NG4 Reference Manual welcome

The NG4 Reference Manual describes how the NG4 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.

$Revision: 1.2 $ $Date: 1995/07/11 22:24:22 $

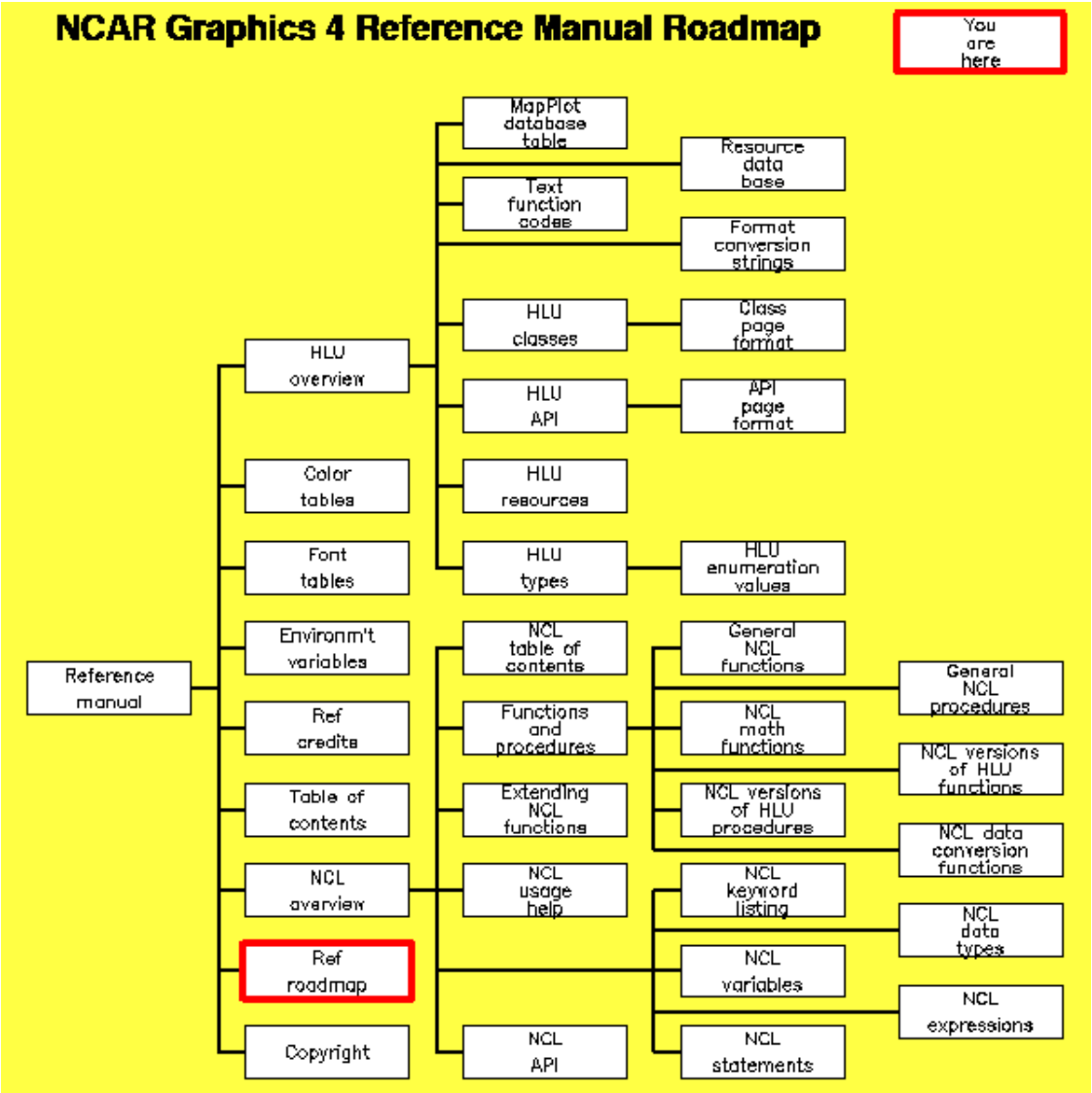
Reference Manual table of contents

Reference Manual introduction
- Table of contents (this file)
- Reference Manual roadmap
- Credits
- Copyright
- Environment variables
- Font tables
- Color tables
- HLU Reference Manual introduction
  - HLU classes
    - Class page format
  - HLU APIs (functions)
    - API page format
  - HLU resources
  - HLU variable types
  - HLU enumeration values
  - Format conversion specification strings
  - Text function codes
  - Resource data base
  - MapPlot database table
- NCL Reference Manual introduction
  - NCL table of contents
  - NCL functions and procedures
    - General NCL functions
    - General NCL procedures
    - NCL math functions
    - NCL versions of HLU functions

NCL versions of HLU procedures
NCL data type conversion functions
NCL language reference
NCL keyword listing
NCL data types
NCL variables
NCL expressions
NCL statements
Extending NCL's functions
NCL usage help
NCL API

$Revision: 1.7 $ $Date: 1995/07/23 19:20:14 $

Reference Manual road map



$Revision: 1.3 $ $Date: 1995/07/11 22:24:03 $

Reference Manual Credits

Acknowledgments

Authors: Ethan Alpert, Dave Brown, Jeff Boote.

Editor: Brian Bevirt

Production: Jacque Marshall

Copyright

Copyright © 1995 University Corporation for Atmospheric Research
All Rights Reserved

Trademarks

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.

$Revision: 1.2 $ $Date: 1995/07/22 18:00:00 $

Copyright

Copyright 1987,1988,1989,1991,1993,1994,1995
University Corporation for Atmospheric Research
All Rights Reserved

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.

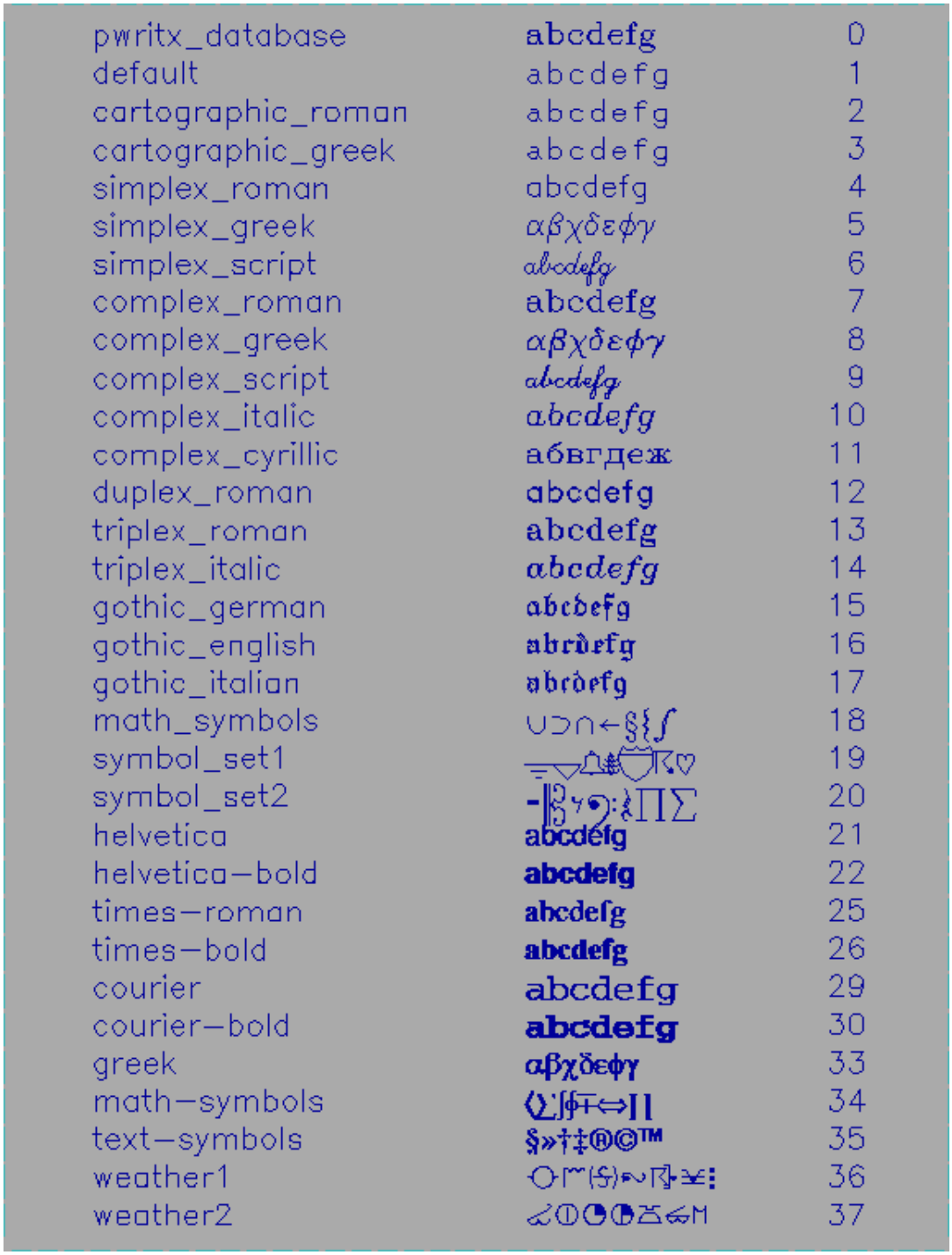
Default: $(NCARG_NCARG)/sysappres

$Revision: 1.4 $ $Date: 1995/07/11 22:24:01 $

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.



$Revision: 1.4 $ $Date: 1995/07/11 22:24:21 $

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.

$Revision: 1.7 $ $Date: 1995/07/20 22:13:00 $

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

$Revision: 1.14 $ $Date: 1995/07/11 22:21:22 $

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.

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

IrregularTransObj
: The **IrregularTransObj** object manages forward and backward 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.

LogLinTransObj
: The **LogLinTransObj** object manages forward and backward 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.

MapTransObj
: The **MapTransObj** object manages forward and backward 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.

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.

TransObj
: The **TransObj** 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.

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.

$Revision: 1.3 $ $Date: 1995/07/20 22:13:13 $

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.

