)$ | :)
```

Variable assignment

It is important to understand what happens when a variable is used in an assignment statement in NCL. The assignment statement functions differently depending on whether the variable being assigned to is currently undefined or defined. The assignment statement also functions differently depending on whether a variable or a value occupies the right side of the assignment.

When a variable appearing on the left side of an assignment has not be defined or was previously deleted, the assignment statement causes the variable to become defined and the data type and dimensionality of the variable is determined by the right side.

When a variable appearing on the left side is already defined, then the right side must have the same type, or be coercible to the type on the left, and the right side must have the same dimensionality.

Value-only assignment

Value-only assignments to variables are fairly straightforward. In essence, value-only assignments mean that the right side of the assignment is **not a variable**, it is the result of an expression, a value. In this case, if the left side variable reference was not defined prior to the assignment statement, the variable on the left side becomes defined and references the value of the right side. No dimension names, coordinate variables or attributes other than *_FillValue* are assigned. If the right side of the expression does not contain any missing values, then *_FillValue* is not assigned either.

If the left side variable was defined prior to the assignment statement, then the value on the left side is assigned the value of the right side. If the left side is a subscripted reference to a variable, then the right side elements are mapped to the appropriate location in the left side variable. If the left side has any attributes, dimension names, or coordinate variables, they are left unchanged since only a value is being assigned to the left side variable. When the left side is defined, then the type of the right side and the dimensionality must match. However, there is one exception to the requirement that the dimension sizes of the left side and the right side match, a single scalar value can be assigned to more than one location. Consider the following example:

```
a = (/1,2,3,4,5,6,7,8,9,10/)
a(0:3) = -1
print(a)

Variable: a
Type: integer
Total Size: 40 bytes
            10 values
Number of Dimensions: 1
Dimensions and sizes:   [10]
Coordinates:
(0)     -1
(1)     -1
(2)     -1
(3)     -1
(4)     5
(5)     6
(6)     7
(7)     8
(8)     9
(9)     10
```

This example demonstrates the value of -1 being assigned to the first four elements of the variable a.

Variable-to-variable assignments

During variable-to-variable assignment attributes, coordinate variables and dimension names, in addition to actual multi-dimensional values, are assigned. Before discussing this type of assignment, it is important to note that the array designator characters '(/' and '/)' can be used when assigning one variable to another to force only the right side's value to be assigned to the left side and the right side's attributes, dimensions, and coordinates are ignored. Essentially using the array designator characters forces value-to-variables assignment. The following shows how the array designator characters can be used to do this:

```
;
; Example of array designator use to force "Value Only" assignment
;
        variable1 = (/ variable2 /)
```

Variable-to-variable assignment occurs when both the left side and the right side are variables. In this situation, the assignment statement also tries to assign attributes, dimension names, and coordinates of the right side to the left side.

The two simplest cases are:

- The left side is undefined prior to the assignment
- The variable on the left side is not subscripted, meaning the entire variable is being referenced

In both these situations, **all** of the right side's attributes, coordinates, and dimension names are assigned to the left side. If the left side has the same dimension and coordinate names, then only the coordinate variable is overwritten with the value and attributes of the right side's coordinate variables. However, if the names of the dimension names **do not match**, a warning message is generated and the names and coordinate variables of the left side are overwritten. As far as attributes go, if the left side has attributes, then the left side's attribute list is merged with that of the right side. If the same attribute name appears on both the left and right sides, the right side's attribute overwrites the left side's. If the types of the attribute values do not match, you could have a type mismatch error.

The following are examples of some variable-to-variable assignment situations:

This first example shows assignment to an undefined variable and then shows the use of the array designator characters '(/' and '/)' to perform a value-only assignment.

```
;
; Create variable be with values, dimension names, coordinate variables
; and attributes
;
        b = (/ (/1.0,2.0,3.0/), (/4.0,5.0,6.0/), (/7.0,8.0,9.0/) /)
        b!0 = "dim0"
        b!1 = "dim1"
        b@units = "none"
        b&dim0 = (/.1,.2,.3/)
        b&dim1 = (/10,100,1000/)

;
; Variable-to-variable assignment with left side undefined
;
        a = b

;
; Use of array designator characters to assign "Value Only" to undefined
; left side
;
        c = (/b/)

;
; This print shows that all of the dimension names, attributes, and coordinate
; variables have been assigned to a.
;
        print(a)

        Variable: a
        Type: float
        Total Size: 36 bytes
                     9 values
        Number of Dimensions: 2
        Dimensions and sizes:   [dim0 | 3] x [dim1 | 3]
        Coordinates:
                    dim0: [0.1..0.3]
                    dim1: [10..1000]
        Number Of Attributes: 1
          units :       none
        (0,0)    1
        (0,1)    2
        (0,2)    3
```

```
        (1,0)   4
        (1,1)   5
        (1,2)   6
        (2,0)   7
        (2,1)   8
        (2,2)   9

;
; This print shows that only the values of b were assigned to c.
;
        print(c)

        Variable: c
        Type: float
        Total Size: 36 bytes
                     9 values
        Number of Dimensions: 2
        Dimensions and sizes:   [3] x [3]
        Coordinates:
        (0,0)   1
        (0,1)   2
        (0,2)   3
        (1,0)   4
        (1,1)   5
        (1,2)   6
        (2,0)   7
        (2,1)   8
        (2,2)   9
```

This second example demonstrates a defined variable being assigned to a defined variable. Note the changes resulting from assignment to the dimension names, attribute values, and coordinate variables in variable *a*. These assignments that change the left side's coordinates and dimension names generate errors. When left and right dimension names are different, NCL considers this an error that the user should be warned about. To avoid these errors you can either make sure before assignment that the left and right sides have the same dimension names, or if you only want to assign a value and don't care about attributes, dimensions, and coordinate variables, you can enclose the right side using '(/' and '/)', which forces NCL to use only the value of the right side.

```
;
; Define variable a with value, dimension names and attributes.
; No coordinate variables assigned.
;
        a = (/ (/1.1,1.2,1.3/), (/2.1,2.2,2.3/), (/3.1,3.2,3.3/) /)
        a!0 = "test0"
        a!1 = "test1"
        a@units = "Degrees"
        a@long_name = "A"

;
; Define variable b with value, dimension names, attributes, and
: coordinate variables.
;
        b = (/ (/1.0,2.0,3.0/), (/4.0,5.0,6.0/), (/7.0,8.0,9.0/) /)
        b!0 = "dim0"
        b!1 = "dim1"
        b@units = "none"
        b&dim0 = (/.1,.2,.3/)
        b&dim1 = (/10,100,1000/)
```

```
;
; Here is the "Variable-to-variable" assignment. The dimension names of a
; change, and the coordinate variables of b are assigned to a. In addition,
; the attribute lists are merged.
;
        a = b
        print(a)

        Variable: a
        Type: float
        Total Size: 36 bytes
                    9 values
        Number of Dimensions: 2
        Dimensions and sizes:   [dim0 | 3] x [dim1 | 3]
        Coordinates:
                    dim0: [0.1..0.3]
                    dim1: [10..1000]
        Number Of Attributes: 2
          units :       none
          long_name :   A
        (0,0)    1
        (0,1)    2
        (0,2)    3
        (1,0)    4
        (1,1)    5
        (1,2)    6
        (2,0)    7
        (2,1)    8
        (2,2)    9
```

The remaining case is that when the left side is subscripted, only a portion of the target variable is being assigned to. The simplest case here is when the left-side dimension names are the same and both the left side and right side have coordinate variables for the same dimensions. In this case, assignment occurs for each coordinate variable. The subscripted left-side coordinate variable is assigned the subscripted right side coordinate. The attributes lists for the right side is merged with that of the left side and assigned to the left side variable. The following demonstrates this kind of variable-to-variable assignment.

```
;
; Define variable a with values, dimension names, attributes and
; coordinate variables.
;
        a = (/ (/1.1,1.2,1.3/), (/2.1,2.2,2.3/), (/3.1,3.2,3.3/) /)
        a!0 = "dim0"
        a!1 = "dim1"
        a&dim0 = (/.1,.2,.3/)
        a&dim1 = (/.1,.01,.001/)
        a@units = "Degrees"
        a@long_name = "A"

;
; Define b with same dimension names, and assign different coordinate
; variables for dim1.
;
        b = (/ (/1.0,2.0,3.0/), (/4.0,5.0,6.0/), (/7.0,8.0,9.0/) /)
        b!0 = "dim0"
        b!1 = "dim1"
        b@units = "none"
```

```
        b&dim0 = (/.1,.2,.3/)
        b&dim1 = (/10.0,100.0,1000.0/)

;
; Here is the example of "Variable to Variable" assignment where the left
; side is already defined. The coordinate variable for "dim1" is overwritten.
;
        b(0,:) = a(0,:)
        print(b)

        Variable: b
        Type: float
        Total Size: 36 bytes
                     9 values
        Number of Dimensions: 2
        Dimensions and sizes:   [dim0 | 3] x [dim1 | 3]
        Coordinates:
                    dim0: [0.1..0.3]
                    dim1: [0.1..0.001]
        Number Of Attributes: 2
          units :       Degrees
          long_name :   A
        (0,0)   1.1
        (0,1)   1.2
        (0,2)   1.3
        (1,0)   4
        (1,1)   5
        (1,2)   6
        (2,0)   7
        (2,1)   8
        (2,2)   9
```

If the left side variable does not have a coordinate variable and the right side does, a coordinate variable is created and assigned. If the left side is subscripted, then the created coordinate variable only has values assigned for the subscripted range, and the rest of the coordinate variable is filled with missing values. The following example illustrates this feature:

```
;
; Define b with no coordinate variables.
;
        b = (/ 1.0,2.0,3.0,4.0,5.0,6.0,7.0,8.0,9.0/)
        b!0 = "dim0"

;
; Define a with coordinate variables.
;
        a = (/ 1.1,1.2,1.3,2.1,2.2/)
        a!0 = "dim0"
        a&dim0 = (/.1,.2,.3,.4,.5/)

;
; Assignment of a to b. Selection of dim0 selects only every other element.
;
        b(::2) = a(:)

;
; Print of the coordinate variable "dim0" demonstrates filling of missing
; value for non-selected element.
;
        print(b&dim0)
```

```
Variable: dim0 (coordinate)
Type: float
Total Size: 36 bytes
            9 values
Number of Dimensions: 1
Dimensions and sizes:   [dim0 | 9]
Coordinates:
Number Of Attributes: 1
  _FillValue :  -999
(0)     0.1
(1)     -999
(2)     0.2
(3)     -999
(4)     0.3
(5)     -999
(6)     0.4
(7)     -999
(8)     0.5
```

The final situation that must be considered when assigning one variable to another is when the dimension names of the left side and the right side do not match. In this case, the assignment overrides the left side's dimension names and coordinate variables, and a warning message is generated. If this is not the desired effect, then the array designator characters '(/' and '/)' can be used to make the assignment a "value-only" assignment.

HLU object variables

HLU object variables are variables of type *graphic*. HLU object variables reference HLU objects that were either created with the create statement or were retrieved from an object with the getvalues statement. Arrays of HLU objects are supported by NCL. The same HLU functions that are available through C and Fortran are available from NCL to operate on HLU objects. See NCL versions of HLU functions and procedures for more information. The interfaces to many of the functions have been modified to support operations on one or more HLU object.

HLU object variables support assignment and all of the other properties of NCL variables. Also, HLU object variables can be compared with the .ne. and .eq. operators.

Files and file variables

Once again it is very important to understand the distinction between a **variable that references a file** (from now on simply called a **file**) and a **file variable**. When a file is opened with the addfile function, a reference to the file is assigned to a variable. The variable's data type is *file*. This variable is a **variable that references a file**. On the other hand, a **file variable** is a variable that is contained within a file. There are many ways to get information about **file variables**. Calling the procedure print with a variable that references a file as a parameter will produce a listing of all of the file's attributes, dimensions, variables, and coordinate variables. The procedure list_filevars produces a similar listing. The function getfilevarnames returns an array of strings that contains the string name of all the file variables in the file. filevardimsizes should always be used to determine the dimension sizes of a file variable. It is very important to use *filevardimsizes* since calling *dimsizes* may inadvertently force the entire file variable to be read into memory. getfilevaratts and getfilevardims are also useful functions.

Opening data files

Opening files is done with the addfile function. Addfile takes two parameters. The first is a UNIX pathname string, either relative or absolute, to the file; the second parameter is a string option. Currently there are three options for the second parameter, "w", "r", and "c". "w" means open the file with read/write permissions, "r" means open the file with read-only permissions, and "c" means create the file. "c" will return an error message if the file already exists. "w" will return an error message if the permissions of the file and/or directory are not correct.

The addfile function uses the file extension (i.e. ".nc" or ".cdf" for netCDF, ".hdf" for HDF, ".ccm" for CCM history files, and ".grb" for GRIB) to determine what type of file to open. Once open, all files and file variables regardless of type are referenced using the same NCL file reference syntax. For more information on what types of file formats are currently supported and special conventions of a specific file format, see the Supported data format information section of the reference guide.

Referencing file variables

The '->' operator is used to reference specific variables in a file. The following examples demonstrate referencing a file variable, a file variable attribute, and a file variable coordinate variable:

```
a = file1->temperature(0,:,:,:)
att = file1->temperature@units
lon = file1->temperature&lon
```

Using the '->' operator requires the variable name to appear immediately after the '->'; parentheses and expressions are not allowed immediately after the '->'. A different kind of file variable access is available to reference variables by a string expression. This is covered later.

File variables function just like regular NCL variables with respect to what is outlined in the NCL properties of variables section. It is important to understand that only the section defined by the variable reference is read or written. This means that NCL allows direct access to file variables, the entire variable does not have to be read in to NCL at one time. For instance, if file1 contains a variable "elev" that is dimensioned [lat | 2159] x [lon | 4320], the selection file1->elev({20:60},{-135:-65}) will read in only the [lat | 480] x [lon | 840] subsection defined by the reference leaving the remaining 8923680 data points in the file.

File variable string references (Using '$' to reference file variables)

Sometimes it is convenient to write generic scripts that do depend on specific variable, attribute, or dimension names. This can be accomplished by using string variables containing the name of a variable rather than by hard-coding the names. Basically, string references work by putting a dollar sign '$' before and after the string variable. When NCL encounters this syntax, it uses the string value of the variable for the variable, attribute, or dimension name. There are several functions that are convenient to use with this feature. They are getfilevarnames, getfilevaratts, and getfilevardims.

The following is an example of how to copy a file from one format to another *without* knowing the names of the variables in the file.

```
gribfile = addfile(ncargpath("data") + "/grb/ced1.lf00.t00z.eta.grb","r")
ncfile = addfile("./ced1.lf00.t00z.eta.nc","c")
names = getfilevarnames( gribfile )
do i = 0, dimsizes( names ) - 1
        ncfile->$names(i)$ = gribfile->$names(i)$
end if
```

The following are examples of referencing attributes and dimensions using string references:

```
ncl < fileinfo.ncl > file.output

        begin
;
; Open a file.
;
        gribfile = addfile(ncargpath("data") + "/grb/ced1.lf00.t00z.eta.grb","r'
;
; Get the names of the variables.
;
        names = getfilevarnames( gribfile )

;
; Loop on all the variables.
;
        do i = 0, dimsizes( names ) - 1
                print("Variable Name: " + names(i))
;
; Retrieve variable information.
;
                dims = getfilevardims(gribfile,names(i))
                sizes = filevardimsizes(gribfile,names(i))
;
; Print variable information.
;
                print(dimsizes(sizes) + " Dimensions:")
                if(.not.any(ismissing(dims))) then
                        do j = 0, dimsizes(dims) -1
                                print( j + ") " + dims(j) + ": " + sizes(j))
                        end do
                end if
                atts = getfilevaratts(gribfile,names(i))
                if(.not.any(ismissing(atts))) then
                        do k = 0, dimsizes(atts) -1
;
; Example of accessing attributes and variable using the string
; reference technique.
;
                                print(atts(k) + ": " +gribfile->$names(i)$@$att:
                        end do
                end if
                delete(atts)
                delete(dims)
                delete(sizes)
                print("")
        end do
end
```

Assignment to file variables

The rules for assignment of variables to files are the same as those for assigning variables to variables with a couple of exceptions. The first and most important is that when assigning a variable to a file, you must define the dimension names. If you don't, NCL will pick ones for you. The second exception is that **dimensions, coordinate variables and variables cannot be deleted from a file**. This means that a warning message will be generated when the assignment tries to delete a dimension name or coordinate variable. To avoid these errors, make sure your dimension names are defined before assignment, and when you do assign a variable to a file variable that is already defined, make sure your right side has the same dimension names and coordinate variables. One way to do this is to use the following four procedures to pre-define your variables, attributes, dimensions, and coordinate variables: filedimdef, filevardef, filevarattdef, and fileattdef.

NCL expressions overview

Properties of expressions

The results of expressions are values. Anything like functions, constant values, and variable references are expressions. Expressions can also be nested by using algebraic operators to string together expressions. Expressions are grouped into subsets. There are algebraic expressions that take numeric operands and produce numeric results. There are logical expressions that take any type of operand and produce logical results. There are functions that can produce any type of value and can produce variables. Finally there are arrays that group one or more expressions into a multi-dimensional value.

Multi-dimensional arrays and expressions

NCL's algebra, like Fortran 90, supports operations on entire arrays rather than single scalar values like C and PASCAL. For array operations to work, both operands must have the same number of dimensions and same size dimensions, otherwise an error condition occurs. Furthermore, the data types of each operand must be equivalent, or one operand must be coercible to the type of the other operand. The following is an example of two entire arrays being multiplied together in a single expression; note that a is *integer* and b is *float*:

```
;
; Define two [2]x[2] arrays
;
        a = (/ (/ 1, 2 /), (/ 3, 4 /) /)
        b = (/ (/ .1, .01 /), (/ .001, .0001 /) /)

;
; Multiplication of two [2]x[2] arrays
;
        c = a * b
        print(c)

        Variable: c
        Type: float
        Total Size: 16 bytes
                    4 values
        Number of Dimensions: 2
```

```
        Dimensions and sizes:   [2] x [2]
        Coordinates:
        (0,0)   0.1
        (0,1)   0.02
        (1,0)   0.003
        (1,1)   0.0004
```

The above example illustrates how multi-dimensional arrays are handled in NCL expressions. The above statement c = a * b is equivalent to looping through both dimensions of a and b, multiplying each element and assigning it to the respective element in c. The following code segment is equivalent to the above example. This shows how NCL's syntax saves a lot of typing.

```
dims = dimsizes(a)
c = new(dims,float)
do i = 0,dims(0) - 1
        do j = 0, dims(1) - 1
                c(i,j) = a(i,j) * b(i,j)
        end do
end do
```

Scalar values and expressions

Scalar values are special cases when considering array operations. When scalar values appear in an expression with a multi-dimensional value, the scalar value is applied to each element of the multi-dimensional value. For example, each element in an entire array can be doubled by multiplying the array by the scalar value 2. The following example shows this.

```
        a = (/ (/ 1, 2 /), (/ 3, 4 /) /)
;
; Multiplication of an entire array by a single scalar
;
        b = a * 2
        print(b)

        Variable: b
        Type: float
        Total Size: 16 bytes
                    4 values
        Number of Dimensions: 2
        Dimensions and sizes:   [2] x [2]
        Coordinates:
        (0,0)   2
        (0,1)   4
        (1,0)   6
        (1,1)   8
```

Expressions and missing values

Missing values were discussed in the variables section of this document. There it was noted that the "_FillValue" attribute is a special attribute that allows for specific elements of an array to be tagged as missing. When any NCL expression is being evaluated, NCL ignores elements that are equal to the value of the "_FillValue" attribute for each variable. When a missing value is ignored, the result of the expression will contain a missing value at the corresponding array index. If more than one term in an expression contains a missing value and the values are not equal, the missing value of the value of the left-most term in the expression containing a missing value is used in the output. The following example illustrates this.

```
;
; Assign three variables values and missing values. Each variable's
; missing value is different.
;
        a = (/1, 2, -99/)
        a@_FillValue = -99
        b = (/4, -999, 5/)
        b@_FillValue = -999
        c = (/-9999, 7, 8/)
        c@_FillValue = -9999

;
; Assignment to variable d of an expression containing all
; three variables with different missing values.
;
        d = a * b * c

;
; Assignment to variable of an expression containing two
; variables with different missing values.
        e = b * c

;
; Results of first assignment. Note resulting missing value.
;
        print(d)

        Variable: d
        Type: integer
        Total Size: 12 bytes
                    3 values
        Number of Dimensions: 1
        Dimensions and sizes:   [3]
        Coordinates:
        Number Of Attributes: 1
          _FillValue :  -99
        (0)     -99
        (1)     -99
        (2)     -99

;
; Results of second assignment. Note resulting missing value.
;
        print(e)

        Variable: e
        Type: integer
        Total Size: 12 bytes
                    3 values
        Number of Dimensions: 1
        Dimensions and sizes:   [3]
        Coordinates:
        Number Of Attributes: 1
          _FillValue :  -999
        (0)     -999
        (1)     -999
        (2)     40
```

Types of expressions

Algebraic

Algebraic expressions operate on arrays of basic numeric types. There are nine main algebraic operators: multiply, divide, exponent, plus, minus, modulus, less than selection, greater than selection, and negative. All of the operators except negative take two operands. Also, in general, both operands should be the same type. If they are not, NCL attempts to coerce them to identical types. If this fails, a type mismatch error is generated. Additionally, either the operands must have the same number of dimensions and dimension sizes, or one operand can be a scalar value. The operators operate on the entire array of data.

The following is a list of the operator characters. The operators are listed in order of precedence: the first one has the highest precedence. The precedence rules can be circumvented by using parentheses '( )' around expressions that should be computed first.

`-`	Negative
`^`	Exponent
`*` `/` `%` `#`	Multiply Divide Modulus Matrix multiply
`+` `-`	Plus Minus
`<` `>`	Less than selection Greater than selection

Negative -

The negative operator negates the value of its operand. The operand can be any numeric type. If it is an array, each element of the array is negated.

Exponent ^

The left operand is "raised" to the power of the right operand. If the left operand is negative, then the right side must be positive and greater-than-or-equal to 1.0. Currently, NCL doesn't support imaginary numbers; it generates an error in this situation. The result type of this operator is always a floating point number unless double precision values are used.

Multiply *

Multiplies left operand by right operand. No special conditions or restrictions.

Divide /

Divides left operand by right operand. When both operands are integer types, the result is an integer type. Therefore, the decimal remainder is truncated.

Matrix Multiply

The operands must be one or two dimensional arrays. Higher dimensionality is not supported. This operand computes the dot product of two single dimension arrays. For 2 D arrays the 1st dimension of the left operand must be the same size as the 0th dimension of the right. The output dimensionality will be the [0th dimension of the left] x [1st dimension of the right].

Modulus %

The result is the integer remainder of integer division. Both operands must be integer types.

Plus +

Adds left operand to right operand. No special conditions or restrictions. If the operands are strings, then the strings are concatenated.

Minus -

Subtracts right operand from left operand. No special conditions or restrictions.

Less-than selection <

For arrays, the less-than selection operator selects all values in the left operand that are less than the corresponding value in the right operand. If the value of the left side is greater than or equal to the corresponding value of the right side, then the right side value is placed in the result. For a scalar and an array, the same selection rules apply, but the scalar is compared against each value in the array operand.

Greater-than selection >

For arrays, the greater-than selection operator selects all values in the left operand that are greater than the corresponding value in the right operand. If the value of the left side is less than or equal to the corresponding value of the right side, then the right side value is placed in the result. For a scalar and an array, the same selection rules apply, but the scalar is compared against each value in the array operand.

Logical

Logical expressions are formed by relational operators. There are ten relational operators:

less-than-or-equal, greater-than-or-equal, less-than, greater-than, equal, not-equal, and, or, exclusive-or, not. All of these operators yield logical values, in other words arrays of **True**, **False** or missing values. And, or, exclusive-or, and not require logical operands, and the rest accept any type.

All of the logical operators function similar to algebraic operators with respect to operations between arrays and arrays, arrays and scalars, and scalars and scalars. For array operands, each corresponding element is compared, and the result is an array of logical values or missing values, the size and dimensionality of the operands. A single scalar can be compared against each element in an array. The result is an array of logical values the size of the array operand. Finally, when two scalars are compared, the result is a scalar logical value.

The following is a list of the logical operators ordered by their precedence.

```
.le.    Less-than-or-equal
.lt.    Less-than
.ge.    Greater-than-or-equal
.gt.    Greater-than
.ne.    Not-equal
.eq.    Equal
.and.   And
.xor.   Exclusive-or
.or.    Or
.not.   Not
```

Less-than-or-equal .le.

Returns True if the left operand is less than or equal to the right operand.

Less-than .lt.

Returns True if the left operand is less than the right operand.

Greater-than-or-equal .ge.

Returns True if the left operand is greater than or equal to the right operand.

Greater-than .gt.

Returns True if the left operand is greater than the right operand.

Not-equal .ne.

Returns True if the left operand is not equal to the right operand.

And .and.

Requires both operands to be logical types. Returns True if and only if both operands are True.

Or .or.

Requires both operands to be logical types. Returns True if either operand is True.

Exclusive-or .xor.

Requires both operands to be logical types. Returns True if one of the operands is True and the other is False.

Not .not.

Takes a single operand which must be logical. Returns True if the operand is False; returns False if the operand is True.

Arrays

An array expression combines either scalar values or other arrays into a single array result. An array expression is made up of expressions separated by commas ’,’ and enclosed in ’(/’ and ’/)’ (called array designator characters). Arrays can be made up of any of the basic types as well as graphical objects. Currently, arrays of files are not supported. Each element separated by commas must have the same dimensionality as the rest of the elements in the array expression. If the types are not the same, the coercion rules are used to convert all of the array elements to the same type. If this fails, an error message is generated.

The following are examples of array expressions.