$Revision: 1.4 $ $Date: 1995/07/13 16:56:11 $

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	TYPE		ACCESS
CLASS		DEFAULT	
amOn	NhlTBoolean		RCSG
AmOn		True	

amViewId	NhlTObjId	CG
AmViewId	<dynamic>	
amResizeNotify	NhlTBoolean	RCSG
AmResizeNotify	False	
amTrackData	NhlTBoolean	RCSG
AmTrackData	False	
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 superclasses 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 1987,1988,1989,1991,1993,1994,1995

University Corporation for Atmospheric Research
All Rights Reserved

$Revision: 1.8 $ $Date: 1995/07/24 18:21:39 $

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 NhlPosition 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 parallelling 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 parallelling 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 parallelling 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 parallelling 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: NhlBOTTOM

amParallelPosF

If *amTrackData* is False, *amParallelPosF* specifies the coordinate of the base location of the annotation parallel to the current *amSide* and directed towards 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 towards* 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 NhlJustification, 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: NhlCENTERCENTER

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

$Revision: 1.5 $ $Date: 1995/07/23 18:38:16 $

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

* 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 1987,1988,1989,1991,1993,1994,1995
University Corporation for Atmospheric Research
All Rights Reserved

$Revision: 1.12 $ $Date: 1995/07/23 18:38:17 $

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

$Revision: 1.7 $ $Date: 1995/07/23 18:38:18 $

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 1987,1988,1989,1991,1993,1994,1995
University Corporation for Atmospheric Research
All Rights Reserved

$Revision: 1.15 $ $Date: 1995/07/24 18:21:40 $

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:      LogLinTransObj,IrregularTransObj,PlotManager

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

Class-defined types

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

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:              NhlTcnLabelScalingMode
Definition:
typedef enum _NhlcnLabelScalingMode {
        NhlSCALEFACTOR          = 0,    /* "ScaleFactor"        */
        NhlCONFINETORANGE       = 1,    /* "ConfineToRange"     */
        NhlTRIMZEROS            = 2,    /* "TrimZeros"          */
        NhlMAXSIGDIGITSLEFT     = 3,    /* "MaxSigDigitsLeft"   */
        NhlINTEGERLINELABELS    = 4     /* "IntegerLineLabels"  */
} NhlcnLabelScalingMode;

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	NhlTcnLevelSelectionMode		RCSG
CnLevelSelectionMode		"AutomaticLevels"	
cnLevelCount	NhlTInteger		G
CnLevelCount		<dynamic>	
cnMaxLevelCount	NhlTInteger		RCSG
CnMaxLevelCount		16	
cnLevelSpacingF	NhlTFloat		RCSG
CnLevelSpacingF		5.0	
cnMinLevelValF	NhlTFloat		RCSG
CnMinLevelValF		<dynamic>	

cnMaxLevelValF		NhlTFloat	RCSG
	CnMaxLevelValF	<dynamic>	
cnLevels		NhlTFloatGenArray	RCSG
	CnLevels	<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
	CnLineColor	"Foreground"	
cnLineColors		NhlTColorIndexGenArray	RCSG
	CnLineColors	<dynamic>	
cnMonoLineDashPattern		NhlTBoolean	RCSG
	CnMonoLineDashPattern	True	
cnLineDashPattern		NhlTDashIndex	RCSG
	CnLineDashPattern	"SolidLine"	
cnLineDashPatterns		NhlTDashIndexGenArray	RCSG
	CnLineDashPatterns	<dynamic>	
cnMonoLineThickness		NhlTBoolean	RCSG
	CnMonoLineThickness	True	
cnLineThicknessF		NhlTFloat	RCSG
	CnLineThicknessF	1.0	
cnLineThicknesses		NhlTFloatGenArray	RCSG
	CnLineThicknesses	1.0 for all elements	
cnLineDashSegLenF		NhlTFloat	RCSG
	CnLineDashSegLenF	0.15	
cnRasterModeOn		NhlTBoolean	RCSG
	CnRasterModeOn	False	
cnRasterCellSizeF		NhlTFloat	RCSG
	CnRasterCellSizeF	False	

cnFillOn	NhlTBoolean		RCSG
CnFillOn		False	
cnFillDrawOrder	NhlTDrawOrder		RCSG
CnFillDrawOrder		"Draw"	
cnFillBackgroundColor	NhlTColorIndex		RCSG
CnFillColor		"Transparent"	
cnMonoFillColor	NhlTBoolean		RCSG
CnMonoFillColor		False	
cnFillColor	NhlTColorIndex		RCSG
CnFillColor		"Foreground"	
cnFillColors	NhlTColorIndexGenArray		RCSG
CnFillColors		<dynamic>	
cnMonoFillPattern	NhlTBoolean		RCSG
CnMonoFillPattern		True	
cnFillPattern	NhlTFillIndex		RCSG
CnFillPattern		"SolidFill"	
cnFillPatterns	NhlTFillIndexGenArray		RCSG
CnFillPatterns		<dynamic>	
cnMonoFillScale	NhlTBoolean		RCSG
CnMonoFillScale		True	
cnFillScaleF	NhlTFloat		RCSG
CnFillScales		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	NhlTcnLabelScalingMode		RCSG
CnLabelScalingMode		"ScaleFactor"	
cnLabelScaleValueF	NhlTFloat		RCSG
CnLabelScaleValueF		1.0	
cnLabelScaleFactorF	NhlTFloat		G

	CnLabelScaleFactorF	<dynamic>	
cnMaxDataValueFormat	NhlTString		RCSG
	CnMaxDataValueFormat	NULL	
cnLineLabelsOn	NhlTBoolean		RCSG
	CnLineLabelsOn	True	
cnExplicitLineLabelsOn	NhlTBoolean		RCSG
	CnExplicitLineLabelsOn	False	
cnLineLabelPlacementMode	NhlTcnLineLabelPlacementMode		RCSG
	cnLineLabelPlacementMode	"Computed"	
cnLineLabelStrings	NhlTStringGenArray		RCSG
	CnLineLabelStrings	<dynamic>	
cnMonoLineLabelFontColor	NhlTBoolean		RCSG
	cnMonoLineLabelFontColor	True	
cnLineLabelFontColor	NhlTColorIndex		RCSG
	cnLineLabelFontColor	"Foreground"	
cnLineLabelFontColors	NhlTColorIndexGenArray		RCSG
	cnLineLabelFontColors	<dynamic>	
cnLineLabelFormat	NhlTString		RCSG
	CnLineLabelFormat	"*+g"	
cnLineLabelFontHeightF	NhlTFloat		RCSG
	cnLineLabelFontHeightF	<dynamic>	
cnLineLabelFont	NhlTFont		RCSG
	CnLineLabelFont	"pwritx"	
cnLineLabelFontAspectF	NhlTFloat		RCSG
	CnLineLabelFontAspectF	1.3125	
cnLineLabelFontThicknessF	NhlTFloat		RCSG
	CnLineLabelFontThicknessF	1.0	
cnLineLabelFontQuality	NhlTFontQuality		RCSG
	CnLineLabelFontQuality	"High"	
cnLineLabelConstantSpacingF	NhlTFloat		RCSG
	CnLineLabelConstantSpacingF	0.0	
cnLineLabelAngleF	NhlTFloat		RCSG
	CnLineLabelAngleF	-1.0	
cnLineLabelFuncCode	NhlTCharacter		RCSG
	CnLineLabelFuncCode	':'	
cnLineLabelBackgroundColor	NhlTColorIndex		RCSG
	CnLineLabelBackgroundColor	"Background"	
cnLineLabelPerimOn	NhlTInteger		RCSG
	cnLineLabelPerimOn	True	
cnLineLabelPerimSpaceF	NhlTFloat		RCSG
	CnLineLabelPerimSpaceF	0.33	

cnLineLabelPerimColor	NhlTColorIndex	RCSG
CnLineLabelPerimColor	"Foreground"	
cnLineLabelPerimThicknessF	NhlTFloat	RCSG
CnLineLabelFontThicknessF	1.0	
cnHighLabelsOn	NhlTBoolean	RCSG
CnHighLabelsOn	True	
cnHighLabelString	NhlTString	RCSG
CnHighLabelString	NULL	
cnHighLabelFormat	NhlTString	RCSG
CnHighLabelFormat	"*+g"	
cnHighLabelFontHeightF	NhlTFloat	RCSG
cnHighLabelFontHeightF	<dynamic>	
cnHighLabelFont	NhlTFont	RCSG
CnHighLabelFont	"pwritx"	
cnHighLabelFontColor	NhlTColorIndex	RCSG
CnHighLabelFontColor	"Foreground"	
cnHighLabelFontAspectF	NhlTFloat	RCSG
CnHighLabelFontAspectF	1.3125	
cnHighLabelFontThicknessF	NhlTFloat	RCSG
CnHighLabelFontThicknessF	1.0	
cnHighLabelFontQuality	NhlTFontQuality	RCSG
CnHighLabelFontQuality	"High"	
cnHighLabelConstantSpacingF	NhlTFloat	RCSG
CnHighLabelConstantSpacingF	0.0	
cnHighLabelAngleF	NhlTFloat	RCSG
CnHighLabelAngleF	0.0	
cnHighLabelFuncCode	NhlTCharacter	RCSG
CnHighLabelFuncCode	':'	
cnHighLabelBackgroundColor	NhlTColorIndex	RCSG
CnHighLabelBackgroundColor	"Background"	
cnHighLabelPerimOn	NhlTInteger	RCSG
cnHighLabelPerimOn	True	
cnHighLabelPerimSpaceF	NhlTFloat	RCSG
CnHighLabelPerimSpaceF	0.33	
cnHighLabelPerimColor	NhlTColorIndex	RCSG
CnHighLabelPerimColor	"Foreground"	
cnHighLabelPerimThicknessF	NhlTFloat	RCSG
CnHighLabelFontThicknessF	1.0	
cnLowLabelsOn	NhlTBoolean	RCSG
CnLowLabelsOn	True	