```
(/ 1, 2, 3, 4, 5 /)
(/ (/ 1, 2, 3 /)^2, (/ 4, 5, 6 /)^3,(/ 7, 8, 9 /)^4 /)
(/ a - b, b + c, c / d /)
```

If a single variable is enclosed in array designator characters ’(/’ and ’/)’, only the variable’s value is referenced and not its associated attributes, dimension names, and coordinates. The array designator characters are useful in performing value only assignment.

Functions

Functions are expressions because they return a value. Functions do not necessarily have to always return the same type or size array every time they are called. Functions are called by referencing their name and providing an argument list. Arguments can be any type, including files. They can also be constant values, variables, or subsections of variables.

Functions can be defined to restrict parameters to be specific types, to have specific numbers of dimensions, and to have specific dimension sizes. If a parameter must be a certain type, then the coercion rules are applied to it.

One very important thing to note is that all parameters are **passed by reference** in NCL, meaning a change of a value in a variable within the function changes the value of the parameter.

However, if a parameter is coerced before the function is called, changes within the function ***will not*** be reflected in the parameter, because coercion can only be applied in a single direction. A warning message is given in this instance.

Expression results can also be passed to functions as parameters. However, since expression results are not referenced by a variable, there is no way that changes made to the parameter can be passed out from the function, unless it is returned by the function.

NCL statements

Statements are the fundamental element of NCL. Everything NCL does happens only after a statement is entered. Statements are not restricted to being a single line of source, and statements can be nested within statements. There are 17 different kinds of statements: assignment, procedure call, function definition, procedure definition, block, do, if-then, if-then-else, break, continue, setvalues, getvalues, return, record, new, stop, and quit.

There are two very important things to understand about statements. First, each statement only executes after it has been parsed and is found to be free of syntax errors. Second, if a runtime error occurs while executing a statement, execution of the statement and any statements in which the erroneous statement is nested is immediately terminated.

Blocks

Blocks provide a way to group a list of commands. Since blocks are statements, the statements within the begin and end do not get executed until the end statement is parsed and the source is determined to be free of syntax errors. Scripts enclosed in a block are parsed and, if there are no syntax errors, executed. When using NCL by either loading scripts or piping scripts into NCL, begin and end blocks should be used. This reduces error cascading, and syntax errors can be found before costly statements preceding the error are executed.

begin
    statement list
end

if statements

The are two kinds of **if** statements: the **if-then** statement and the **if-then-else** statement. These function like if statements in other popular programming languages. The following is the syntax for the NCL if statements:

if(*scalar_logical_expression*) then
    statement list
end if
if(*scalar_logical_expression*) then
    statement list
else
    statement list

end if

The *scalar_logical_expression* is an expression made of relational operators. For *if* statements, the result of the logical expression within the parentheses **must result in a single scalar value** and **must not be a missing value**.

Important note: never search for a missing value using a logical expression. For example if statements of the following form should not be used:

```
if( a(i) .eq. a@_FillValue) then
        . . .
```

The above statement will return a **missing value** when *a* is equal to the attribute "_FillValue". This will in turn cause an error message and premature termination of the script. In NCL when expressions, even logical expressions, contain a missing value the missing value is propagated to the result. See the discussion on Expressions and missing values.

Logical operators operate on entire arrays of values, when this is the case they produce array results, and each element in the array result could be True, False, or a missing value. Since *if* statements require the results of the logical expression to be a single scalar logical value a FATAL error message is generated and execution is terminated if the conditional part of the *if* statement is not scalar. Therefore there are three functions that help with the above problems. The functions *any* and *all* return a single True scalar value if any element in a logical array is True, or if all elements in a logical array are True, respectively. The function *ismissing* returns an array of True or False values at locations where the input is either a missing value or a normal value Combined with lazy conditional expression evaluation, the ismissing function can filter possible error conditions before they occur.

Lazy expression evaluation means that if the first term of an .AND. expression is False, or the first term of an .OR. expression is True, the entire expression--regardless of the result of the remaining terms--is False and True respectively. When this is the case, NCL does not evaluate the remaining terms. This is a very useful tool for avoiding error conditions. For example, consider the integer *i* and the array *a*. Lazy evaluation can avoid indexing the array *a* with *i* when *i* is out-of-range. The following example demonstrate this:

```
if((i .lt. dimsizes(a)) .and. (a(i) .gt. 10 )) then
. . .
end if

if(.not.ismissing(a(i)) .and. (a(i) .gt. 10)) then
. . .
end if
```

Loops

NCL provides two kinds of do loops: a while loop that repeats until its *scalar_logical_expression* evaluates to False, and a traditional do loop that loops from a start value through an end value incrementing or decrementing a loop variable either by one or by a specified stride. With all kinds of loops, the keywords *break* and *continue* can be used. *Break* causes the loop to abort, and *continue* causes the loop to proceed directly to the next iteration without executing the remaining statements in the block.

do

For the do loop, an identifier, a scalar integer start expression, a scalar integer end expression, and an optional positive scalar integer stride expression are required. The identifier can be either a) a defined variable or variable subsection or b) an undefined name. Before the loop starts, the *start_expr* value is assigned to the identifier. Then after each iteration of the loop, either the identifier is incremented or the identifier is assigned its next value based on the stride specified and the direction. The following is an example of a loop without specification of the stride. In this kind of loop, the identifier is incremented by one after each iteration. If the start expression is greater than the end expression, no iterations of the loop occur. If the start expression is equal to the end, then one iteration occurs. To loop backward, see the next example.

do *identifier* = *scalar_start_expr* , *end_expr*
    statement list
end do

The following shows how to specify a stridden loop. The following form is also necessary to loop backwards. With this form, if start is greater than end, then the identifier is decremented by the stride value, which must always be positive. If the start is less than the end, then the identifier is incremented by the stride value.

do *identifier* = *scalar_start_expr* , *scalar_end_expr* , *scalar_stride_expr*
    statement list
end do

while

This example shows the while loop syntax. The *scalar_logical_expression* adheres to the same restrictions placed on the conditional expression of the if statement. Specifically, it must be a scalar logical value and not be a missing value. Also lazy evaluation of .AND. and .OR. statements occurs.

do while (*scalar_logical_expression*)
    statement list
end do

Assignment

The assignment statement can be used to assign values of any type to variables, subsections of variables, coordinate variables, and attributes. In addition, the assignment statement is used to assign string values to dimension names. Several NCL language constructs are used to identify what type of assignment is to take place. The constructs '->', '!', '&', and '@' are used to specify file variable assignment, dimension name assignment, coordinate variable assignment, and attribute assignment, respectively. Without these constructs, normal-value-to-variable assignment occurs.

The following passages cover the syntax of the various assignment statements. The section on variable assignment covers the semantics of variable-to-variable assignment. Variable-to-variable assignment copies not only a variable's value but all attributes, dimensions, and coordinate variables associated with that variable. The variable assignment section as well as the properties of variables are a very important sections to read and understand.

Dimension name assignment

To assign a dimension name to a defined variable or file variable, the variable name is followed by the '!' operator followed by the dimension number to which the new name is to be assigned. Examples:

```
a!0 = "Dimension1"
thefile->a!0 = "Dimension1"
```

Coordinate variable assignment

To associate a coordinate variable with a dimension of a variable, the variable name is followed by the '&' operator, followed by the name of the dimension to which the coordinate variable is to belong. Three restrictions apply to coordinate variables. First, the dimension to which the coordinate variable is being assigned must have a name. Second, the value being assigned must be a single dimension array of the same size as the named dimension it is being associated with. Finally, if coordinate is going to be used, the values in the coordinate variable must be monotonically increasing or decreasing. If the value assigned is non-monotonic or contains missing values, the assignment still occurs but a WARNING message is generated and attempts to use coordinate subscripting will fail with a FATAL message. Examples:

```
a&Dimension1 = (/.1,.2,.3,.4,.5,.../)
thefile->a&Dimension1 = (/.1,.2,.3,.4,.5,.../)
```

Using string references:

```
dimname = "Dimension1"
a&$dimname$ = (/.1,.2,.3,.4,.5,.../)
```

Using file string references:

```
dimname = "Dimension1"
varname = "a"
thefile->$varname$&$dimname$ = (/.1,.2,.3,.4,.5,.../)
```

Attribute assignment

To assign an attribute to a variable, the variable name is followed by the '@' operator, followed by the name of the attribute. Attributes must have a single dimension but can be any size. Examples:

```
a@units = "Degrees C"
a@_FillValue = -9999.0
thefile->a@units = "Degrees C"
```

Using string references:

```
attnames = (/"units","_FillValue"/)
a@$attnames(0)$ = "Degrees C"
a@$attnames(1)$ = -9999.0
```

Using file string references:

```
attnames = (/"units","_FillValue"/)
varname = "a"
thefile->$varname$&$attnames(0)$ = "Degrees C"
thefile->$varname$&$attnames(1)$ =  -9999.0
```

The _FillValue attribute is a special reserved attribute. It allows the user to associate a missing value with a variable. This allows values to be filtered when the array of values is an operand to an algebraic expression. When set, all of the missing values in the value array referenced by the variable are changed to the new missing value.

Procedures

NCL procedures, in many ways, are the same as in most programming languages, but NCL procedures also have some distinct differences. The key differences are in how the types and dimension sizes for parameters are specified and handled by NCL. In NCL, parameters can be specified to be very constrained and require a specific type, number of dimensions, and a dimension size for each dimension, or parameters can have no type or dimension constraints. Parameters in NCL are always **passed by reference**, meaning changes to their values, attributes, dimension names, and coordinate variables within functions change their values outside of the functions. There is one very important exception that does generate a WARNING message: when an input parameter must be coerced to the correct type for the procedure, NCL is not able to coerce data in the reverse direction so the parameter is not affected by changes made to it within the procedure. See the discussion on coercion.

Parameters that are subsections of variables (i.e. subscripted elements) are passed by reference. Which means when the procedure finishes, the values of the subsection are mapped back into their original locations in the referenced variable.

External Procedures

External procedures are procedures that exist in external shared libraries. These procedures are loaded during execution of the external statement. To call one of these procedures or functions, you preceded the procedure name and argument list with the name provided to the **external** statement followed by two colons. For example:

```
external elib "./elib.so"

. . .

elib::eproc(param0,param1)
```

Function and procedure definitions

Functions and procedures are defined using a similar syntax; the only difference is that the keyword

procedure is used instead of the keyword **function**. To define a new function, the proper keyword is entered, followed by the symbol name of the function, followed by a list of declarations that can optionally be followed by a list of local variable names. Finally, a block of statements enclosed in **begin** and **end** follows.

> function *function_name* (*declaration_list*)
> local *local_identifier_list*
> begin
>   *statement list*
>   return(*return_value*)
> end

The local statement and function and procedure scope

It is very important to note that if a variable intended to be local to the function or procedure does not occur in the local list, then the function may find that variable name outside of the scope of the function or procedure.

It is also important to note the consequences of not putting a local variable into the local list. There are two possibilities. First, if the variable is defined **at the time of function definition** and is in the function's scope, the variable will be referenced. If the variable is not defined at the time of definition, it will be an undefined variable local to the function.

Placing the local variable's name in the local list assures that the variable truly is local only to the current function. The scope rules for NCL are identical to Pascal. The main NCL prompt is the bottom level of NCL's scope. Function and procedure blocks are the next level. When a variable is referenced in a function, NCL first looks within the scope of the function for that variable. If it doesn't exist, NCL looks in the scope containing the function and continues to search outward until either the bottom level is reached or a variable of the correct name is found. If it is not found, then an undefined variable reference error message is generated.

Constraining the type and dimensionality of input parameters

As noted previously, parameters can be constrained or unconstrained. The following shows the syntax of the variable declaration. The square brackets ('[' and ']') are used to denote optional parts of the declaration syntax and are not part of the structure of the declaration list. Each declaration in the declaration list is separated by a comma.

Declaration list element syntax:

> *variable_name* [*dimension_size_list*] [: *data_type*]

Unlike the preceding example syntax, the characters '[' and ']' are part of the syntax for the dimension size list. They are used to separate dimension sizes. Each pair of brackets can contain an integer representing the size of the dimension or a star to indicate that the dimension may be any size. The number of pairs of brackets is the number of dimensions the parameter will be constrained to.

Dimension size list syntax:

[*dimension_size*]
or
[*]

Local variable list syntax:

identifier_name , *identifier_name* , . . .

The local variable list is very important. If a local variable name is not in the local list and the same identifier is defined in a lower scope level, then the variable reference in the function or procedure will refer to that variable. Sometimes this may be the desired effect. However, if the variable is to be local, it should be referenced in the local list.

Visualization blocks

Visualization blocks are the interface between the HLU library and NCL. There are three types: **create, setvalues, and getvalues**. These set or retrieve HLU resources from HLU objects. Creating graphics with the HLUs can range from very simple to very complex. Only the syntax for the interface is discussed here. More information on HLU usage as well as NCL examples can be found at the following locations:

- HLU user guide
- HLU reference guide
- Pick a utility page
- NCL scientific tutorial

create

The **create** visualization block is actually an expression; it is described here for consistency. The **create** block returns an id to a newly created HLU object. HLU resources are assigned using the syntax for the NCL resource list. In NCL resource lists, resource names are entered by their string representation followed by a colon (':') and an NCL expression. The **create** block requires a string name for the HLU object, which can be different than the name of the variable it is being assigned to. Create also requires a HLU object class identifier that tells NCL what type of HLU object to **create**. Finally **create** either needs a reference to the object's parent, usually a workstation, or the keyword "defaultapp" to signify that the object's parent is the default application class. Refer to the HLU user guide and reference guide for a discussion of HLU objects and application classes.

obj = create *obj_name obj_class parent*
    resource_string : *value_expression*
end create

The following is a list of all the HLU class identifiers recognized by NCL:

- annoManagerClass
- appClass
- contourPlotClass
- coordArraysClass

- irregularPlotClass
- graphicStyleClass
- labelBarClass
- legendClass
- logLinPlotClass
- mapPlotClass
- psWorkstationClass
- scalarFieldClass
- streamlinePlotClass
- textItemClass
- tickMarkClass
- titleClass
- ncgmWorkstationClass
- vectorFieldClass
- vectorPlotClass
- xWorkstationClass
- xyPlotClass

setvalues

Use **setvalues** to modify resources in an HLU object after it has been created. **setvalues** takes an NCL resource list in the same fashion that **create** does. It then sets each resource to its corresponding value. Note: *obj_reference* can be one or more HLU objects. If more than one the resources are repeatedly applied to each HLU object.

setvalues *obj_reference*
resource_string : *value_expression*
end setvalues

getvalues

Use **getvalues** to retrieve resource values from HLU objects. It is different than the **setvalues** and **create** NCL resource list because it requires an identifier rather than an expression after the colon. **getvalues** functions identically to the assignment with respect to how it assigns values to the identifier.

getvalues *obj_reference*
resource_string : *identifier*
end getvalues

load a shared library

The **external** command loads a specially created shared library. Shared libraries are one way in which NCL can be extended to call custom C and Fortran functions and procedures. The syntax for the **external** command is:

external *string_name path_name*

The process for creating these shared libraries is described in the extending NCL section. The syntax for calling functions and procedures defined in external shared libraries is described in the procedures section of this page. The rules for call procedures also apply to functions.

load script from a file

The load command is used to load external scripts from files. These external scripts can either be sets of functions and procedures or actual NCL programs. Currently, *file_path* must be a constant value; string expressions will generate syntax error messages. You can think of the load command as being analogous to the C #include statement. The following is an example:

> load "*file_path*"

record commands

The record command will cause all syntactically correct commands entered at the command line to be saved in the file pointed to by the *file_path* constant string value. The "stop record" command terminates this process.

> record "*file_path*"
> ...
> *statements*
> ...
> stop record

NCL API

In addition to functioning as a command line interpreter, NCL has an API for submitting NCL scripts from within an application. The API provides GUI and application writers a uniform way to do data access, manipulation, and visualization from within their application using NCL scripts. This should simplify the implementation of application-specific GUIs by reducing design time spent on the visualization code and the data access code.

There are several functions and a special NCL library needed to use the API. The name of the library is libnclapi.a. It should be located in the same directory that NCAR Graphics is installed in. To use the API, you must link to this library and use the following functions. NclInitServer and NclCloseServer must be called at the beginning and end of the application respectively.

Data structures

The following are descriptions of the data structures returned by the NCL API functions.

NclApiScalar

```
typedef union _NclApiScalar {
```

```
        double  doubleval;
        float   floatval;
        int     intval;
        long    longval;
        short   shortval;
        char    charval;
        string  stringval;
        byte    byteval;
        logical logicalval;
        obj     objval;
}NclApiScalar;
```

NclExtValueRec

```
struct _NclExtValueRec {
        int type;
        int constant;
        void *value;
        NclApiScalar missing;
        int elem_size;
        int totalelements;
        int n_dims;
        int dim_sizes[NCL_MAX_DIMENSIONS];
};

typedef struct _NclExtValueRec NclExtValueRec;
```

NclApiDataList

```
struct  _NclApiDataList {
        int kind;
        struct _NclApiDataList *next;
        union {
                struct _NclApiFuncInfoRec *func;
                struct _NclApiVarInfoRec *var;
                struct _NclApiFileInfoRec *file;
                struct _NclApiHLUVarInfoRec *hlu;
                struct _NclApiHLUObjInfoRec *hlu_obj;
        } u;
};

struct _NclApiHLUObjInfoRec {
        NclQuark name;
        int obj_id;
        NhlClass obj_class;
};

struct _NclApiHLUVarInfoRec {
        NclQuark name;
        int     n_objs;
        struct _NclApiHLUInfoRec* objs;
};

struct _NclApiHLUInfoRec {
        NclQuark obj_name;
        NclQuark obj_class;
        int     obj_id;
};
```

```
struct _NclApiFuncInfoRec {
        NclQuark name;
        int     kind;
        int     nparams;
        struct _NclArgTemplate *theargs;
};

struct _NclApiVarInfoRec {
        NclQuark name;
        NclQuark data_type_quark;
        NclVarTypes     type;   /* value, hlu, file */
        int     n_dims;
        NclDimRec *dim_info;
        int     n_atts;
        NclQuark        *attnames;
};

struct _NclApiFileInfoRec {
        NclQuark name;
        NclQuark path;
        int file_type;
        int wr_status;
        int     n_dims;
        NclDimRec *dim_info;
        int     n_atts;
        NclQuark *attnames;
        int     n_vars;
        NclQuark *var_names;
};
```

NclCloseServer
: Closes the connection to the NCL server.

NclFreeDataList
: Frees the data list records returned by several of the API functions.

NclFreeExtValue
: Frees value records returned by several of the API functions.

NclGetDelHLUObjsList
: Returns a list of HLU objects that have been deleted since the last call to this function.

NclGetErrorId
: Returns the HLU error object id being used by the NCL server.

NclGetExprValue
: Evaluates and returns the result of an NCL expression.

NclGetFileList
: Returns a list of currently opened files and information about their contents.

NclGetFileVarsList
: Returns a list of variable names and information for a given file.

NclGetHLUObjId
: Returns the id of an HLU object that is referenced by an NCL variable.

NclGetHLUObjsList
: Returns a list of all HLU objects and related information referenced by NCL variables.

NclGetNewHLUObjsList
: Returns a list of all HLU objects created since the last call to this function.

NclGetProcFuncList
: Returns a list of functions and parameter types for all functions currently defined in the NCL

interpreter.

NclGetVarList
: Returns a list of variables and their related information for all variables currently defined in NCL.

NclGetVarValue
: Returns a record containing the value of data reference by a variable.

NclInitServer
: Must be called at the beginning of the program to initialize NCL.

NclPrintErrorMsgs
: Prints out buffered error messages.

NclSubmitBlock1
: Submits a block of NCL source that is a contiguous block of text.

NclSubmitBlock2
: Submits a block of NCL commands that are each individual lines of source.

NclSubmitCommand
: Submits a single NCL command.

NCL API Functions

NclCloseServer

Synopsis

```
#include <ncarg/ncl/defs.h>
#include <ncarg/ncl/ApiRecords.h>
#include <ncarg/ncl/NclApi.h>

void NclCloseServer(
        void
);
```

Description

Must be called to shut down the NCL server.

NclFreeDataList

Synopsis

```
#include <ncarg/ncl/defs.h>
#include <ncarg/ncl/ApiRecords.h>
#include <ncarg/ncl/NclApi.h>

void NclFreeDataList (
```

```
        NclApiDataList *the_list;
);
```

Arguments

the_list
A pointer to the NclApiDataList to be freed.

Description

Frees the pointer to an NclApiDataList that was returned by one of the other API functions.

NclFreeExtValue

Synopsis

```
#include <ncarg/ncl/defs.h>
#include <ncarg/ncl/ApiRecords.h>
#include <ncarg/ncl/NclApi.h>

void NclFreeExtValue(
        NclExtValueRec *val
);
```

Arguments

val
Val is a pointer to an NclExtValueRec that was returned by one of the other API functions.

Description

Frees the data pointed to by *val*.

NclGetDelHLUObjsList

Synopsis

```
#include <ncarg/ncl/defs.h>
#include <ncarg/ncl/ApiRecords.h>
#include <ncarg/ncl/NclApi.h>

NclApiDataList *NclGetDelHLUObjsList(
        void
);
```

Description

Returns a list of HLU objects that have been deleted since the last call to this function.

NclGetErrorId

Synopsis

```
#include <ncarg/ncl/defs.h>
#include <ncarg/ncl/ApiRecords.h>
#include <ncarg/ncl/NclApi.h>

int NclGetErrorId(
        void
);
```

Description

Returns the HLU error object id being used.

NclGetExprValue

Synopsis

```
#include <ncarg/ncl/defs.h>
#include <ncarg/ncl/ApiRecords.h>
#include <ncarg/ncl/NclApi.h>

NclExtValueRec *NclGetExprValue(
        char * expression
);
```

Arguments

expression
: A string representing a valid NCL expression.

Description

This function submits the string *expression* to NCL for evaluation. The result of the expression is returned as a pointer to an NclExtValueRec. If the evaluation of the expression fails, NULL is returned. The errors generated by the expression string can be accessed using NclPrintErrorMsgs.

NclGetFileList

Synopsis

```
#include <ncarg/ncl/defs.h>
#include <ncarg/ncl/ApiRecords.h>
#include <ncarg/ncl/NclApi.h>

NclApiDataList *NclGetFileList(
        void
);
```

Description

Returns a list of currently referenced files and information about the file: the name, path, file type, read/write status, number of file dimensions, the names of all the dimensions, file attributes, and all of the file variable names.

NclGetFileVarsList

Synopsis

```
#include <ncarg/ncl/defs.h>
#include <ncarg/ncl/ApiRecords.h>
#include <ncarg/ncl/NclApi.h>

NclApiDataList* NclGetFileVarsList(
        NclQuark filevar
);
```

Arguments

filevar
An NclQuark representation of a variable name.

Description

Returns a list the variable information for all the file variables in the file *filevar* quark. This information includes the variable name, variable type, dimension sizes, dimension names, and attribute names. If the file does not exist, NULL is returned.

NclGetHLUObjId

Synopsis

```
#include <ncarg/ncl/defs.h>
#include <ncarg/ncl/ApiRecords.h>
#include <ncarg/ncl/NclApi.h>

int NclGetHLUObjId(
        char *var_name
```

```
);
```

Arguments

var_name
: Name of variable referencing an HLU object.

Description

Returns the HLU integer object id associated with a variable name.

NclGetHLUObjsList

Synopsis

```
#include <ncarg/ncl/defs.h>
#include <ncarg/ncl/ApiRecords.h>
#include <ncarg/ncl/NclApi.h>

NclApiDataList *NclGetHLUObjsList(
        void
);
```

Description

Returns a list of all the HLU objects currently referenced by NCL symbols.

NclGetNewHLUObjsList

Synopsis

```
#include <ncarg/ncl/defs.h>
#include <ncarg/ncl/ApiRecords.h>
#include <ncarg/ncl/NclApi.h>

NclApiDataList *NclGetNewHLUObjsList(
        void
);
```

Description

Returns a list of HLU objects that have been created since a previous call to this function.

NclGetProcFuncList

Synopsis

```
#include <ncarg/ncl/defs.h>
#include <ncarg/ncl/ApiRecords.h>
#include <ncarg/ncl/NclApi.h>

NclApiDataList *NclGetProcFuncList(
        (void)
);
```

Description

Returns the list of currently defined functions and procedures as well as information about parameter types and dimension sizes.

NclGetVarList

Synopsis

```
#include <ncarg/ncl/defs.h>
#include <ncarg/ncl/ApiRecords.h>
#include <ncarg/ncl/NclApi.h>

NclApiDataList *NclGetVarList(
        void
);
```

Description

Returns the list of currently defined variables, their types, dimensionality, dimension names, attribute names, and symbol table entries.

NclGetVarValue

Synopsis

```
#include <ncarg/ncl/defs.h>
#include <ncarg/ncl/ApiRecords.h>
#include <ncarg/ncl/NclApi.h>

NclExtValueRec *NclGetVarValue(
        char* var_name,
        int copy_data
);
```

Arguments

var_name

The name of the variable whose data are to be returned.
copy_data

True or False value. Specifies whether the data referenced by the variable should be copied before it is returned.

Description

This function returns the data portion of an NCL variable. The data portion is the actual multidimensional array and its dimensionality.

NclInitServer

Synopsis

```
#include <ncarg/ncl/defs.h>
#include <ncarg/ncl/ApiRecords.h>
#include <ncarg/ncl/NclApi.h>

int     NclInitServer(
        NhlErrorTypes error_level
);
```

Arguments

error_level

Specifies the error level to be reported.

Description

Initializes the NCL server. This must be called before any HLU calls or calls to any of the other functions listed here. The error level parameter allows the user to set what level of messages will be printed by NclPrintErrorMsgs.

NclPrintErrorMsgs

Synopsis

```
#include <ncarg/ncl/defs.h>
```

```
#include <ncarg/ncl/ApiRecords.h>
#include <ncarg/ncl/NclApi.h>

void NclPrintErrorMsgs(
        void
);
```

Description

Prints out the currently buffered error messages to whatever the errFileName or errFilePtr resources are set to.

NclSubmitBlock1

Synopsis