cnLowLabelString	NhlTString	RCSG
CnLowLabelString	NULL	
cnLowLabelFormat	NhlTString	RCSG
CnLowLabelFormat	"*+g"	
cnLowLabelFontHeightF	NhlTFloat	RCSG
cnLowLabelFontHeightF	<dynamic>	
cnLowLabelFont	NhlTFont	RCSG
CnLowLabelFont	"pwritx"	
cnLowLabelFontColor	NhlTColorIndex	RCSG
CnLowLabelFontColor	"Foreground"	
cnLowLabelFontAspectF	NhlTFloat	RCSG
CnLowLabelFontAspectF	1.3125	
cnLowLabelFontThicknessF	NhlTFloat	RCSG
CnLowLabelFontThicknessF	1.0	
cnLowLabelFontQuality	NhlTFontQuality	RCSG
CnLowLabelFontQuality	"High"	
cnLowLabelConstantSpacingF	NhlTFloat	RCSG
CnLowLabelConstantSpacingF	0.0	
cnLowLabelAngleF	NhlTFloat	RCSG
CnLowLabelAngleF	0.0	
cnLowLabelFuncCode	NhlTCharacter	RCSG
CnLowLabelFuncCode	':'	
cnLowLabelBackgroundColor	NhlTColorIndex	RCSG
CnLowLabelBackgroundColor	"Background"	
cnLowLabelPerimOn	NhlTInteger	RCSG
cnLowLabelPerimOn	True	
cnLowLabelPerimSpaceF	NhlTFloat	RCSG
CnLowLabelPerimSpaceF	0.33	
cnLowLabelPerimColor	NhlTColorIndex	RCSG
CnLowLabelPerimColor	"Foreground"	
cnLowLabelPerimThicknessF	NhlTFloat	RCSG
CnLowLabelFontThicknessF	1.0	
cnInfoLabelOn	NhlTBoolean	RCSG
CnInfoLabelOn	True	
cnInfoLabelString	NhlTString	RCSG
CnInfoLabelString	NULL	
cnInfoLabelFormat	NhlTString	RCSG
CnInfoLabelFormat	"*+g"	
cnInfoLabelFontHeightF	NhlTFloat	RCSG
cnInfoLabelFontHeightF	<dynamic>	
cnInfoLabelTextDirection	NhlTTextDirection	RCSG

```
      CnInfoLabelTextDirection        "Across"
-------------------------------------------------------------
 cnInfoLabelFont                NhlTFont                RCSG
      CnInfoLabelFont                 "pwritx"
-------------------------------------------------------------
 cnInfoLabelFontColor           NhlTColorIndex          RCSG
      CnInfoLabelFontColor            "Foreground"
-------------------------------------------------------------
 cnInfoLabelFontAspectF         NhlTFloat               RCSG
      CnInfoLabelFontAspectF          1.3125
-------------------------------------------------------------
 cnInfoLabelFontThicknessF      NhlTFloat               RCSG
      CnInfoLabelFontThicknessF       1.0
-------------------------------------------------------------
 cnInfoLabelFontQuality         NhlTFontQuality         RCSG
      CnInfoLabelFontQuality          "High"
-------------------------------------------------------------
 cnInfoLabelConstantSpacingF    NhlTFloat               RCSG
      CnInfoLabelConstantSpacingF     0.0
-------------------------------------------------------------
 cnInfoLabelAngleF              NhlTFloat               RCSG
      CnInfoLabelAngleF               0.0
-------------------------------------------------------------
 cnInfoLabelFuncCode            NhlTCharacter           RCSG
      CnInfoLabelFuncCode             ':'
-------------------------------------------------------------
 cnInfoLabelBackgroundColor     NhlTColorIndex          RCSG
      CnInfoLabelBackgroundColor      "Background"
-------------------------------------------------------------
 cnInfoLabelPerimOn             NhlTInteger             RCSG
      cnInfoLabelPerimOn              True
-------------------------------------------------------------
 cnInfoLabelPerimSpaceF         NhlTFloat               RCSG
      CnInfoLabelPerimSpaceF          0.33
-------------------------------------------------------------
 cnInfoLabelPerimColor          NhlTColorIndex          RCSG
      CnInfoLabelPerimColor           "Foreground"
-------------------------------------------------------------
 cnInfoLabelPerimThicknessF     NhlTFloat               RCSG
      CnInfoLabelFontThicknessF       1.0
-------------------------------------------------------------
 cnInfoLabelZone                NhlTInteger             RCSG
      CnInfoLabelZone                 5
-------------------------------------------------------------
 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
      CnNoDataLabelOn                 True
-------------------------------------------------------------
 cnNoDataLabelString            NhlTString              RCSG
      CnNoDataLabelString             NULL
```

cnConstFLabelOn	NhlTBoolean	RCSG
CnConstFLabelOn	True	
cnConstFLabelString	NhlTString	RCSG
CnConstFLabelString	NULL	
cnConstFLabelFormat	NhlTString	RCSG
CnConstFLabelFormat	"*+g"	
cnConstFLabelFontHeightF	NhlTFloat	RCSG
cnConstFLabelFontHeightF	<dynamic>	
cnConstFLabelTextDirection	NhlTTextDirection	RCSG
CnConstFLabelTextDirection	"Across"	
cnConstFLabelFont	NhlTFont	RCSG
CnConstFLabelFont	"pwritx"	
cnConstFLabelFontColor	NhlTColorIndex	RCSG
CnConstFLabelFontColor	"Foreground"	
cnConstFLabelFontAspectF	NhlTFloat	RCSG
CnConstFLabelFontAspectF	1.3125	
cnConstFLabelFontThicknessF	NhlTFloat	RCSG
CnConstFLabelFontThicknessF	1.0	
cnConstFLabelFontQuality	NhlTFontQuality	RCSG
CnConstFLabelFontQuality	"High"	
cnConstFLabelConstantSpacingF	NhlTFloat	RCSG
CnConstFLabelConstantSpacingF	0.0	
cnConstFLabelAngleF	NhlTFloat	RCSG
CnConstFLabelAngleF	0.0	
cnConstFLabelFuncCode	NhlTCharacter	RCSG
CnConstFLabelFuncCode	':'	
cnConstFLabelBackgroundColor	NhlTColorIndex	RCSG
CnConstFLabelBackgroundColor	"Background"	
cnConstFLabelPerimOn	NhlTInteger	RCSG
cnConstFLabelPerimOn	True	
cnConstFLabelPerimSpaceF	NhlTFloat	RCSG
CnConstFLabelPerimSpaceF	0.33	
cnConstFLabelPerimColor	NhlTColorIndex	RCSG
CnConstFLabelPerimColor	"Foreground"	
cnConstFLabelPerimThicknessF	NhlTFloat	RCSG
CnConstFLabelFontThicknessF	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	NhlTInteger	RCSG
cnMissingValPerimOn	False	
cnMissingValPerimThicknessF	NhlTFloat	RCSG
CnMissingValPerimThicknessF	1.0	
cnMissingValPerimDashPattern	NhlTDashIndex	RCSG
CnMissingValPerimDashPattern	"SolidLine"	
cnMissingValPerimColor	NhlTColorIndex	RCSG
CnMissingValPerimColor	"Foreground"	
cnMissingValFillColor	NhlTColorIndex	RCSG
CnMissingValFillColor	"Background"	
cnMissingValFillPattern	NhlTInteger	RCSG
CnMissingValFillPattern	"HollowFill"	
cnMissingValFillScaleF	NhlTFloat	RCSG
CnMissingValFillScaleF	1.0	
cnGridBoundPerimOn	NhlTInteger	RCSG
cnGridBoundPerimOn	False	
cnGridBoundPerimThicknessF	NhlTFloat	RCSG
CnGridBoundPerimThicknessF	1.0	
cnGridBoundPerimDashPattern	NhlTDashIndex	RCSG
CnGridBoundPerimDashPattern	"SolidLine"	
cnGridBoundPerimColor	NhlTColorIndex	RCSG
CnGridBoundPerimColor	"Foreground"	
cnOutOfRangePerimOn	NhlTInteger	RCSG
cnOutOfRangePerimOn	False	
cnOutOfRangePerimThicknessF	NhlTFloat	RCSG
CnOutOfRangePerimThicknessF	1.0	
cnOutOfRangePerimDashPattern	NhlTDashIndex	RCSG
CnOutOfRangePerimDashPattern	"SolidLine"	
cnOutOfRangePerimColor	NhlTColorIndex	RCSG
CnOutOfRangePerimColor	"Foreground"	
cnSmoothingOn	NhlTBoolean	RCSG
CnSmoothingOn	False	
cnSmoothingTensionF	NhlTFloat	RCSG
CnSmoothingTensionF	2.5	
cnSmoothingDistanceF	NhlTFloat	RCSG

	CnSmoothingDistanceF	0.01	
cnMaxPointDistanceF	NhlTFloat		RCSG
	CnMaxPointDistanceF	0.05	
cnExplicitLabelBarLabelsOn	NhlTBoolean		RCSG
	CnExplicitLabelBarLabelsOn	False	
cnLabelBarEndLabelsOn	NhlTBoolean		RCSG
	CnLabelBarEndLabelsOn	False	
cnExplicitLegendLabelsOn	NhlTBoolean		RCSG
	CnExplicitLegendLabelsOn	False	
cnLegendLevelFlags	NhlTcnLevelUseModeGenArray		RCSG
	CnLegendLevelFlags	NULL	

Data specific resources

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

Composite resources

LogLinTransObj resources

The **ContourPlot** class uses the **LogLinTransObj** to handle its transformations as long as neither of the **ScalarField** array resources, *sfXArray* and *sfYArray* are set.

You can access all **LogLinTransObj** 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. However, note that **ContourPlot** intercepts the **LogLinTransObj** 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.
- trXLog
  The **ContourPlot** object issues a warning if *trXLog* is set True and the implicitly or explicitly set value of *trXMinF* is less than or equal to 0.0. In this case, it sets *trXLog* False.
- 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.
- trYLog
  The **ContourPlot** object issues a warning if *trYLog* is set True and the implicitly or explicitly set value of *trYMinF* is less than or equal to 0.0. In this case, it sets *trYLog* False.

IrregularTransObj resources

ContourPlot automatically uses the **IrregularTransObj** instead of the **LogLinTransObj** 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.

The **ContourPlot** class currently blocks access to the following **IrregularTransObj** resources:

- trXAxisType
- trXCoordPoints
- trXInterPoints
- trXTensionF
- trXSamples
- trYAxisType
- trYCoordPoints
- trYInterPoints
- trYTensionF
- trYSamples

ContourPlot allows access to the following **IrregularTransObj** resources but intercepts them 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. If the **ScalarField** resource *sfXArray* is set, implying that the X Axis coordinate system is irregular, **ContourPlot** issues a warning if you attempt to set *trXMinF* to a value outside the range of values contained in *sfXArray* and *trXMinF* reverts to its default value.
- 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. If the **ScalarField** resource

sfXArray is set, implying that the X Axis coordinate system is irregular, **ContourPlot** issues a warning if you attempt to set *trXMaxF* to a value outside the range of values contained in *sfXArray* and *trXMaxF* reverts to its default value.

- 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. If the **ScalarField** resource *sfYArray* is set, implying that the Y Axis coordinate system is irregular, **ContourPlot** issues a warning if you attempt to set *trYMinF* to a value outside the range of values contained in *sfYArray* and *trXMaxF* reverts to its default value.
- 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. If the **ScalarField** resource *sfYArray* is set, implying that the Y Axis coordinate system is irregular, **ContourPlot** issues a warning if you attempt to set *trYMaxF* to a value outside the range of values contained in *sfYArray* and *trYMaxF* reverts to its default value.
- trYReverse
  By default *trYReverse* is set based on the direction of the Y Axis implied by the contents of the **ScalarField** object.

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.
- 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 one **Legend** resource, as follows:

- lgLabelStrings
  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 labelled. You can cause a number of annotational items to appear, including an informational label, tickmarks, 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 contour plot. You need 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 before calling the **Draw** method with a **ContourPlot** object 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 is 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 is irregularly spaced, the **ContourPlot** receives information defining this spacing.

Coordinate transformations

The **ContourPlot** object intrinsically supports linear, logarithmic, and irregular rectangular gridded data coordinate spaces. The **ContourPlot** object instantiates child objects to manage transformations between the data coordinate system and NDC space. The **LogLinTransObj** manages linear and logarithmic transformations, and the **IrregularTransObj** manages the tranformation 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. If either of the coordinate axes is irregularly spaced, you may limit the coordinate extent to a subspace of the data coordinate extent of the **ScalarField** object data, but you are not allowed to define a coordinate range that extend outside the range of the data. If no coordinate axes are irregular, you may define coordinate extents that arbitrarily intersect or encompass the data extent.