```
#include <ncarg/ncl/defs.h>
#include <ncarg/ncl/ApiRecords.h>
#include <ncarg/ncl/NclApi.h>

int NclSubmitBlock1(
        char* script,
        int script_size
);
```

Arguments

script

A single block of memory containing one or more '\n'-terminated lines of NCL source.
script_size

Size of the block of memory.

Description

Submits a script for execution to the NCL server. If the return value is 0, the script failed and error messages may need to be printed out using the NclPrintErrorMsgs function. A nonzero number means the script completed correctly.

NclSubmitBlock2

Synopsis

```
#include <ncarg/ncl/defs.h>
```

```
#include <ncarg/ncl/ApiRecords.h>
#include <ncarg/ncl/NclApi.h>

int NclSubmitBlock2(
        char *script[]
);
```

Arguments

script

An array of NULL terminated lines of NCL source.

Description

Functions similar to the NclSubmitBlock1 function call with the exception that the input script is an array of strings rather than a block of text. For hard-coded script, this is the ideal form to submit scripts. For files, however, it is easier to use the NclSubmitBlock1.

NclSubmitCommand

Synopsis

```
#include <ncarg/ncl/defs.h>
#include <ncarg/ncl/ApiRecords.h>
#include <ncarg/ncl/NclApi.h>

int NclSubmitCommand(
        char *command;
);
```

Arguments

command

A single NCL statement.

Description

Works like both the block functions, except only a single statement is submitted.

Extending the NCL function and procedure set

Writing your own wrapper

NCL has been designed to allow users the ability to extend the built-in default function set. This functionality requires a fair amount of C and Fortran knowledge. In addition, if you intend to call a Fortran function, some specific knowledge of how to call Fortran from C is needed. With release 4.1 extending NCL to call FORTRAN subroutines and functions has been greatly simplified. At release 4.1 the **external statment** was added to support loading of specially created shared libraries. Also a utility called **wrapit77** was added to generate all the C code necessary to build an NCL shared library.

If you want to build your own custom version of NCL which includes your functions and procedures, the following steps should be followed. However, if you want to enable the installed version of NCL to call you functions and procedures using the **external** statement then follow the instructions for writing a wrapper, if you're calling a C function, and then follow the directions for building a shared library. If you're calling FORTRAN then follow all the steps for using wrapit77

The general approach for extending the NCL function set is as follows:

- Write a function or procedure in C or Fortran.
- Write a wrapper function that calls the internal NCL function NclGetArgValue to retrieve parameters from NCL and then calls the actual function. The wrapper function can also call the internal function NclReturnValue to pass a return value back to NCL if necessary.
- Write a C function called NclAddUserFuncs and use SetArgTemplate with either NclRegisterFunc or NclRegisterProc to assign an NCL function name to the user function.
- Compile and link files containing the actual function wrapper, the wrapper function, and NclAddUserFuncs with the library libncl.a to create a new executable of NCL.

The NCL binaries installed on your system should contain a library called "libncl.a". To determine if this library has been installed on your system, run NCL and enter the following command:

```
ncl 0> system("ls " + ncargpath("lib"))
```

If libncl.a is not one of the libraries listed by the above command, contact your site administrator.

This section uses an example to demonstrate how to extend the function set. In this example we will write a function that evaluates a polynomial given a value and arrays of polynomial coefficients. The intended NCL function interface is:

```
function poly(
        a[*]:float,
        b[*]:float,
        c[*]:float,
        x[1]:float
)
```

Writing the function in C or Fortran

There are only a few restrictions necessary for C and Fortran functions. First and most important, the C function, Fortran function, or Fortran subroutine name must not conflict with any function names in the

NCL library. Use the UNIX function "nm" to accomplish this. Second, for C, none of the data passed in from the wrapper can be freed. Finally, for Fortran, if I/O is being done you must make sure that the unit number being used is not already in use. It is also important to follow good programming practice by freeing any memory allocated by the function unless it is being passed back as a return value, and closing all files opened by the function. Also, only the internal functions specified on this page should be used.

The following is the C source for the function poly in a file called poly.c :

```
void poly_C(float *outval, float *a, float *b, float *c, float x, int n) {
        int i;

        for(i = 0; i < n ; i++) {
                outval[i] = (a[i] * x * x) + (b[i] * x) + c[i];
        }
}
```

Writing the wrapper function

The wrapper function must be written in C and use the function NclGetArgValue to retrieve the values of the parameters from NCL. The interface for the NclGetArgValue provides information as to the number of dimensions, dimension sizes, the missing values if any, and the type. The prototype for NclGetArgValues is located in "NclBuiltInSupport.h". The interface for NclGetArgValue follows:

```
void *NclGetArgValue(
        int arg_num,                    /* IN:  argument number to retrieve
        int n_args,                     /* IN:  total number of arguments */
        int* n_dims,                    /* OUT: number of dimensions of para
        int* dimsizes,                  /* OUT: dimension sizes of paramete
        NclScalar* missing,             /* OUT: missing value if any */
        int *has_missing,               /* OUT: whether or not a missing va
                                             present in the parameter */
        NclBasicDataTypes *type,        /* OUT: data type of parameter */
        int access_type                 /* IN:  Either 0 for don't care, 2
);
```

The input argument *arg_num* is the number of the argument requested. The arguments are numbered from left to right (as they appear in NCL) starting at 0 through n-1 where n is the total number of arguments to the function. The input argument *n_args* is the total number of arguments for the NCL function. The final input parameter is *access_type*. *Access_type* tells NCL whether or not the parameter being accessed will be modified or not. This is necessary to support NCL's pass-by-value parameter passing scheme.

The actual data associated with the parameter is returned as a void pointer by the NclGetArgValue function which must be cast to the appropriate type. The parameters *n_dims* and *dimsizes* represent the rank of the parameter. The type parameter is set to one of the following basic data types. NclBasicDataTypes is defined in "NclDataDefs.h".

```
typedef enum  {
        NCL_none =      0,
        NCL_short =     01,
        NCL_int =       02,
        NCL_long =      04,
```

```
        NCL_float =     010,
        NCL_double =    020,
        NCL_char =      040,
        NCL_byte =      0100,
        NCL_string =    0200,
        NCL_numeric =   0400,
        NCL_logical =   01000,
        NCL_obj =       02000
} NclBasicDataTypes;
```

The contents of the *has_missing* parameter is set to 1 if the parameter possibly contains missing values. There are some situations where it is possible to have *has_missing* set and not actually have a missing value in the data. This occurs when a subsection of a variable, which has missing values, is subscripted and the portion subscripted does not actually contain missing values. However, if the *has_missing* parameter is set to 0, then there are no missing values. The value of the missing value is returned in the union NclScalar. NclScalar is defined in "NclDataDefs.h". It is a union of all the primitive types in NCL.

```
typedef union _NclScalar {
        double  doubleval;
        float   floatval;
        int     intval;
        long    longval;
        short   shortval;
        char    charval;
        string  stringval;
        byte    byteval;
        logical logicalval;
        obj     objval;
}NclScalar;
```

Finally, return values are passed back to the NCL environment via the NclReturnValue. NclReturnValue provides an interface for passing back data. It currently does not support passing back coordinate arrays or attributes, with the exception of missing values.

The prototype for the NclReturnValues function is:

```
NhlErrorTypes NclReturnValue(
void *value,                   /* Pointer to data to be returned */
int n_dims,                    /* Number of dimensions of the data */
int* dimsizes,                 /* Dimension sizes of the data */
NclScalar* missing,            /* If non-NULL is the missing value for the (
                                  otherwise NULL if no missing values in dat
NclBasicDataTypes type,        /* Ncl basic type identifier */
int copy_data                  /* True if data should be copied before retu:
);
```

One major goal of designing the wrapper function should be to establish a scheme for handling missing values if the function you are adding does not already handle them. In the following example, missing values are filtered out and propagated through to the output.

The source for the wrapper function follows:

```
#include <stdio.h>
/*
* The following are the required NCAR Graphics include files.
* They should be located in ${NCARG_ROOT}/include
```

```
*/
#include <ncarg/hlu/hlu.h>
#include <ncarg/hlu/NresDB.h>
#include <ncarg/ncl/defs.h>
#include <ncarg/ncl/NclDataDefs.h>
#include <ncarg/ncl/NclBuiltInSupport.h>

/*
* Declare C function version
*/
extern void poly_C(float *outval, float *a, float *b, float *c, float x, int n);

NhlErrorTypes poly_W( void )
{

        float *a;
        int n_dims_a,dimsizes_a[NCL_MAX_DIMENSIONS],has_missing_a;
        NclScalar missing_a;
        float *b;
        int n_dims_b,dimsizes_b[NCL_MAX_DIMENSIONS],has_missing_b;
        NclScalar missing_b;
        float *c;
        int n_dims_c,dimsizes_c[NCL_MAX_DIMENSIONS],has_missing_c;
        NclScalar missing_c;
        float *x;
        NclScalar missing_x;
        float *output_val;
        NclScalar tmp_missing;
        int has_missing_x;
        int i;

/*
* Retrieve parameters
*/

/*
* Note any of the pointer parameters can be set to NULL, which
* implies you don't care about the its value. In this example
* the type parameter is set to NULL because the function
* is later registered to only accept floating point numbers.
* Also the parameter x is later registered to be a single
* dimension floating point value so dimension sizes and the
* number of dimensions are not needed.
*/
        a = (float*)NclGetArgValue(
                0,
                4,
                &n_dims_a,
                dimsizes_a,
                &missing_a,
                &has_missing_a,
                NULL,
                2
        );
        b = (float*)NclGetArgValue(
                1,
                4,
                &n_dims_b,
                dimsizes_b,
                &missing_b,
                &has_missing_b,
```

```
                NULL,
                2
        );
        c = (float*)NclGetArgValue(
                2,
                4,
                &n_dims_c,
                dimsizes_c,
                &missing_c,
                &has_missing_c,
                NULL,
                2
        );
        x = (float*)NclGetArgValue(
                3,
                4,
                NULL,
                NULL,
                &missing_x,
                &has_missing_x,
                NULL,
                2
        );

/*
* This is the only dimension size check needed since the function
* is registered to only accept single dimension parameters.
*/
        if((dimsizes_a[0] == dimsizes_b[0])&&(dimsizes_b[0] == dimsizes_c[0]))
/*
* The case where x is a missing value should be handled. One option
* is to print an error message and return NhlFATAL. However a better
* option is to return an array of size n_dims_a filled with missing
* values.
*/
                if((has_missing_x) && (missing_x.floatval == *x)) {
                        output_val = (float*)malloc(sizeof(float)*dimsizes_a[0]
                        for(i = 0; i < n_dims_a; i++) {
                                output_val[i] = *x;
                        }
                        return(NclReturnValue(
                                (void*)output_val,
                                n_dims_a,
                                dimsizes_a,
                                &missing_x,
                                NCL_float,
                                0
                        ));
                }
/*
* The following section allocates the output memory and calls the
* poly function
*/
                output_val = (float*)malloc(sizeof(float)*dimsizes_a[0]);
                poly_C(output_val,a,b,c,*x,dimsizes_a[0]);
/*
* A scheme for handling missing values in the input. NCL's algebra
* handles missing value by propagating the missing value of the left
* most term in an expression to the result whenever any term contains
* a missing value.
*/
```

```
                if(has_missing_a || has_missing_b || has_missing_c) {
/*
* Chooses suitable missing value by checking from left to right for
* missing values.
*/
                        if(has_missing_a) {
                                tmp_missing.floatval == missing_a.floatval;
                        } else if (has_missing_b) {
                                tmp_missing.floatval == missing_b.floatval;
                        } else {
                                tmp_missing.floatval == missing_c.floatval;
                        }
/*
* Checks for missing values in the input and changes output value to missing
* value if any of the terms contain missing values
*/
                        for(i = 0; i < dimsizes_a[0]; i++) {
                                if((has_missing_a)&&(a[i] == missing_a.floatval)
                                        output_val[i] = tmp_missing.floatval;
                                }
                                if((has_missing_b)&&(b[i] == missing_b.floatval)
                                        output_val[i] = tmp_missing.floatval;
                                }
                                if((has_missing_c)&&(c[i] == missing_c.floatval)
                                        output_val[i] = tmp_missing.floatval;
                                }
                        }
/*
* Returns value to NCL
*/
                        return(NclReturnValue(
                                (void*)output_val,
                                n_dims_a,
                                dimsizes_a,
                                &tmp_missing,
                                NCL_float,
                                0
                        ));
                } else {
/*
* Returns value to NCL
*/
                        return(NclReturnValue(
                                (void*)output_val,
                                n_dims_a,
                                dimsizes_a,
                                NULL,
                                NCL_float,
                                0
                        ));
                }
        } else {
                NhlPError(NhlFATAL,NhlEUNKNOWN,"poly: the dimension sizes of pa
                return(NhlFATAL);
        }

}
```

Writing NclAddUserFuncs to register the function with NCL

Note: If you intend to create a shared library for use with either NCL_DEF_LIBRARY_DIR or the external statment follow these instructions but name the function **Init** rather than **NclAddUserFuncs**. Also if you intend to call a FORTRAN function or subroutine the utility wrapit77 can be used in place of this step in the wrapper creation.

The next step in extending the NCL function set is to write a function called NclAddUserFuncs. This function takes no arguments and does not return anything. Its purpose is to allow the user to call NclRegisterFunc and NclRegisterProc to tell NCL the NCL string name of the function, what the function pointer to call is, how many parameters the function takes, and what the type and dimensionality of the parameters are.

There are three C functions that must be called to register a C wrapper function as as an NCL built-in function. For each function being registered in NclAddUserFuncs, the function NewArgs must be called to initialize a private array that will hold the descriptions of each parameter. NewArgs takes a single integer argument that is the number of arguments for the function to be registered. It returns a void pointer that must be used in subsequent calls to SetArgTemplate and NclRegisterFunc.

The next function called is SetArgTemplate. It must be called for each parameter to the function. SetArgTemplate takes the pointer returned by NewArgs, the current parameter number, the parameter's type, number of dimensions, and dimension sizes as input. SetArgTemplate does not return a value; it simply modifies the private array returned by NewArgs.

Once each parameter has been defined by SetArgTemplate, then either the function NclRegisterFunc or NclRegisterProc can be called depending on whether the C function being added is meant to be an NCL function or procedure. The prototypes for these four functions are located in the include file NclBuiltIns.h, which should be located in ${NCARG_ROOT}/include/ncarg/ncl:

```
void *NewArgs(
int  n                        /* Total number of parameters to function being :
);

void SetArgTemplate(
        void*   args,           /* Array returned by NewArgs */
        int     arg_num,        /* Argument number being defined */
        char*   type_name,      /* string representing type, set to NclANY if ar
                                   type is accepted */
        int     n_dims,         /* number of dimensions of argument, set to
                                   NclANY if any number of dimensions supported
        int*    dimsizes        /* dimension size for each dimension, set to Nc:
                           if all dimension sizes are supported */
);

extern void NclRegisterFunc(
        NclPubBuiltInProcWrapper thefuncptr,      /* The C wrapper function */
        void* args,                               /* The array returned by NewA:
                                                     by successive calls to Set/
        char* fname,                              /* String name of function to
        int n_args                                /* Total number of arguments :
);
```

The following is the source for the function NclAddUserFuncs. More than one user function can be registered from within this function.