In addition to the intrinsic transformation support, by overlaying the **ContourPlot** object on a base plot, you may map contours of arbitrary **ScalarField** data into a variety of coordinate spaces. You can overlay

the **ContourPlot** object on a **MapPlot** object to map the data into any of 10 different map projections. You can overlay contours of irregularly gridded data on the **LogLinPlot** object to have the data appear in a linear or logarithmic coordinate space. You can also overlay any contoured data on an **IrregularPlot** object, either to project regularly gridded data into an irregular coordinate space, or perhaps to overcome the limitations imposed on the data extent of the **ContourPlot** object's intrinsic irregular transformation. Use the **NhlAddOverlay** function to accomplish the overlay operation.

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* order. There are three drawing phases: the *predraw* phase, the *draw* phase, and the *postdraw* phase. By default, **ContourPlot** object 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, tickmarks 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 tickmarks 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 other 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 data set you are contouring, and therefore can be used to enforce consistency in the contouring of a series of related data sets. `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, line labels automatically appear at contour levels spaced apart by the value of the resource *cnLineLabelInterval*, set by default to the value 2.

Controlling line attributes

You can control whether contour lines appear at all by setting the boolean resource *cnLinesOn*. By default it is set to 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; the array resources by setting them False. There is also a scalar resource, *cnLineDashSegLenF* that sets the segment length for all dash patterns used.

Controlling fill attributes

You turn on fill for contour regions by setting the boolean resource *cnFillOn* True. Its default value is False. **ContourPlot** supports two methods for filling contour regions: raster and non-raster. You turn on the raster method by setting the boolean resource *cnRasterModeOn* True.

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; the array resources by setting them False.

Raster fill is based on an array of square cells superimposed on the area occupied by the data grid. Each cell is colored according to the value of the data at its centerpoint. You can control the size of the cell used to create this grid using the resource *cnRasterCellSizeF* and the colors used for each contour level using the resource *cnFillColors*. The remaining fill attribute 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 attempts to sort out the label types and classify the label attributes.

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

In order to simplify the management of label resources, there are several resources that allow you tie most 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. *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 placed by the lower level utilities, some attributes of high and low labels are tied together and cannot be set individually. 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 type as they apply to all label types. The '...' in the resource name is understood to stand for the identifying name of each label type, as follows:

- *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, 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 data set 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.

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 of 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 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 cannot be set differently from 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 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 which side of the plot viewport to locate the label.

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.

The position of line labels can be adjusted using the resource *cnLineLabelPlacementMode*. Also you can control in a limited way the location of high and low labels with the resource *cnHighLowLabelOverlapMode*.

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, where '...' in the resource name is understood to stand for the identifying name of each boundary type, *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 the **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

6: 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.

8: You cannot yet access the parameters, RC1, RC2, and RC3 that control the degree of randomnesss for the `Randomized` method of *cnLineLabelPlacementMode*.

9: You cannot control the parameters that control the algorithm for the `Computed` method of *cnLineLabelPlacementMode*. Should the name of this option be changed? Under certain conditions when part of the data space is outside the viewport, the Computed methed seems to fail, and the plot appears without any line labels at all. This problem requires further investigation.

13: Large values of *cnSmoothingTensionF* above 10.0 or so cause the **ContourPlot** object often 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.

15: There are problems with contour fill dropping out when projected through some of the map projections.

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 1987,1988,1989,1991,1993,1994,1995
University Corporation for Atmospheric Research
All Rights Reserved

$Revision: 1.12 $ $Date: 1995/07/24 23:22:06 $

ContourPlot class resource descriptions

cnScalarFieldData

Specifies the id of a **ScalarField** data object. There is no default for this resource; it the only resource that must be set in order for the **ContourPlot** object to draw a plot. You *may* create a **ContourPlot** object without setting the *cnScalarFieldData*, and auxiliary annotations such as tickmarks 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 provides a number of resources for controlling the ingestion of the raw data.

Default: <None>

cnLevelSelectionMode

This enumerated resource of type NhlcnLevelSelectionMode 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** set 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 cause the absolute maximum number of levels, currently 256, 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 256, **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 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* 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 256), **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 256), 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 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 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 NhlcnLevelUseMode sets the rendering of all contour levels to one of four possible values:

`NoLine`
: No lines or label 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 to True to enable all possible settings of *cnLevelFlag*.

Default:

cnLevelFlags

When *cnMonoLevelFlag* is False, *cnLevelFlags* is an array of type NhlcnLevelUseModeGenArray 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 line or label appears 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 labelled level to the next labelled

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 labelled.

Default: 2

cnLinesOn

If this boolean resource is set to 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 NhlDrawOrder 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 pre-draw phase, but will underlie elements drawn during the post-draw phase.

`PostDraw`
: Draw the lines after the standard draw; the lines will overlay any elements drawn during the pre-draw 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 inidividually 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 inidividually 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 scheme 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 dash pattern of each line individually.

Default: True

cnLineThicknessF

When *cnMonoLineThickness* is True, this resource sets a uniform line thickness for all contour lines.

Default:

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 before the dash pattern repeats itself. This resource automatically scales with changes in the size of the viewport of the **ContourPlot** object. strong>ContourPlot sets the default value of *cnLineDashSegLenF* dynamically based upon the ratio of the actual plot viewport width to a reference viewport width of 0.5 and a reference dash segment length of 0.075.

Default: 0.075 (for a viewport width of 0.5)

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, the areas between contour levels are colored using a raster filling technique. The area encompassing the data is divided into an array of square cells. The data value of the center point of each cell determines its color. This is in contrast to the ordinary method of coloring the irregular shapes 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*

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).

Processing time, memory requirements, and output quality all vary inversely with the size of the cells in the cell array.

Default: False

cnRasterCellSizeF

This resource defines the length of each side of the squares that make up the cell array when the raster mode of contour filling is enabled by setting *cnRasterModeOn* True. Its units are NDC. Processing time, memory requirements, and output quality all vary inversely with the value of this resource. Its value scales proportionally to changes in the viewport width, unless you explicitly set *cnRasterCellSizeF* during the same call. This has the effect of maintaining the rendering at the same level of image quality regardless of its size within the viewport.

Default: <dynamic> -- 0.001 for a viewport width of 0.5

cnFillDrawOrder

This resource of type NhlDrawOrder 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 pre-draw phase, but will underlie elements drawn during the post-draw phase.

`PostDraw`

Fill contour areas after the standard draw; the area fill overlay any elements drawn during the pre-draw 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 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 similiar 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* plus, 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:

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 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* plus one, 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, the 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* plus one, 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 NhlDrawOrder 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 areas 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 pre-draw phase as well as fill and contour lines drawn during the standard draw phase, but will underlie elements drawn during the post-draw phase.

`PostDraw`
: Draw contour labels after the standard draw; the labels will overlay any elements drawn during the pre-draw and draw phases as well as lines and fill drawn during the post-draw 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 NhlcnHighLowLabelOverlapMode. It controls how high and low labels are allowed to overlap other elements of the **ContourPlot** plot and has 6 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.

V4.0 Status Note 6

Default: `IgnoreOverlap`

cnLabelScalingMode

This resource of type NhlcnLabelScalingMode 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 multiplies 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 data value with the maximum absolute value will be less than *cnLabelScaleValueF*, but greater or equal to *cnLabelScaleValueF* divided by 10.0. As an example, setting *cnLabelScaleValueF* to 100.0 would result in the number used to represent the maximum absolute data value falling 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*.

`IntegerLineLabels`

The **ContourPlot** object selects a scale factor such that the numbers representing labelled contour levels (as determined by examination of the values of the *cnLabelFlags* array) are reduced to integers.

In each case, the scale factor selected is placed into the read-only resource, *cnLabelScaleFactorF*.

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 multiplied 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 applied to the scalar field data in order to obtain the numbers used in **ContourPlot** object labels. 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*.

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 sets the dynamic attribute on 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: NULL

cnLineLabelsOn

If this boolean resource is set to 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 NhlcnLineLabelPlacementMode determines the algorithm **ContourPlot** uses to place line labels. There are three choices:

`Constant`

Labels are equally-spaced along the lines.

`Randomized`

ContourPlot places labels using a randomizing algorithm to vary the distance between labels.

V4.0 Status Note 8

`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.

V4.0 Status Note 9

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 by 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: NULL

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: "*+g"

cnLineLabelFontHeightF

This resource controls the height in NDC 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>

cnLineLabelFont

This resource of type NhlFont specifies the font used to render line labels.

Default: 0

cnLineLabelFontAspectF

This resource determines the shape of the line label characters. Values increasing from 1.0 results in thinner characters. Values decreasing from one makes 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 *lbLabelFont* specifies a filled font (font indexes 21 through 30 and 33 through 37).

Default: 1.0

cnLineLabelFontQuality

This resource of type NhlFontQuality 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 NhlCharacter 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 to False, **ContourPlot** will not create labels for highs in the

ScalarField data.

Default: True

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 low at the point where the label is to appear. *cnHighLabelFormat* determines the format of the number and *cnLabelScaleFactorF* determines the scale factor applied to the actual data value. Note that the value of *cnHighLabelString* does *not* take its value from *cnLineLabelString* when *cnHighUseLowLabelRes* is True.

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: "*+g"

cnHighLabelFontHeightF

This resource controls the height in NDC 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>

cnHighLabelFont

This resource of type NhlFont specifies the font used to render high labels.

Default: 0

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 results in thinner characters. Values decreasing from one makes 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 *lbLabelFont* specifies a filled font (font indexes 21 through 30 and 33 through 37).

Default: 1.0

cnHighLabelFontQuality

This resource of type NhlFontQuality 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. Note that the value of *cnHighLabelAngleF* does *not* take its value from *cnLineLabelAngleF* when *cnHighUseLowLabelRes* is True.

Default: 0.0

cnHighLabelFuncCode

This resource of type NhlCharacter 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 to False, **ContourPlot** will not create labels for lows in the **ScalarField** data.

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 low at the point where the label is to appear. *cnLowLabelFormat* determines the format of the number and *cnLabelScaleFactorF* determines the scale factor applied to the actual data value. Note that the value of *cnLowLabelString* does *not* take its value from *cnHighLabelString* when *cnLowUseHighLabelRes* is True.

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: "*+g"

cnLowLabelFontHeightF

This resource controls the height in NDC 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>

cnLowLabelFont

This resource of type NhlFont specifies the font used to render low labels.

Default: 0

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 results in thinner characters. Values decreasing from one makes 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 *lbLabelFont* specifies a filled font (font indexes 21 through 30 and 33 through 37).

Default: 1.0

cnLowLabelFontQuality

This resource of type NhlFontQuality 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 NhlCharacter 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 to 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 the each number and except for the number generated from SFU, all values are scaled by 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 the *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: "*+g"

cnInfoLabelTextDirection

This resource of type NhlTextDirection specifies the direction of the text in the informational label. There are three choices of which two are actually the same:

`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 *cnInfoLabelAngleF* is applied to the informational label.

Default: `Across`

cnInfoLabelFontHeightF
This resource controls the height in NDC 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>

cnInfoLabelFont
This resource of type NhlFont specifies the font used to render the informational label.

Default: 0

cnInfoLabelFontColor
This resource specifies the hlu color index used to render informational label text.

Default: True

cnInfoLabelFontAspectF
This resource determines the shape of the informational label characters. Values increasing from 1.0 results in thinner characters. Values decreasing from one makes 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 *lbLabelFont* specifies a filled font (font indexes 21 through 30 and 33 through 37).

Default: 1.0

cnInfoLabelFontQuality
This resource of type NhlFontQuality 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 NhlCharacter 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 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: 5

cnInfoLabelSide

This resource of type NhlPosition 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, which behave as follows, unless *cnInfoLabelZone* is set to one of the special zones (0 or 1):

`Top`
: The **PlotManager** locates the informational label annotation relative to a line parallelling 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`
: The **PlotManager** locates the informational label annotation relative to a line parallelling 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`
: The **PlotManager** locates the informational label annotation relative to a line parallelling 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`
: The **PlotManager** locates the informational label annotation relative to a line parallelling 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 towards 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 NhlJustification, 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 is determined to have a single constant value within the limits of the available precision or when no **ScalarField** data supplied.

Default: True

cnConstFLabelString

Specifies the string to use when drawing an 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 by 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: "*+g"

cnConstFLabelTextDirection

This resource of type NhlTextDirection specifies the direction of the text in the constant field label. There are three choices of which two are actually the same:

`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 *cnInfoLabelAngleF*.

Default: `Across`

cnConstFLabelFontHeightF

This resource controls the height in NDC 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 info label text height scales with changes to the viewport width, unless you explicitly set *cnConstFLabelFontHeightF* during the same call.

Default: <dynamic>

cnConstFLabelFont

This resource of type NhlFont specifies the font used to render the constant field label.

Default: 0

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 results in thinner characters. Values decreasing from one makes 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 through 30 and 33 through 37).

Default: 1.0

cnConstFLabelFontQuality

This resource of type NhlFontQuality 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 NhlCharacter 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 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.

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 NhlPosition 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`

The **PlotManager** locates the constant field label annotation relative to a line parallelling 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`

The **PlotManager** locates the constant field label annotation relative to a line parallelling 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`

The **PlotManager** locates the constant field label annotation relative to a line parallelling 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`

The **PlotManager** locates the constant field label annotation relative to a line parallelling 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 towards 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 NhlJustification, 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** plot.

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 will 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 is 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.

V4.0 Status Note 13

Default: -2.5

cnSmoothingDistanceF

cnSmoothingDistanceF sets the distance between points used to draw the 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 both along the width and height of the viewport allowed before checking for a discontinous 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 is 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 wish 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 data set, and at the other end will be a string representation of the maximum value in the data set. 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 wish to perform.

Default: False

cnLegendLevelFlags

This array resource of type NhlcnLevelUseModeGenArray 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

$Revision: 1.11 $ $Date: 1995/07/24 18:21:42 $

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	NhlTInteger	RCSG
CtYElementSize	<dynamic>	
ctYMissingV	NhlTVariable	RCSG
diMissingValue	NULL	
ctYMaxV	NhlTVariable	RCSG
CtYMaxV	<dynamic>	
ctYMinV	NhlTVariable	RCSG
CtYMinV	<dynamic>	

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 1987,1988,1989,1991,1993,1994,1995
University Corporation for Atmospheric Research
All Rights Reserved

$Revision: 1.13 $ $Date: 1995/07/23 18:38:22 $

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.

$Revision: 1.7 $ $Date: 1995/07/23 18:38:23 $

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	TYPE	ACCESS	
CLASS	DEFAULT		
caCopyArrays	NhlTBoolean	RCSG	

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

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 1987,1988,1989,1991,1993,1994,1995
University Corporation for Atmospheric Research
All Rights Reserved

$Revision: 1.13 $ $Date: 1995/07/23 18:38:24 $

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.

$Revision: 1.7 $ $Date: 1995/07/23 18:38:25 $

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>
```

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 1987,1988,1989,1991,1993,1994,1995
University Corporation for Atmospheric Research
All Rights Reserved

$Revision: 1.11 $ $Date: 1995/07/24 18:21:43 $

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 to 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

$Revision: 1.7 $ $Date: 1995/07/23 18:38:27 $

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 1987,1988,1989,1991,1993,1994,1995
University Corporation for Atmospheric Research
All Rights Reserved

$Revision: 1.8 $ $Date: 1995/07/23 18:38:28 $

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 1987,1988,1989,1991,1993,1994,1995
University Corporation for Atmospheric Research
All Rights Reserved

$Revision: 1.6 $ $Date: 1995/07/23 18:38:29 $

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,
        NhlWARNING      = -3,
        NhlINFO         = -2,
        NhlNOERROR      = -1
} 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 1987,1988,1989,1991,1993,1994,1995
University Corporation for Atmospheric Research
All Rights Reserved

$Revision: 1.12 $ $Date: 1995/07/23 18:38:31 $

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.

$Revision: 1.5 $ $Date: 1995/07/23 18:38:31 $

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:      IrregularTransObj,PlotManager
```

Resources

Local resources

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

Composite resources

IrregularTransObj resources

You can access all **IrregularTransObj** resources. These resources define a mapping from a regular grid to the irregular coordinate system for each coordinate axis.

PlotManager resources

Assuming *tfPlotManagerOn* is True when a **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

The **IrregularPlot** object has no **Draw** method of its own. Its primary function is to act as a base plot over which other plot objects may be overlaid transforming their coordinates into an irregular coordinate space. Although it has no **Draw** method, the **IrregularPlot** object *does* allow you to draw immediate mode polylines, using the **Transform** class **DataPolyline** functions.

You can transform a plot object into an irregularly-gridded coordinate space by overlaying it on a **IrregularPlot** acting as a base plot. This irregularly-gridded space may encompass the data space of any number of regularly or irregularly gridded plot objects with partially or completely disjoint data coordinate extents. With such a space defined for an **IrregularPlot** base plot, you can combine into a single plot many disjoint plot objects.

You define the irregular mapping, as well as the coordinate axis extents and directions, using the resources of the **IrregularTransObj** composite class member. Note that the **IrregularTransObj** allows you to define an irregular transformation for one axis and a linear or logarithmic transformation for the other axis.

Assuming *tfPlotManagerOn* is True when you create a **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.

The immediate mode lines draw into the **Workstation** object as soon as you call the **DataPolyline** function. They are *not* part of the object because they do not appear when you execute the **Draw** method for the **IrregularPlot** object. At the present time there is no way to save these lines into a segment.

Support functions

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

Status

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

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

See also

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

Copyright

Copyright 1987,1988,1989,1991,1993,1994,1995
University Corporation for Atmospheric Research
All Rights Reserved

$Revision: 1.12 $ $Date: 1995/07/24 18:21:43 $

IrregularPlot class resource descriptions

The IrregularPlot class has no local resources.

$Revision: 1.4 $ $Date: 1995/07/23 18:38:33 $

IrregularTransObj class

The **IrregularTransObj** class manages forward and backward transformations in an irregular rectangular coordinate space.