```
#include <stdio.h>
#include <ncarg/hlu/hlu.h>
#include <ncarg/hlu/NresDB.h>
#include <ncarg/ncl/defs.h>
#include <ncarg/ncl/NclBuiltIns.h>

/*
* Declare wrapper function
*/
extern NhlErrorTypes poly_W(
#if NhlNeedProto
void
#endif
);

void NclAddUserFuncs
#if NhlNeedProto
(void)
#else
()
#endif
{
        void *args;
        int dimsizes[NCL_MAX_DIMENSIONS];
        int nargs = 0;
/*
* Create private argument array
*/
        args = NewArgs(4);
/*
* Configure first three parameters identically as single dimension float
* arrays of any size
*/
        SetArgTemplate(args,nargs,"float",1,NclANY);nargs++;
        SetArgTemplate(args,nargs,"float",1,NclANY);nargs++;
        SetArgTemplate(args,nargs,"float",1,NclANY);nargs++;
/*
* Configure fourth parameter as single scalar floating point value
*/
        dimsizes[0] = 1;
        SetArgTemplate(args,nargs,"float",1,dimsizes);nargs++;
/*
* Register wrapper function pointer  and argument templates
*/
        NclRegisterFunc(poly_W,args,"poly",nargs);

        return;
}

/*
* NOTE: the following function stubs must be included with the
* source for function NclAddUserFuncs. Future releases of this
* documentation will describe their uses. For now though NCL
* WILL NOT compile WITHOUT them.
*/

void NclAddUserFileFormats
#if NhlNeedProto
(void)
#else
()
```

```
#endif
{
        return;
}

void NclAddUserHLUObjs
#if       NhlNeedProto
(void)
#else
()
#endif
{
        return;
}
```

Compiling and linking files

Note: at the 4.1 release new functionality was added to support loading shared libraries to extend NCL. The directions for compiling a shared library can be found in the section titled Using wrapit77 to generate wrappers. This is the recommended method of extending NCL since a new version of NCL does not need to be created.

The final step is to compile and link the source files created with libncl.a and the other NCAR Graphics libraries. The simplest way to compile these functions into an executable is to use the nhlcc script from the UNIX prompt. nhlcc should be located in ${NCARG_ROOT}/bin. Nhlcc is a script that will link in the appropriate HLU, LLU, and other support libraries. The compile line for this example using the nhlcc command is:

```
nhlcc -o ncl ${NCARG_ROOT}/lib/libncl.a AddUserFuncs.c poly_w.c poly.c -L${NCAR(
```

AddUserFuncs.c contains the function NclAddUserFuncs and the function stubs NclAddUserFileFormats and NclAddUserHLUObjs, poly_w.c contains the wrapper function, and poly.c contains the actual function source. The extra libraries on the end are for netCDF, HDF, and lex respectively, which are used by NCL. The following is a list of all libraries needed in the order they should linked in to compile line:

```
libhlu.a              HLU library

libncarg.a            NCAR Graphics LLU library

libncarg_gks.a        NCAR Graphics GKS library

libncarg_c.a          NCAR Graphics common code

libl.a                System LEX library

libnetcdf.a           NetCDF library

libX11.a              X11 library

libdf.a               HDF library

libF77.a              NOTE: This library is part of the C-Fortran interface l:
                      might be different for different architectures consult t
                      programmers manual for the machine you are using.
```

```
libI77.a                NOTE: This library is part of the C-Fortran interface l:
                        might be different for different architectures consult t
                        programmers manual for the machine you are using.

libU77.a                NOTE: This library is part of the C-Fortran interface l:
                        might be different for different architectures consult t
                        programmers manual for the machine you are using.

libm.a                  System Math library

libc.a                  C library
```

Writing your own wrapper as a shared library

To create a wrapper and compile it as a shared library the steps of writing a C or Fortran function or procedure and writing a wrapper function are the same. However, instead of creating a C function called NclAddUserFuncs you must create a C function name "Init" which contains the same calls as NclAddUserFuncs. Then you must compile each piece of source code and link them together as a shared library.

Using wrapit77 to generate wrappers

With release 4.1 a utility application named *wrapit77* is included with the release. This utility facilitates generating a C source code wrapper for Fortran 77 functions and subroutines. Below is a step by step description of how to use *wrapit77*. Starting with a Fortran procedure and ending with a functionally equivalent NCL procedure is a five-step process:

Step 1
: Create code blocks that describe the Fortran procedure and its arguments. These code blocks are similar to Fortran 90 interface blocks (although the blocks created here must use Fortran 77 syntax), and C function prototypes.

Step 2
: Create an NCL wrapper function by running the program **wrapit77** that is supplied with the NCAR Graphics Software (version 4.1 or later). **wrapit77** uses the code blocks in Step 1 as input.

Step 3
: Create object modules (".o" files) for the NCL wrapper function generated in Step 2 and for the original Fortran source.

Step 4
: Create a dynamic shared object from the object files created in Step 3 using the UNIX **ld** command. A dynamic shared object is an object module whose entries are loaded on an as-needed basis.

Step 5
: Inform NCL where to locate the dynamic shared object created in Step 4.

Consider each of the above steps in detail:

Step 1 - describe your procedure

A *wrapit interface block* is a sequence of Fortran 77 statements that specifies a procedure and its arguments. These statements are bracketed by Fortran comment lines:

```
C NCLFORTSTART
```

at the start and

```
C NCLEND
```

at the end. These comment lines are known as *wrapit interface block delimiters*. Any blanks in the above two lines are ignored and the lines are case-insensitive.

The following is an example of a wrapit interface block for a Fortran function that finds the arc length of a polyline:

```
C NCLFORTSTART
      FUNCTION ARCLN(NUMPNT, POINTX, POINTY)
      DIMENSION POINTX(NUMPNT), POINTY(NUMPNT)
C NCLEND
```

The statements between wrapit interface block delimiters should contain only a procedure statement and non-executable declarations describing its arguments. Any other statements (except comment statements) in a wrapit interface block will cause **wrapit77** to exit with an error. Unless specifically typed, all variables are typed according to Fortran default typing.

Fortran 90 codes can use **wrapit77** to generate an NCL wrapper function if you can provide a wrapit interface block for it, i.e. provided that you can specify the Fortran 90 procedure interface with Fortran 77 syntax. In some cases this will not be possible, for example if you are using pointers or structures as arguments, if the Fortran 90 procedure can have missing arguments, if the procedure is recursive, or if you are using keyword arguments. Many times, however, it is easy to convert Fortran 90 syntax to Fortran 77. For example, the Fortran 90 statements:

```
      CHARACTER (LEN=*)::NAME
      REAL,INTENT(IN)::NUMBER
```

could be specified as

```
      CHARACTER*(*) NAME
      REAL NUMBER
```

in the wrapit interface block.

Note: **wrapit77** scans its input file for wrapit interface blocks, so if you can easily arrange your Fortran codes so that the wrapit interface blocks are embedded in them, it may be more convenient for you to use your original Fortran source file as input to **wrapit77**, rather than use a separate file containing only wrapit interface blocks. See example 2 in this section for details on this.

Step 2 - create an NCL wrapper function by running wrapit77

wrapit77 scans its input for wrapit interface blocks and produces an NCL wrapper function as output. Any source code in the input to **wrapit77** that lies outside of a wrapit interface block is ignored, so the input to **wrapit77** can simply be a sequence of wrapit interface blocks, or it can be complete Fortran source with embedded wrapit interface blocks.

For example, if your input wrapit interface block is in file `wrapper_input`, then executing

```
wrapit77 < wrapper_input >! wrapper_W.c
```

will create the NCL wrapper function `wrapper_W.c`. See the examples in this section for sample uses of **wrapit77**. You can name your NCL wrapper function anything you want, but the usual convention is to append the "_W" followed by ".c" (since the wrapper function is C code).

Step 3 - create object modules

Object modules (".o" files) need to be created for your original Fortran source and for the NCL wrapper function created in Step 2. The NCL wrapper function is C source and contains some special include files. You should use the **nhlcc** command to create the ".o" file for the NCL wrapper function, since it will correctly locate the include files. For example, if you have named your NCL wrapper function `wrapper_W.c`, then execute

```
nhlcc -c wrapper_W.c
```

to create `wrapper_W.o`. For consistency with the use of **nhlcc**, you may want to use the command **nhlf77** to create the ".o" file for your Fortran source, but any way is fine.

Step 4 - create a dynamic shared object module

Dynamic shared objects are object modules whose entries are loaded on an as-needed basis. The UNIX command **ld** is used to create the dynamic shared objects (".so" files) used with NCL. The object modules (".o" files) you created in Step 3 will be used to create your dynamic shared object. For each of the supported systems (except for Crays, where **wrapit77** is not currently supported), the requirements are different. Here is a summary of what you need to do on each system (where the NCL wrapper function object code is named `fcode_W.o` and the object for the original Fortran is named `fcode.o`) to create a dynamic shared object named `fcode.so`:

- DEC Alpha running Digital UNIX 4.0:

```
nhlcc -c fcode_W.c
nhlf77 -c fcode.f
ld -shared -o fcode.so fcode_W.o fcode.o
```

- HP 700 series running HPUX 10.20:

```
nhlcc +z -c fcode_W.c
nhlf77 +z -c fcode.f
ld -b -o fcode.so fcode_W.o fcode.o
```

- Pentium PC running Linux RedHat 5.0:

```
nhlcc -c fcode_W.c
nhlf77 -c fcode.f
```

```
ld -shared -o fcode.so fcode_W.o fcode.o
```

- SGI running IRIX 6.2 (-64):

```
nhlcc -c fcode_W.c
nhlf77 -c fcode.f
ld -64 -shared -o fcode.so fcode_W.o fcode.o
```

- SGI running IRIX 6.2 (-n32):

```
nhlcc -c fcode_W.c
nhlf77 -c fcode.f
ld -n32 -shared -o fcode.so fcode_W.o fcode.o
```

- Sun running Solaris 2.x:

```
nhlcc -c fcode_W.c
nhlf77 -c fcode.f
ld -o fcode.so fcode_W.o fcode.o -G
```

Step 5 - inform NCL where to locate your dynamic shared object

NCL gets its information about externally defined procedures from dynamic shared objects, as created in Step 4.

Once you have created your dynamic shared object containing the NCL wrapper function and the object for the original code, there are three ways to tell NCL where to look for the dynamic shared object:

1. Use an NCL `external` statement.
2. Use the system environment variable LD_LIBRARY_PATH.
3. Use the NCL environment variable NCL_DEF_LIB_DIR.

Accessing dynamic shared objects using an `external` statement

In this case, you insert a statement of the form:

external *SO_NAME* "*path_name*"

at the beginning of your NCL script. In this statement *SO_NAME* indicates the name you want the dynamic shared object to be known by in the NCL program and *path_name* is a relative or full pathname of the dynamic shared object file you created. The final argument to the `external` statement is an NCL string variable, so the quotes are required. The case where *path_name* is simply the name of the dynamic shared object itself (and *not* a pathname) is special and is treated in the "Accessing dynamic shared objects using LD_LIBRARY_PATH" section.

Once the dynamic shared object has been accessed and named, you can call your new NCL function or procedure with an NCL statement of the form:

```
SO_NAME::fortran_name(arguments)
```

where *SO_NAME* is the name that you gave the dynamic shared object in your NCL `external` statement and *fortran_name* is the name of the Fortran function or subroutine that you specified in your wrapit interface block. *fortran_name* must be entered as either all uppercase, or all lowercase, regardless of what case the original Fortran procedure name was. Using an NCL name that is all uppercase or lowercase will produce the same results. For example, if your original Fortran procedure was named "CalcX", then you can use either "CALCX" or "calcx" to invoke that function from NCL, but you cannot use "CalcX" or any other mixed-case form of the name. Most NCL programmers tend to use lowercase for procedure names. See the examples in this section for complete examples of using dynamic shared objects and calling your new NCL procedures.

Accessing dynamic shared objects using LD_LIBRARY_PATH

In this case, you insert a statement of the form:

```
external SO_NAME "shared_object_name"
```

at the beginning of your NCL script. In this statement, *shared_object_name* is the name of your dynamic shared object file. NCL will look in the directories listed in the environment variable `LD_LIBRARY_PATH` for the specified dynamic shared object. If it does not find it there, an error will be reported. `LD_LIBRARY_PATH` is a system environment variable and is a colon-separated list of directory pathnames. In attempting to set this environment variable, make sure that it is not already set for use by other applications; if it is, just add the pathname of the dynamic shared object to it.

Once NCL has located the dynamic shared object, you can invoke the function or procedure in the same way that was discussed above under "Accessing dynamic shared objects using an `external` statement."

Accessing dynamic shared objects using NCL_DEF_LIB_DIR

If the NCL environment variable `NCL_DEF_LIB_DIR` is defined and is a valid directory pathname, NCL will look in that directory first for dynamic shared objects (you can have as many dynamic shared objects in this directory as you want). Any entry in any dynamic shared object in the directory specified by `NCL_DEF_LIB_DIR` will be loaded automatically as an NCL built-in. In particular this means that, in order to invoke your new function or procedure, you *should not* prepend *SO_NAME::* to the function or procedure name as was required in either of the two other methods for accessing dynamic shared objects discussed above. NCL will try to load *everything* in the directory specified by `NCL_DEF_LIB_DIR`, so you will get warning messages about files in the directory that are not shared objects.

This approach can be more convenient than the other two approaches, but it lacks the advantage of having the *SO_NAME::* prepended to externally defined procedures which makes their externally defined status quickly recognized from the syntax.

The environment variables `NCL_DEF_LIB_DIR` and `LD_LIBRARY_PATH` are independent. The dynamic shared objects in the directory specified by `NCL_DEF_LIB_DIR` are loaded before any dynamic shared object specified on an external statement, whether that dynamic shared object is specified by way of a relative or direct pathname or by way of a directory specified using the `LD_LIBRARY_PATH` environment variable.

Examples

This section contains examples illustrating how to incorporate sample Fortran codes into NCL. Example 1 is of particular importance, since it illustrates in detail all of the steps required to incorporate Fortran procedures into NCL.

- Example 1 -- Fortran subroutine and function
- Example 2 -- embedded wrapit interface blocks
- Example 3 -- subroutine from a commercial library
- Example 4 -- subroutine with CHARACTER input and output arguments
- Example 5 -- subroutine with a 2-dimensional array; printing

Example 1 -- Fortran subroutine and function

Begin with the Fortran source (in file `ex01.f`):

```
      SUBROUTINE CQUAD (A, B, C, NQ, X, QUAD)
      REAL X(NQ),QUAD(NQ)
C
C  Calculate quadratic polynomial values.
C
      DO 10 I=1,NQ
        QUAD(I) = A*X(I)**2 + B*X(I) + C
   10 CONTINUE
C
      RETURN
      END
      FUNCTION ARCLN (NUMPNT, POINTX, POINTY)
      DIMENSION POINTX(NUMPNT),POINTY(NUMPNT)
C
C  Calculate arc lengths.
C
      IF (NUMPNT .LT. 2) THEN
        PRINT *, 'ARCLN: Number of points must be at least 2'
        STOP
      ENDIF
      ARCLN = 0.
      DO 10 I=2,NUMPNT
        PDIST = SQRT((POINTX(I)-POINTX(I-1))**2 +
     +                        (POINTY(I)-POINTY(I-1))**2)
        ARCLN = ARCLN + PDIST
   10 CONTINUE
C
      RETURN
      END
```

This first example follows in detail the five-step process described above.

Step 1 - define the wrapit interface block.

Create a file `ex01.wib` that contains the following two wrapit interface blocks:

```
C NCLFORTSTART
      SUBROUTINE CQUAD (A,B,C,NQ,X,QUAD)
```

```
      REAL X(NQ),QUAD(NQ)
C NCLEND

C NCLFORTSTART
      FUNCTION ARCLN (NUMPNT,POINTX,POINTY)
      DIMENSION POINTX(NUMPNT),POINTY(NUMPNT)
C NCLEND
```

Step 2 - create an NCL wrapper function.

Execute the following:

```
wrapit77 < ex01.wib >! ex01_W.c
```

to create the NCL wrapper function `ex01_W.c`.

Step 3 - create the object modules (the ".o" files).

Execute the following:

```
nhlcc -c ex01_W.c
nhlf77 -c ex01.f
```

to create `ex01_W.o` and `ex01.o` (on an HP system you will need to add a "`+z`" flag to each of the above commands).

Step 4 - create a dynamic shared object module.

For this, use the appropriate **ld** command listed in the section above discussing this. For example, if you are running on a Sun Solaris machine, then you would execute:

```
  ld -o ex01.so ex01_W.o ex01.o -G
```

to create the dynamic shared object `ex01.so`.

Step 5 - tell NCL where your dynamic shared object is.

The external statement in the following example script tells NCL where to look for the dynamic shared object you just created.

```
external EX01 "./ex01.so"

begin
;
; calculate three values of a quadratic equation
;
   nump = 3
   x = (/ -1., 0.0, 1.0 /)
   qval = new(nump,float)
   EX01::cquad(-1, 2, 3, nump, x, qval) ; call the new NCL version of
                                        ; your original Fortran subroutine
   print("Polynomial value = " + qval)

;
```

```
; calculate an arc length.
;
   xc = (/ 0., 1., 2. /)
   yc = (/ 0., 1., 0. /)
   arclen = EX01::arcln(nump,xc,yc)     ; call the new NCL version of
                                        ; your original Fortran function
   print("Arc length = " + arclen)

end
```

If you submit the above script to the NCL interpreter, it produces the output:

```
opening: ./ex01.so
(0)     Polynomial value = 0
(1)     Polynomial value = 3
(2)     Polynomial value = 4
(0)     Arc length = 2.82843
```

The numbers in parentheses at the left in the above printout are an artifact of how the NCL **print** function works and are of no relevance in this example.

Example 2 -- embedded wrapit interface blocks

Instead of starting with the Fortran source as in example 1 above, you could have used the following code (in file ex02.f) as input to **wrapit77**:

```
C NCLFORTSTART
      SUBROUTINE CQUAD (A,B,C,NQ,X,QUAD)
      REAL X(NQ),QUAD(NQ)
C NCLEND
C
C  Calculate quadratic polynomial values.
C
      DO 10 I=1,NQ
        QUAD(I) = A*X(I)**2 + B*X(I) + C
   10 CONTINUE
C
      RETURN
      END

C NCLFORTSTART
      FUNCTION ARCLN (NUMPNT, POINTX, POINTY)
      DIMENSION POINTX(NUMPNT),POINTY(NUMPNT)
C NCLEND
C
C  Calculate arc lengths.
C
      IF (NUMPNT .LT. 2) THEN
        PRINT *, 'ARCLN: Number of points must be at least 2'
        STOP
      ENDIF
      ARCLN = 0.
      DO 10 I=2,NUMPNT
        PDIST = SQRT((POINTX(I)-POINTX(I-1))**2 +
     +                        (POINTY(I)-POINTY(I-1))**2)
        ARCLN = ARCLN + PDIST
   10 CONTINUE
C
```

```
      RETURN
      END
```

In this example, the wrapit interface blocks are embedded directly into the Fortran code, thus avoiding the need to create a separate file containing them. All that **wrapit77** is looking for in its input is blocks delimited by comment lines containing `NCLFORTSTART` and `NCLEND`.

Now execute:

```
wrapit77 < ex02.f >! ex02_W.c
```

and proceed as in steps 3-5 of Example 1.

Example 3 -- subroutine from a commercial library

Suppose you are calling:

```
      SUBROUTINE LIBSUB(ARG1)
```

from a commercial library named `commercial.a`. Using the following wrapit interface block (in file `libsub.wib`):

```
C NCLFORTSTART
      SUBROUTINE LIBSUB(ARG1)
C NCLEND
```

Create an NCL wrapper function and its object module by executing:

```
wrapit77 < libsub.wib > libsub_W.c
nhlcc -c libsub_W.c
```

To create a dynamic shared object named `libsub.so`, use the appropriate **ld** command by loading `libsub_W.o` and `commercial.a`. For example, if you are running on a Sun Solaris machine, you would execute:

```
  ld -o libsub.so libsub_W.o commercial.a -G
```

to create the dynamic shared object `libsub.so`.

Example 4 -- subroutine with CHARACTER input and output arguments

Start with a Fortran subroutine that takes an input string and returns a number of letters based on the length of the input string:

```
      SUBROUTINE EX04 (STRIN,STROUT)
      CHARACTER*(*) STRIN
      CHARACTER*26  ABET,STROUT
      DATA ABET/'ABCDEFGHIJKLMNOPQRSTUVWXYZ'/
C
      IMX = MIN(LEN(STRIN),26)
      STROUT = ABET(1:IMX)
C
      RETURN
```

```
      END
```

You could use embedded wrapit interface blocks, as in example 2 above, or use the following wrapit interface block:

```
C NCLFORTSTART
      SUBROUTINE EX04 (STRIN,STROUT)
      CHARACTER*(*) STRIN
      CHARACTER*26  STROUT
C NCLEND
```

to create the NCL wrapper and dynamic shared object (named `ex04.so`). Passing the following NCL script to the NCL interpreter:

```
external EXAMPLE04_SO "./ex04.so"

begin

 cstr = new(26,character)        ;  create a character array of length 26
 EXAMPLE04_SO::ex04("fifteen letters",cstr)
 str = chartostring(cstr)
 print(str)

end
```

produces the output:

```
opening: ./ex04.so


Variable: str
Type: string
Total Size: 4 bytes
            1 values
Number of Dimensions: 1
Dimensions and sizes:   [1]
Coordinates:
(0)     ABCDEFGHIJKLMNO
```

Example 5 -- subroutine with a 2-dimensional array; printing

Start with one subroutine that calculates a function of two variables and stores the results in a 2-dimensional array, and another subroutine that prints 2-dimensional arrays by rows.

```
      SUBROUTINE EX05(M,N,X,Y,FXY)
      REAL X(M),Y(N),FXY(M,N)
C
C  Calculate FXY(I,J) = 2*I+J
C
      DO 10 J=1,N
        DO 20 I=1,M
          FXY(I,J) = 2.*REAL(I) + REAL(J)
   20   CONTINUE
   10 CONTINUE
C
      RETURN
      END
```

```
      SUBROUTINE PRT2D(M,N,A)
      REAL A(M,N)
C
C  Print the array A by rows using an F6.1 format with
C  7 values per line.
C
      DO 10 J=1,N
        PRINT *,'Row',J,':'
        DO 20 I=1,M/7
          WRITE(6,500) (A(LL,J),LL=(I-1)*7+1,I*7)
  500     FORMAT(7F6.1)
   20   CONTINUE
        IF (MOD(M,7) .NE. 0) WRITE(6,500) (A(LL,J),LL=(M/7)*7+1,M)
        PRINT *,' '
   10 CONTINUE
C
      RETURN
      END
```

Use the following wrapit interface block:

```
C NCLFORTSTART
      SUBROUTINE EX05(M,N,X,Y,FXY)
      REAL X(M),Y(N),FXY(M,N)
C NCLEND
C NCLFORTSTART
      SUBROUTINE PRT2D(M,N,A)
      REAL A(M,N)
C NCLEND
```

to create the NCL wrapper function and the dynamic shared object (named `ex05.so`). Then the following NCL script:

```
external EX05 "./ex05.so"

begin
;
; calculate three values of a quadratic equation
;
   m = 11
   n = 3
   x = new(m,float)
   y = new(n,float)
   fxy = new((/n,m/),float)
   EX05::ex05(m,n,x,y,fxy)
   EX05::prt2d(m,n,fxy)
end
```

will create the 2-dimensional array `fxy` in a manner compatible with other NCL procedures. Passing the above NCL script to the NCL interpreter produces the output:

```
opening: ./ex05.so
 Row  1:
    3.0   5.0   7.0   9.0  11.0  13.0  15.0
   17.0  19.0  21.0  23.0

 Row  2:
    4.0   6.0   8.0  10.0  12.0  14.0  16.0
   18.0  20.0  22.0  24.0
```