Synopsis

```
Header file:            ncarg/hlu/IrregularTransObj.h
Class name:             irregularTransObjClass
Class pointer:          <Not referenceable>
Fortran class function: <Not referenceable>
Superclass:             TransObj
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

IrregularTransObj Resource Set			
NAME	TYPE	ACCESS	
CLASS	DEFAULT		
trXAxisType	NhlTAxisType	RCSG	
TrXAxisType	"LinearAxis"		
trXCoordPoints	NhlTFloatGenArray	RCSG	
TrXCoordPoints	NULL		
trXInterPoints	NhlTFloatGenArray	RCSG	
TrXInterPoints	NULL		
trXMaxF	NhlTFloat	RCSG	
TrXMaxF	0.0		
trXMinF	NhlTFloat	RCSG	
TrXMinF	0.0		
trXReverse	NhlTBoolean	RCSG	
TrXReverse	False		
trXTensionF	NhlTFloat	RCSG	
TrXTensionF	2.0		
trXSamples	NhlTInteger	RCSG	
TrXSamples	9		

trYAxisType TrYAxisType	NhlTAxisType "LinearAxis"	RCSG
trYCoordPoints TrYCoordPoints	NhlTFloatGenArray NULL	RCSG
trYInterPoints TrYInterPoints	NhlTFloatGenArray NULL	RCSG
trYMaxF TrYMaxF	NhlTFloat 0.0	RCSG
trYMinF TrYMinF	NhlTFloat 0.0	RCSG
trYReverse TrYReverse	NhlTBoolean False	RCSG
trYTensionF TrYTensionF	NhlTFloat 2.0	RCSG
trYSamples TrYSamples	NhlTInteger 9	RCSG

Composite resources

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

Superclass resources

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

- **TransObj** resources

Description

The **IrregularTransObj** provides transformations into a rectangular irregularly gridded coordinate space for plot objects belonging to the **Transform** class. It contains resources for specifying the mapping from the linear space into the irregularly gridded space. Within the defined coordinate bounds you can specify the coordinate extent along each of its axes. You can specify the direction of increasing coordinate values. You can also choose to create a transformation that is irregular along one axis and logarithmic or linear along the other. (Actually, there is nothing to prevent you from making both axes linear or logarithmic, but you would pay a performance penalty compared with simply using the **LogLinTransObj** in this case.)

Currently some plot objects, such as the **XyPlot** object, have their own resources for setting the **IrregularTransObj** resources. For these objects the **IrregularTransObj** resources are considered private. The **ContourPlot** object allows access only to the boundary setting resources. The **IrregularPlot** object currently allows you to access all resources of the **IrregularTransObj**.

The **ContourPlot** object requires that the extent of an irregular coordinate system be coincident with or a subset of the data extent as defined by the array resources *sfXArray* and *sfYArray* belonging to the **ScalarField** object. However, by overlaying a **ContourPlot** object on an **IrregularPlot** object, you can create a **ContourPlot** plot that is arbitrarily positioned in relation to the plot as a whole.

Support functions

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

4:Large values of *trXTensionF* or *trYTensionF* (> ~10.0) may cause instability in the irregular transformation.

See also

- IrregularPlot object
- PlotManager object
- ContourPlot object
- XyPlot object
- TransObj object
- Obj object

Copyright

Copyright 1987,1988,1989,1991,1993,1994,1995
University Corporation for Atmospheric Research
All Rights Reserved

$Revision: 1.12 $ $Date: 1995/07/24 18:21:44 $

IrregularTransObj 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 establish for the X Axis with boundaries defined by the values of *trXMinF* and *trXMaxF*.

This resource may be intercepted or disabled by:

- **ContourPlot**
- **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 coodinate 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 **IrregularTransObj** 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`.

V4.0 Status Note 2

This resource may be intercepted or disabled by:

- **ContourPlot**
- **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, **IrregularTransObj** 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**
- **XyPlot**

Default: NULL

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 *trXAxisType* is `LogAxis` *trXMinF* must be greater than 0.0. If *trXAxisType* is `IrregularAxis`, *trXMinF* is constrained to lie within the bounds of the irregular coordinate system established by *trXCoordPoints*. In other words, you can establish a subset of the irregular coordinate system using *trXMinF* and *trXMaxF*, but you can expand the range of the coordinate system only by modifying the *trXCoordPoints* array. If you set *trXCoordPoints* and do not set *trXMinF* explicitly in the same call, *trXMinF* is set to the minimum coordinate value contained in *trXCoordPoints*.

This resource may be intercepted or disabled by:

- **ContourPlot**
- **XyPlot**

Default: 0.0

trXMaxF

Specifies a maximumum 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 *trXAxisType* is `IrregularAxis`, *trXMinF* is constrained to lie within the bounds of the irregular coordinate system established by *trXCoordPoints*. In other words, you can establish a subset of the irregular coordinate system using *trXMaxF* and *trXMinF*, but you can expand the range of the coordinate system only by modifying the *trXCoordPoints* array. If you set *trXCoordPoints* and do not set *trXMaxF* explicitly in the same call, *trXMaxF* is set to the maximum coordinate value contained in *trXCoordPoints*.

This resource may be intercepted or disabled by:

- **ContourPlot**
- **XyPlot**

Default: 0.0

trXReverse

If the boolean resource *trXReverse* is set True, the direction of the X Axis coordinate values is reversed: if *trAxisType* is `LogAxis` or `LinearAxis` or if *trAxisType* is `IrregularAxis` and *trXCoordPoints* is monotonically increasing, the coordinate values will increase from right to left rather than from left to right.

This resource may be intercepted or disabled by:

- **ContourPlot**

Default: False

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.

This resource may be intercepted or disabled by:

- **ContourPlot**
- **XyPlot**

V4.0 Status Note 4

Default: 2.0

trXSamples

Specifies the number of sample points over the X-Axis coordinate system **IrregularTransObj** 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**
- **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 establish for the Y Axis with boundaries defined by the values of *trYMinF* and *trYMaxF*.

This resource may be intercepted or disabled by:

- **ContourPlot**
- **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 coodinate 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 **IrregularTransObj** 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**
- **XyPlot**

V4.0 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, **IrregularTransObj** 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**
- **XyPlot**

Default: NULL

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 *trYAxisType* is `LogAxis` *trYMinF* must be greater than 0.0. If *trYAxisType* is `IrregularAxis`, *trYMinF* is constrained to lie within the bounds of the irregular coordinate system established by *trYCoordPoints*. In other words, you can establish a subset of the irregular coordinate system using *trYMinF* and *trYMaxF*, but you can expand the range of the coordinate system only by modifying the *trYCoordPoints* array. If you set *trYCoordPoints* and do not set *trYMinF* explicitly in the same call, *trYMinF* is set to the minimum coordinate value contained in *trYCoordPoints*.

This resource may be intercepted or disabled by:

- **ContourPlot**
- **XyPlot**

Default: 0.0

trYMaxF

Specifies a maximumum 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 *trYAxisType* is `IrregularAxis`, *trYMinF* is constrained to lie within the bounds of the irregular coordinate system established by *trYCoordPoints*. In other words, you can establish a subset of the irregular coordinate system using *trYMaxF* and *trYMinF*, but you can expand the range of the coordinate system only by modifying the *trYCoordPoints* array. If you set *trYCoordPoints* and do not set *trYMaxF* explicitly in the same call, *trYMaxF* is set to the maximum coordinate value contained in *trYCoordPoints*.

This resource may be intercepted or disabled by:

- **ContourPlot**
- **XyPlot**

Default: 0.0

trYReverse

If the boolean resource *trYReverse* is set True, the direction of the Y Axis coordinate values is reversed: if *trAyisType* is `LogAxis` or `LinearAxis` or if *trAxisType* is `IrregularAxis` and *trYCoordPoints* is monotonically increasing, the coordinate values will increase from top to bottom rather than from bottom to top.

This resource may be intercepted or disabled by:

- **ContourPlot**

Default: False

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.

This resource may be intercepted or disabled by:

- **ContourPlot**
- **XyPlot**

V4.0 NOTE: Large values of this parameter (> ~10.0) may cause instability in the irregular transformation.

Default: 2.0

trYSamples

Specifies the number of sample points over the Y-Axis coordinate system **IrregularTransObj** 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**
- **XyPlot**

Default: 9

$Revision: 1.9 $ $Date: 1995/07/24 23:22:08 $

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	TYPE	ACCESS	
CLASS	DEFAULT		
lbLabelBarOn	NhlTBoolean	RCSG	
lbLabelBarOn	True		
lbOrientation	NhlTOrientation	RCSG	
LbOrientation	"Vertical"		
lbJustification	NhlTJustification	RCSG	
LbJustification	"BottomLeft"		
lbBoxMajorExtentF	NhlTFloat	RCSG	
LbBoxMajorExtentF	1.0		
lbBoxMinorExtentF	NhlTFloat	RCSG	
LbBoxMinorExtentF	0.33		
lbBoxCount	NhlTInteger	RCSG	
LbBoxCount	16		
lbBoxSizing	NhlTlbBoxSizingMode	RCSG	
LbBoxSizing	"UniformSizing"		
lbAutoManage	NhlTBoolean	RCSG	
LbAutoManage	True		
lbLabelOffsetF	NhlTFloat	RCSG	
LbLabelOffsetF	0.1		
lbTitleOffsetF	NhlTFloat	RCSG	
LbTitleOffsetF	0.03		
lbLeftMarginF	NhlTFloat	RCSG	
LbLeftMarginF	0.05		
lbRightMarginF	NhlTFloat	RCSG	
LbRightMarginF	0.05		
lbBottomMarginF	NhlTFloat	RCSG	

LbBottomMarginF	0.05	
lbTopMarginF	NhlTFloat	RCSG
LbTopMarginF	0.05	
lbMonoFillColor	NhlTBoolean	RCSG
LbMonoFillColor	False	
lbFillColor	NhlTColorIndex	RCSG
LbFillColor	"Foreground"	
lbFillColors	NhlTColorIndexGenArray	RCSG
LbFillColors	<dynamic>	
lbMonoFillPattern	NhlTBoolean	RCSG
LbMonoFillColor	False	
lbFillPattern	NhlTFillIndex	RCSG
LbFillPattern	"SolidFill"	
lbFillPatterns	NhlTFillIndexGenArray	RCSG
LbFillPatterns	<dynamic>	
lbMonoFillScale	NhlTBoolean	RCSG
LbMonoFillScale	True	
lbFillScaleF	NhlTFloat	RCSG
LbFillScaleF	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
LbLabelAngleF	0.0	
lbLabelAlignment	NhlTlbLabelAlignmentMode	RCSG
LbLabelAlignment	"BoxCenters"	
lbLabelDirection	NhlTTextDirection	RCSG
LbLabelDirection	"Across"	
lbLabelJust	NhlTJustification	RCSG
LbLabelJust	"CenterCenter"	
lbLabelFont	NhlTFont	RCSG
Font	"pwritx"	
lbLabelFontColor	NhlTInteger	RCSG
LbLabelFontColor	"Foreground"	

lbLabelFontHeightF	NhlTFloat	RCSG
LbLabelFontHeightF	0.02	
lbLabelFontAspectF	NhlTFloat	RCSG
LbLabelFontAspectF	1.0	
lbLabelFontThicknessF	NhlTFloat	RCSG
LbLabelFontThicknessF	1.0	
lbLabelFontQuality	NhlTFontQuality	RCSG
LbLabelFontQuality	"High"	
lbLabelConstantSpacingF	NhlTFloat	RCSG
LbLabelConstantSpacingF	0.0	
lbLabelFuncCode	NhlTCharacter	RCSG
LbLabelFuncCode	':'	
lbLabelStride	NhlTInteger	RCSG
LbLabelStride	1	
lbTitleString	NhlTString	RCSG
LbTitleString	<dynamic>	
lbTitleOn	NhlTBoolean	RCSG
lbTitleOn	True	
lbTitlePosition	NhlTPosition	RCSG
LbTitlePosition	"Top"	
lbTitleExtentF	NhlTFloat	RCSG
LbTitleExtentF	0.15	
lbTitleAngleF	NhlTFloat	RCSG
LbTitleAngleF	0.0	
lbTitleDirection	NhlTTextDirection	RCSG
LbTitleDirection	"Across"	
lbTitleFont	NhlTFont	RCSG
Font	"pwritx"	
lbTitleFontColor	NhlTInteger	RCSG
LbTitleFontColor	"Foreground"	
lbTitleJust	NhlTJustification	RCSG
LbTitleJust	"CenterCenter"	
lbTitleFontHeightF	NhlTFloat	RCSG
LbTitleFontHeightF	0.025	
lbTitleFontAspectF	NhlTFloat	RCSG
LbTitleFontAspectF	1.0	
lbTitleFontThicknessF	NhlTFloat	RCSG
LbTitleFontThicknessF	1.0	
lbTitleFontQuality	NhlTFontQuality	RCSG
LbTitleFontQuality	"High"	

lbTitleConstantSpacingF LbTitleConstantSpacingF	NhlTFloat 0.0	RCSG
lbTitleFuncCode LbTitleFuncCode	NhlTCharacter ':'	RCSG
lbBoxLinesOn lbBoxLinesOn	NhlTBoolean True	RCSG
lbBoxLineColor LbBoxLineColor	NhlTInteger "Foreground"	RCSG
lbBoxLineThicknessF LbBoxLineThicknessF	NhlTFloat 1.0	RCSG
lbBoxLineDashPattern LbBoxLineDashPattern	NhlTInteger 0	RCSG
lbBoxLineDashSegLenF lbBoxLineDashSegLenF	NhlTFloat 0.15	RCSG
lbPerimOn lbPerimOn	NhlTBoolean True	RCSG
lbPerimColor LbPerimColor	NhlTInteger "Foreground"	RCSG
lbPerimFill LbPerimFill	NhlTInteger "HollowFill"	RCSG
lbPerimFillColor LbPerimFillColor	NhlTInteger "Foreground"	RCSG
lbPerimThicknessF LbPerimThicknessF	NhlTFloat 1.0	RCSG
lbPerimDashPattern LbPerimDashPattern	NhlTInteger 0	RCSG
lbPerimDashSegLenF lbPerimDashSegLenF	NhlTFloat 0.15	RCSG
lbFillBackground LbFillBackground	NhlTInteger "Transparent"	RCSG
lbFillLineThicknessF LbFillLineThicknessF	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 to 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

See also

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

Copyright

Copyright 1987,1988,1989,1991,1993,1994,1995
University Corporation for Atmospheric Research
All Rights Reserved

$Revision: 1.22 $ $Date: 1995/07/24 18:21:44 $

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**

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**

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**

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**

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**

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**

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**

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**

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**

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**

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**

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**

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 through 30 and 33 through 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.

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

$Revision: 1.14 $ $Date: 1995/07/23 18:38:37 $

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

	LgItemTypes	<dynamic>	
lgMonoDashIndex	NhlTBoolean		RCSG
	LgMonoDashIndex	True	
lgDashIndex	NhlTDashIndex		RCSG
	LgDashIndex	"SolidLine"	
lgDashIndexes	NhlTDashIndexGenArray		RCSG
	LgDashIndexes	<dynamic>	
lgMonoMarkerIndex	NhlTBoolean		RCSG
	LgMonoMarkerIndex	True	
lgMarkerIndex	NhlTMarkerIndex		RCSG
	LgMarkerIndex	"default"	
lgMarkerIndexes	NhlTMarkerIndexGenArray		RCSG
	LgMarkerIndexes	<dynamic>	
lgLineLabelStrings	NhlTStringGenArray		RCSG
	LgLineLabelStrings	<dynamic>	
lgMonoLineColor	NhlTBoolean		RCSG
	LgMonoLineColor	False	
lgLineColor	NhlTColorIndex		RCSG
	LgLineColor	"Foreground"	
lgLineColors	NhlTColorIndexGenArray		RCSG
	LgLineColors	<dynamic>	
lgMonoMarkerColor	NhlTBoolean		RCSG
	LgMonoMarkerColor	False	
lgMarkerColor	NhlTColorIndex		RCSG
	LgMarkerColor	"Foreground"	
lgMarkerColors	NhlTColorIndexGenArray		RCSG
	LgMarkerColors	<dynamic>	
lgMonoLineDashSegLen	NhlTBoolean		RCSG
	LgMonoLineDashSegLen	True	
lgLineDashSegLenF	NhlTFloat		RCSG
	LgLineDashSegLenF	0.15	
lgLineDashSegLens	NhlTFloatGenArray		RCSG
	LgLineDashSegLens	NULL	
lgMonoLineThickness	NhlTBoolean		RCSG
	LgMonoLineThickness	True	
lgLineThicknessF	NhlTFloat		RCSG
	LgLineThicknessF	1.0	
lgLineThicknesses	NhlTFloatGenArray		RCSG
	LgLineThicknesses	<dynamic>	
lgMonoMarkerThickness	NhlTBoolean		RCSG
	LgMonoMarkerThickness	True	

lgMarkerThicknessF	NhlTFloat	RCSG
LgMarkerThicknessF	1.0	
lgMarkerThicknesses	NhlTFloatGenArray	RCSG
LgMarkerThicknesses	<dynamic>	
lgMonoLineLabelFontHeight	NhlTBoolean	RCSG
LgMonoLineLabelFontHeight	True	
lgLineLabelFontHeightF	NhlTFloat	RCSG
LgLineLabelFontHeightF	0.01	
lgLineLabelFontHeights	NhlTFloatGenArray	RCSG
LgLineLabelFontHeights	<dynamic>	
lgMonoMarkerSize	NhlTBoolean	RCSG
LgMonoMarkerSize	True	
lgMarkerSizeF	NhlTFloat	RCSG
LgMarkerSizeF	0.01	
lgMarkerSizes	NhlTFloatGenArray	RCSG
LgMarkerSizes	<dynamic>	
lgLabelStrings	NhlTStringGenArray	RCSG
LgLabelStrings	<dynamic>	
lgItemPositions	NhlTFloatGenArray	RCSG
LgItemPositions	NULL	
lgMonoLineLabelFontColor	NhlTBoolean	RCSG
LgMonoLineLabelFontColor	False	
lgLineLabelFontColor	NhlTColorIndex	RCSG
LgLineLabelFontColor	"Foreground"	
lgLineLabelFontColors	NhlTColorIndexGenArray	RCSG
LgLineLabelFontColors	<dynamic>	
lgLineLabelsOn	NhlTBoolean	RCSG
LgLineLabelsOn	True	
lgLineLabelFont	NhlTFont	RCSG
LgLineLabelFont	"pwritx"	
lgLineLabelFontAspectF	NhlTFloat	RCSG
LgLineLabelFontAspectF	1.0	
lgLineLabelFontThicknessF	NhlTFloat	RCSG
LgLineLabelFontThicknessF	1.0	
lgLineLabelFontQuality	NhlTFontQuality	RCSG
LgLineLabelFontQuality	"High"	
lgLineLabelConstantSpacingF	NhlTFloat	RCSG
LgLineLabelConstantSpacingF	0.0	
lgLineLabelFuncCode	NhlTCharacter	RCSG
LgLineLabelFuncCode	':'	

lgLabelsOn	NhlTBoolean	RCSG
LgLabelsOn	True	
lgLabelPosition	NhlTPosition	RCSG
LgLabelPosition	"Right"	
lgLabelAngleF	NhlTFloat	RCSG
LgLabelAngleF	0.0	
lgLabelAlignment	NhlTlgLabelAlignmentMode	RCSG
LgLabelAlignment	"ItemCenters"	
lgLabelDirection	NhlTTextDirection	RCSG
LgLabelDirection	"Across"	
lgLabelJust	NhlTJustification	RCSG
LgLabelJust	"CenterCenter"	
lgLabelFont	NhlTFont	RCSG
Font	"pwritx"	
lgLabelFontColor	NhlTColorIndex	RCSG
LgLabelFontColor	"Foreground"	
lgLabelFontHeightF	NhlTFloat	RCSG
LgLabelFontHeightF	0.02	
lgLabelFontAspectF	NhlTFloat	RCSG
LgLabelFontAspectF	1.0	
lgLabelFontThicknessF	NhlTFloat	RCSG
LgLabelFontThicknessF	1.0	
lgLabelFontQuality	NhlTFontQuality	RCSG
LgLabelFontQuality	"High"	
lgLabelConstantSpacingF	NhlTFloat	RCSG
LgLabelConstantSpacingF	0.0	
lgLabelFuncCode	NhlTCharacter	RCSG
LgLabelFuncCode	':'	
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
LgTitleAngleF	0.0	
lgTitleDirection	NhlTTextDirection	RCSG

LgTitleDirection	"Across"	
lgTitleFont	NhlTFont	RCSG
Font	"pwritx"	
lgTitleFontColor	NhlTColorIndex	RCSG
LgTitleFontColor	"Foreground"	
lgTitleJust	NhlTJustification	RCSG
LgTitleJust	"CenterCenter"	
lgTitleFontHeightF	NhlTFloat	RCSG
LgTitleFontHeightF	0.025	
lgTitleFontAspectF	NhlTFloat	RCSG
LgTitleFontAspectF	1.0	
lgTitleFontThicknessF	NhlTFloat	RCSG
LgTitleFontThicknessF	1.0	
lgTitleFontQuality	NhlTFontQuality	RCSG
LgTitleFontQuality	"High"	
lgTitleConstantSpacingF	NhlTFloat	RCSG
LgTitleConstantSpacingF	0.0	
lgTitleFuncCode	NhlTCharacter	RCSG
LgTitleFuncCode	':'	
lgBoxLinesOn	NhlTBoolean	RCSG
LgBoxLinesOn	False	
lgBoxLineColor	NhlTColorIndex	RCSG
LgBoxLineColor	"Foreground"	
lgBoxLineThicknessF	NhlTFloat	RCSG
LgBoxLineThicknessF	1.0	
lgBoxLineDashPattern	NhlTDashIndex	RCSG
LgBoxLineDashPattern	"SolidLine"	
lgBoxLineDashSegLenF	NhlTFloat	RCSG
LgBoxLineDashSegLenF	0.15	
lgPerimOn	NhlTBoolean	RCSG
LgPerimOn	False	
lgPerimColor	NhlTColorIndex	RCSG
LgPerimColor	"Foreground"	
lgPerimFill	NhlTFillIndex	RCSG
LgPerimFill	"HollowFill"	
lgPerimFillColor	NhlTColorIndex	RCSG
LgPerimFillColor	"Background"	
lgPerimThicknessF	NhlTFloat	RCSG
LgPerimThicknessF	1.0	
lgPerimDashPattern	NhlTDashIndex	RCSG
LgPerimDashPattern	"SolidLine"	