```
 Row  3:
     5.0    7.0    9.0   11.0   13.0   15.0   17.0
    19.0   21.0   23.0   25.0
```

Special considerations

This section contains several things that you should know to avoid common problems.

Array dimensioning:

For NCL arrays, the fastest-varying dimension is the rightmost, while for Fortran it is the leftmost dimension. Therefore, if `XA` is a Fortran array dimensioned `idim` x `jdim`, this array will be dimensioned `jdim` x `idim` in NCL. Also, Fortran array subscripts start at 1, whereas NCL array subscripts start at 0. Example 5 in this section illustrates these concepts.

Arrays of character strings:

Currently **wrapit77** honors only non-dimensioned Fortran type CHARACTER variables. You cannot pass arrays of NCL strings to Fortran, nor can you pass Fortran CHARACTER arrays from Fortran back to NCL.

Passing strings from NCL to Fortran:

If you want to pass an NCL variable of type string to a Fortran procedure, then the argument to the Fortran procedure *must* be declared as `CHARACTER*(*)`. See example 4 in this section.

Passing Fortran CHARACTER variables to NCL:

If you want to pass a Fortran CHARACTER variable back to NCL, then the Fortran argument must be a variable of type `CHARACTER` of fixed length, and the corresponding NCL variable must be a *character array* of the same length. If you want to use the NCL character array as an NCL string, you will need to use the NCL conversion function **chartostring**. See example 4 in this section.

Complex numbers:

NCL does not have a complex data type. If you want to bring complex numbers into NCL, you will have to do it by bringing in the real and imaginary parts as separate arrays. This will most likely require that you write an interface subroutine to your Fortran code that splits up the Fortran COMPLEX numbers into real and imaginary parts. Although you will not be able to do arithmetic on the complex numbers in NCL, you can still do analysis on the real and imaginary parts separately.

Procedure name conflicts:

If the procedure that you are incorporating into NCL has the same name as a currently existing built-in NCL procedure, NCL will choose its built-in and not your procedure. However, most UNIX **ld** commands recognize the `-B symbolic` flag, and using it when you create your dynamic shared object will force NCL to load your procedure in preference to its own built-ins. The `-B symbolic` can cause **ld** to report missing entries that in fact are ultimately not missing. It is probably best just to be careful to avoid defining a procedure with the same name as an NCL built-in.

NCL termination:

If a Fortran procedure that you have incorporated into NCL executes a STOP statement, or if a C function executes an `exit` statement, then the NCL interpreter will abort.

Unsupported Fortran 77 syntax in wrapit interface blocks:

Fortran COMMON blocks:

Fortran COMMON blocks are not allowed in a wrapit interface block. This would preclude your having adjustable arrays whose dimensions are passed in a COMMON block, or using COMMON to pass values for variables.

Fortran ENTRY statements:

There is no way to accommodate an ENTRY statement in a Fortran procedure.

Alternate return arguments:

Subroutines with alternate return arguments are not allowed.

begin keyword

The *begin* keyword is used in the following NCL statement types:

```
Blocks
Function and Procedure Definitions
```

---

break keyword

The *break* keyword can be used in the following NCL statement types:

```
Do
While
If
```

---

continue keyword

The *continue* keyword can be used in the following NCL statement types:

```
Do
While
If
```

---

end keyword

The *end* keyword is used in the following NCL statement types:

```
Do
While
If
create
getvalues
setvalues
Blocks
Function and procedure definitions
```

Usage help page

This page provides helpful hints for using NCL efficiently and effectively. This is not a complete and comprehensive usage FAQ, but it lists some very important issues that users should be aware of when using NCL. A general working knowledge of the NCL syntax is required for this page.

For some NCL usage examples, see the NCL scientific tutorial and the "pick-a-utility" section of the Quick Start Guide.

Writing efficient NCL source

There are many features in NCL that when not used -- or used improperly -- result in inefficient data processing. Many of these problems result from the fact that NCL is an interpreted language and must do additional processing for each statement, subscript, and expression executed. Therefore a good rule of thumb is to strive to reduce the number of statements, subscripts, and expressions executed.

Removing loops

The most important fundamental to remember is that NCL is more efficient when NCL’s array operations are used. Array operations mean an entire array is used rather than a subscripted element of the array. Consider multiplying two 2-dimensional floating point arrays dimensioned 100x100. One way would be to loop from 0-99 for both dimensions and subscripting individual elements of each array, multiplying them together and assigning them to a result.

```
do i = 0,99
  do j = 0,99
    c(i,j) = a(i,j)*b(i,j)
  end do
end do
```

Although intuitive for most programmers, this is the least efficient way to do this in NCL. All of NCL's algebraic operators allow arrays of similar dimensions and types to be used as operand. For example, the above loop could be rewritten as follows:

```
c = a*b
```

By counting each each statement, subscript, and expression in the original loop, you would get three subscripts, one statement, and one expression. These values must be multiplied by 10000 since there are that many iterations, bringing the totals to 30000 subscripts, 10000 statements, and 10000 expressions. The second example has one statement and one expression. By removing unnecessary loops and using NCL's array syntax in NCL source, very large performance gains can be made.

Removing scalar expressions

Sometimes it is impossible to completely remove loops. When this is the case, attention should be focused on removing scalar expressions or unnecessary expressions and statements inside of loops. Consider the following example:

```
do i = 0,999
  do j = 0,999
    T(i,j) = 100.0 - 8 * sqrt(i^2 + j^2)
  end do
end do
```

Using the counting scheme outlined above, this loop has one subscript, one statement, and six expressions (the function sqrt is counted as an expression). These must be multiplied by 1000000 iterations. Now, using a little algebra and knowing that the operators and functions can accept entire arrays as well as individual elements, the above loops can be rewritten as:

```
do i = 0,999
  do j = 0,999
    T(i,j) = i^2 + j^2
  end do
end do
T = 100 - 8 * sqrt(T)
```

This version of the loops contains one subscript, one statement, and three expressions times 1000000 iterations. The additional line adds merely three total expressions, making the total just slightly over half the original. In this example, operations common to all elements of the array T were moved outside the loop. By the way, the above loops are not actually necessary. They can be completely removed by initializing T using ispan. This is left as an exercise for the reader.

Use built-in functions and procedures when applicable

NCL has many time-saving functions and procedures that operate on entire arrays. Before using loops to compute values, check the built-in function set to see if something is available. The following are some of the most useful time-saving functions.

- ismissing
- ndtooned and onedtond
- mask
- max, min, avg, sum, product, stddev, variance
- ind, num, any, all
- dimsizes, filevardimsizes

Know when memory is allocated

Memory allocation in NCL occurs during a *new* call, assignment, subscripting, and expression evaluation. In the cases of subscripting and expression evaluation, this memory is temporary and NCL will either free it or reuse it as efficiently as possible.

Avoid using *new* to create variables unless it is absolutely necessary. Consider the following source:

```
T = new(filevardimsizes(file1,"T"),float)
. . .
T = file1->T
```

This causes an extra array the size of variable T to be unnecessarily allocated. The file variable reference, *file1->T*, needs to allocate an array the size of T in order to read it from the file. At this point there are two arrays. During the assignment, one array is copied to the other and then freed. This is unnecessary since NCL will implicitly define variables when they appear on the left side of an assignment. Therefore the above statements should be written as:

```
T = file1->T
```

This way the temporary value allocated by the file read is merely assigned to the variable T. In this way only one allocation occurs. One example of when this is unavoidable is when reading from multiple files into one array. The following is an example of that:

```
filenames = (/"a.nc","b.nc","c.nc","e.nc"/)
file1 = addfile(filenames(0),"r")
dims = filevardimsizes(file1,"T")
T = new((/dimsizes(filenames),dims(0),dims(1),dims(2)/),float)
T(0,:,:,:) = file1->T
do i = 1,dimsizes(filenames)-1
        file1 = addfile(filenames(i),"r")
        T(i,:,:,:) = file1->T
end do
```

Learn and take advantage of all three styles of NCL subscripting to subset variables

NCL has a very powerful variable subscripting syntax with features not found in other languages. Learning when to use these features can be critical for writing efficient NCL source.

Consider the operation of transposing a two-dimensional array. One could write the following inefficient source;

```
dims = dimsizes(T)
Ttranspose = new((/dims(1),dims(0)/),float)
do i = 0, dims(1)-1
  do j= 0, dims(0)-1
    Ttranspose(i,j) = T(j,i)
  end do
end do
```

After taking the time to learn learn NCL Named Subscripting, the above can be rewritten in just three lines of source:

```
T!0 = "x"
T!1 = "y"
Ttranspose = T( y | :, x | :)
```

Consider the operation of reversing the order of a dimension:

```
dims = dimsizes(T)
do i = 0,dims(1)/2 -1
        tmp = T(:,i)
        T(:,i) = T(:,(dims(1)-1) - i)
        T(:,(dims(1)-1) - i) = tmp
end do
```

The above operation can be rewritten using only one line of NCL source:

```
T = T(:,::-1)
```

It pays to take the time to learn NCL's unique syntax.

Reduce disk I/O when working with files

Disk I/O is usually much slower than memory accesses, therefore it is recommended that searches or access to individual elements of arrays in files be reduced to improve performance. For example, the first script will run much faster than the second because the search variable is not read from disk on every iteration of the loop. The following file "sao.cdf" contains a two-dimensional character array called "id" in which the three-character station ids are stored. The loops look for the index of a specific id.

```
First loop:

file1 = addfile("sao.cdf","r")
id = file1->id
do i = 0, dimsizes(id(:,0)) - 1
        if( id(i,:) .eq. "DEN")
                break
        end if
end do
print(i)

Second loop:
```

```
file1 = addfile("sao.cdf","r")
do i = 0, dimsizes(file1->id(:,0)) - 1
        if( file1->id(i,:) .eq. "DEN")
                break
        end if
end do
print(i)
```

Build custom functions in Fortran or C when NCL is too slow

NCL has the ability to load dynamic shared libraries which can contain Fortran or C functions and procedures. Using external Fortran and C functions is highly recommended when trying to boost NCL's performance. To configure external functions and procedures, a C source code wrapper must be written. This wrapper de-references parameters from NCL's stack and passes them to the C or Fortran routine. A utility called wrapit77 is available to automatically generate wrappers for Fortran source code. All the user has to do is place the comment "C NCLFORTSTART" before the subroutine or function declaration and "C NCLEND" after the parameter data declarations. Then the user executes the command *wrapit77 < file.f > file_W.c* to generate the C source code wrapper. Next they compile and link into a shared library the Fortran and C files. The shared object is loaded using the *external* command. For more information on this see Extending the NCL function and procedure set.

File I/O issues

Each supported data format may have its own limitations with respect to operations provided by NCL. See the Supported data format information for specific information, conventions, and limitations on each data format in NCL. The following passages are recommendations for using files and working with them.

If you don't know the names of the variables in a file, use the function *getfilevarnames* to achieve this. The syntax for using a string to read a variable is:

```
        names = getfilevarnames(file1)
        var0 = file1->$names(i)$
```

The '$' operator tells NCL to use the string values between the '$'s as the name of the variable to read. Similarly, attributes and coordinate variables can be read in this fashion.

It is often necessary to know the dimension sizes of variables in files. ***Never*** use the *dimsizes* function for this. This will cause ***the entire variable*** to be read in and then discarded. Use the *filevardimsizes* function for this.

Differences between various supported formats with respect to efficiency

It is important to understand efficiency issues with respect to the different formats supported by NCL. There are big differences between the netCDF, GRIB, and CCM formats. Both CCM and GRIB are not by definition self-describing. This means the entire files must be scanned from beginning to end when opened with the *addfile* command. Because of this, there is a noticeable overhead to opening these files when they are large. Knowing this can save time when designing scripts. By contrast, the expense of opening HDF and netCDF files is almost negligible

Writing netCDF files efficiently

As of release 4.1, some new procedures have been added to better support writing files efficiently. NetCDF has some strange performance problems stemming from how the data are arranged on disk. Attributes and ancillary information, if written after data, can cause a tremendous amount of file copies which slow NCL down. By pre-defining the dimensions, attributes, and variables, much time can be saved. Essentially the rule of thumb is to define and write the data in the order it will be laid out in the file. The four procedures are filedimdef, filevardef, filevarattdef, fileattdef. The basic strategy is to first define all the dimensions and then all the variables in the order they'll be written. If at all possible, define one dimension as unlimited. The unlimited dimension initially has no size, meaning any variables added to the file containing the unlimited dimension will also have no size. Variables with no size can be added without incurring a file copy. After adding each variable, add its attributes. By proceeding in this fashion, the file copies are minimized, and when eventual assignment to the variables occurs, file copies again are minimized.

Miscellaneous advice

Defining functions and procedures in scripts

Because functions and procedures can only be defined once, it is important not to put them in scripts that are intended to be run multiple times from the same invocation of NCL. As of release 4.1, functions and procedures can be undefined using the *undef* procedure. When writing functions and procedures that will be loaded, it is probably a good idea to insert an *undef* call immediately before the definition.

```
undef("mymax")
function mymax(a,b)
begin
. . .
```

Use of blocks

When writing scripts, it can be very useful to enclose the script in a block. This forces NCL to scan in the entire source of the script before executing any of the commands nested in the block. This means any syntax errors will be reported before commands are executed. For example, by placing a begin and end around the following, the syntax error after the long loop is detected before the loop is executed.

```
begin
        tmp = new((/1000000/),float)
        do i = 0,999999
                tmp(i) = i
        end do
        asciiwrite("tmp.ascii",tmp(500000:)
end
```