```
|-------------------------------------------------------------|
| lgPerimDashSegLenF            NhlTFloat               RCSG  |
|       LgPerimDashSegLenF              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 accomodate 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 to 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*.

Status

Support functions

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

See also

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

Copyright

Copyright 1987,1988,1989,1991,1993,1994,1995
University Corporation for Atmospheric Research
All Rights Reserved

$Revision: 1.22 $ $Date: 1995/07/24 18:21:45 $

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 towards the body

of the text (i.e. positive for right-justified text is towards 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 indices.

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 indices.

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**.

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.

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

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:      LogLinTransObj, PlotManager
```

Resources

Local resources

The LogLinPlot object has no local resources.

Composite resources

LogLinTransObj resources

You can access all **LogLinTransObj** 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

The **LogLinPlot** object has no **Draw** method of its own. Its primary function is to act as a base plot over which other plot objects may be overlaid, transforming their coordinates into a log or linear coordinate space. Although it has no **Draw** method, the **LogLinPlot** object *does* allow you to draw immediate mode polylines, using the **Transform** class **DataPolyline** functions.

You can transform a plot object into log or linear coodinate space by overlaying it on a **LogLinPlot** object acting as a base plot. The log or linear coordinate space may emcompass 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 **LogLinTransObj** composite class member.

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.

The immediate mode lines draw into the **Workstation** object as soon as you call the **DataPolyline** function. They are *not* part of the object because they do not appear when you execute the **Draw** method for the **LogLinPlot** object. At the present time there is no way to save these lines into a segment.

Support functions

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

Status

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

See also

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

Copyright

Copyright 1987,1988,1989,1991,1993,1994,1995
University Corporation for Atmospheric Research
All Rights Reserved

$Revision: 1.13 $ $Date: 1995/07/24 18:21:47 $

LogLinPlot class resource descriptions

The LogLinPlot class has no local resources.

$Revision: 1.4 $ $Date: 1995/07/23 18:38:41 $

LogLinTransObj class

The **LogLinTransObj** class manages forward and backward transformations in log or linear coordinate space.

Synopsis

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

Resources

Local resources

You may access **LogLinTransObj** resources only as composite resources of certain objects that instantiate a **LogLinTransObj** child object. These include:

- **ContourPlot** object
- **LogLinPlot** object

LogLinTransObj Resource Set			
NAME	TYPE	ACCESS	
CLASS	DEFAULT		
trXMinF	NhlTFloat	RCSG	
TrXMinF	0.0		
trXMaxF	NhlTFloat	RCSG	
TrXMaxF	1.0		
trXLog	NhlTInteger	RCSG	
TrXLog	0		
trXReverse	NhlTInteger	RCSG	
TrXReverse	0		
trYMinF	NhlTFloat	RCSG	
TrYMinF	0.0		
trYMaxF	NhlTFloat	RCSG	
TrYMaxF	1.0		
trYLog	NhlTInteger	RCSG	
TrYLog	0		
trYReverse	NhlTInteger	RCSG	
TrYReverse	0		

Composite resources

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

Superclass resources

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

- **TransObj** resources

Description

The **LogLinTransObj** provides transformations in logarithmic or linear coordate space for plot objects belonging to the **Transform** class. It contains resources for specifying the coordinate extent along each of its axes. You can specify the direction of each axis. You can also independently specify whether each axis is to be logarithmic or linear.

You do not create objects of the **LogLinTransObj** class directly. You may, however, set its resources via certain objects that instantiate a **LogLinTransObj** child. Currently some plot objects, such as the **XyPlot** object, have their own resources for setting the **LogLinTransObj** resources. For these objects the **LogLinTransObj** resources are considered private. Other objects, such as the **ContourPlot** and the **LogLinPlot** object, depend on **LogLinTransObj** resources for control of the data coordinate space. For these objects, you use **LogLinTransObj** resources directly to control the object's coordinate space.

By default, the **ContourPlot** object sets the coordinate space to coincide with the bounds of the **ScalarField** object data space. However, by manipulating the **LogLinTransObj** resources you can define the coordinate space to intersect or encompass the data space arbitrarily. Note that the **IrregularTransObj** defines all the same resources as the **LogLinTransObj**. Therefore, you can position the **ContourPlot** object data space in the same way regardless of whether the **ScalarField** object data is irregularly spaced or not.

Support functions

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

Status

See also

- LogLinPlot object
- PlotManager object
- ContourPlot object
- XyPlot object
- TransObj object
- Obj object

Copyright

Copyright 1987,1988,1989,1991,1993,1994,1995

University Corporation for Atmospheric Research
All Rights Reserved

$Revision: 1.13 $ $Date: 1995/07/24 18:21:47 $

---

f

LogLinTransObj class resource descriptions

trXMinF

Specifies the minimum X coordinate value used to define a range along the X-Axis in a log or linear coordinate system. *trXMinF* must be less than *trXMaxF*.

This resource may be intercepted or disabled by:

- **ContourPlot**
- **XyPlot**

Default: 0.0

trXMaxF

Specifies the maximum X coordinate value used to define a range along the X-Axis in a log or linear coordinate system. *trXMaxF* must be greater than *trXMinF*.

This resource may be intercepted or disabled by:

- **ContourPlot**
- **XyPlot**

Default: 1.0

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**

Default: 0

trXReverse

If this boolean resource is True, the X-Axis coordinate system has its maximum value at the left end; otherwise the maximum value is at the right end.

This resource may be intercepted or disabled by:

- **ContourPlot**

Default: 0

trYMinF

Specifies the minimum Y coordinate value used to define a range along the Y-Axis in a log or linear coordinate system. *trYMinF* must be less than *trYMaxF*.

This resource may be intercepted or disabled by:

- **ContourPlot**
- **XyPlot**

Default: 0.0

trYMaxF

Specifies the maximum Y coordinate value used to define a range along the Y-Axis in a log or linear coordinate system. *trYMaxF* must be greater than *trYMinF*.

This resource may be intercepted or disabled by:

- **ContourPlot**
- **XyPlot**

Default: 1.0

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**

Default: 0

trYReverse

If this boolean resource is True, the Y-Axis coordinate system has its maximum value at the bottom end; otherwise the maximum value is at the top end.

This resource may be intercepted or disabled by:

- **ContourPlot**

Default: 0

$Revision: 1.7 $ $Date: 1995/07/24 23:22:10 $

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:      MapTransObj,PlotManager
```

Class-defined types

```
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 CLASS	TYPE DEFAULT	ACCESS
mpDataBaseVersion MpDataBaseVersion	NhlTString "NCARG_4.0"	G
mpShapeMode MpShapeMode	NhlTMapShapeMode "FixedAspectFitBB"	RCSG
mpAreaNames MpAreaNames	NhlTStringGenArray <array>	RCSG
mpAreaTypes MpAreaTypes	NhlTIntegerGenArray <array>	G
mpFixedAreaGroups MpFixedAreaGroups	NhlTIntegerGenArray <array>	G
mpDynamicAreaGroups MpDynamicAreaGroups	NhlTIntegerGenArray <array>	RCSG
mpAreaGroupCount MpAreaGroupCount	NhlTInteger 10	RCSG
mpOutlineOn MpOutlineOn	NhlTBoolean True	RCSG
mpOutlineDrawOrder MpOutlineDrawOrder	NhlTDrawOrder "PostDraw"	RCSG
mpOutlineBoundarySets MpOutlineBoundarySets	NhlTMapBoundarySets "Geophysical"	RCSG
mpOutlineSpecifiers MpOutlineSpecifiers	NhlTStringGenArray <array>	RCSG
mpAreaMaskingOn MpAreaMaskingOn	NhlTBoolean False	RCSG
mpMaskAreaSpecifiers MpMaskAreaSpecifiers	NhlTStringGenArray <array>	RCSG
mpFillOn MpFillOn	NhlTBoolean False	RCSG
mpFillDrawOrder MpFillDrawOrder	NhlTDrawOrder "Draw"	RCSG
mpFillBoundarySets MpFillBoundarySets	NhlTMapBoundarySets "Geophysical"	RCSG
mpFillPatternBackground MpFillPatternBackground	NhlTColorIndex "Background"	RCSG
mpMonoFillColor MpMonoFillColor	NhlTBoolean False	RCSG
mpFillColor MpFillColor	NhlTColorIndex "Foreground"	RCSG

mpFillColors MpFillColors	NhlTColorIndexGenArray <array>	RCSG
mpMonoFillPattern MpMonoFillPattern	NhlTBoolean True	RCSG
mpFillPattern MpFillPattern	NhlTFillIndex "SolidFill"	RCSG
mpFillPatterns MpFillPatterns	NhlTFillIndexGenArray <array>	RCSG
mpMonoFillScale MpMonoFillScale	NhlTBoolean True	RCSG
mpFillScaleF MpFillScaleF	NhlTFloat 1.0	RCSG
mpFillScales MpFillScales	NhlTFloatGenArray <array>	RCSG
mpFillAreaSpecifiers MpFillAreaSpecifiers	NhlTStringGenArray <array>	RCSG
mpSpecifiedFillDirectIndexing MpSpecifiedFillDirectIndexing	NhlTBoolean False	RCSG
mpSpecifiedFillPriority MpSpecifiedFillPriority	NhlTSpecifiedFillPriority "GeophysicalPriority"	RCSG
mpSpecifiedFillColors MpSpecifiedFillColors	NhlTColorIndexGenArray <dynamic array>	RCSG
mpSpecifiedFillPatterns MpSpecifiedFillPatterns	NhlTFillIndexGenArray <dynamic array>	RCSG
mpSpecifiedFillScales MpSpecifiedFillScales	NhlTFloatGenArray <dynamic array>	RCSG
mpDefaultFillColor MpDefaultFillColor	NhlTColorIndex 106	RCSG
mpDefaultFillPattern MpDefaultFillPattern	NhlTFillIndex "SolidFill"	RCSG
mpDefaultFillScaleF MpDefaultFillScaleF	NhlTFloat 1.0	RCSG
mpOceanFillColor MpOceanFillColor	NhlTColorIndex 104	RCSG
mpOceanFillPattern MpOceanFillPattern	NhlTFillIndex "SolidFill"	RCSG
mpOceanFillScaleF MpOceanFillScaleF	NhlTFloat 1.0	RCSG
mpLandFillColor MpLandFillColor	NhlTColorIndex 57	RCSG

mpLandFillPattern	NhlTFillIndex	RCSG
MpLandFillPattern	"SolidFill"	
mpLandFillScaleF	NhlTFloat	RCSG
MpLandFillScaleF	1.0	
mpInlandWaterFillColor	NhlTColorIndex	RCSG
MpInlandWaterFillColor	104	
mpInlandWaterFillPattern	NhlTFillIndex	RCSG
MpInlandWaterFillPattern	"SolidFill"	
mpInlandWaterFillScaleF	NhlTFloat	RCSG
MpInlandWaterFillScaleF	1.0	
mpGeophysicalLineColor	NhlTColorIndex	RCSG
MpGeophysicalLineColor	"Foreground	
mpGeophysicalLineDashPattern	NhlTDashIndex	RCSG
MpGeophysicalLineDashPattern	"SolidLine"	
mpGeophysicalLineDashSegLenF	NhlTFloat	RCSG
MpGeophysicalLineDashSegLenF	<dynamic>	
mpGeophysicalLineThicknessF	NhlTFloat	RCSG
MpGeophysicalLineThicknessF	1.0	
mpUSStateLineColor	NhlTColorIndex	RCSG
MpUSStateLineColor	"Foreground"	
mpUSStateLineDashPattern	NhlTDashIndex	RCSG
MpUSStateLineDashPattern	"SolidLine"	
mpUSStateLineDashSegLenF	NhlTFloat	RCSG
MpUSStateLineDashSegLenF	<dynamic>	
mpUSStateLineThicknessF	NhlTFloat	RCSG
MpUSStateLineThicknessF	1.0	
mpNationalLineColor	NhlTColorIndex	RCSG
MpNationalLineColor	"Foreground"	
mpNationalLineDashPattern	NhlTDashIndex	RCSG
MpNationalLineDashPattern	"SolidLine"	
mpNationalLineDashSegLenF	NhlTFloat	RCSG
MpNationalLineDashSegLenF	<dynamic>	
mpNationalLineThicknessF	NhlTFloat	RCSG
MpNationalLineThicknessF	1.0	
mpGridAndLimbOn	NhlTBoolean	RCSG
MpGridAndLimbOn	True	
mpGridAndLimbDrawOrder	NhlTDrawOrder	RCSG
MpGridAndLimbDrawOrder	"PostDraw"	
mpGridSpacingF	NhlTFloat	RCSG
MpGridSpacingF	0.1	
mpGridMaskMode	NhlTMapGridMaskMode	RCSG

MpGridMaskMode	"MaskNone"	
mpGridLineColor	NhlTColorIndex	RCSG
MpGridLineColor	"Foreground"	
mpGridLineDashPattern	NhlTDashIndex	RCSG
MpGridLineDashPattern	"SolidLine"	
mpGridLineDashSegLenF	NhlTFloat	RCSG
MpGridLineDashSegLenF	<dynamic>	
mpGridLineThicknessF	NhlTFloat	RCSG
MpGridLineThicknessF	1.0	
mpLimbLineColor	NhlTColorIndex	RCSG
MpLimbLineColor	"Foreground"	
mpLimbLineDashPattern	NhlTDashIndex	RCSG
MpLimbLineDashPattern	"SolidLine"	
mpLimbLineDashSegLenF	NhlTFloat	RCSG
MpLimbLineDashSegLenF	<dynamic>	
mpLimbLineThicknessF	NhlTFloat	RCSG
MpLimbLineThicknessF	1.0	
mpPerimOn	NhlTBoolean	RCSG
MpPerimOn	True	
mpPerimDrawOrder	NhlTDrawOrder	RCSG
MpPerimDrawOrder	"Draw"	
mpPerimLineColor	NhlTColorIndex	RCSG
MpPerimLineColor	"Foreground"	
mpPerimLineDashPattern	NhlTDashIndex	RCSG
MpPerimLineDashPattern	"SolidLine"	
mpPerimLineDashSegLenF	NhlTFloat	RCSG
MpPerimLineDashSegLenF	<dynamic>	
mpPerimLineThicknessF	NhlTFloat	RCSG
MpPerimLineThicknessF	1.0	
mpLabelsOn	NhlTBoolean	RCSG
MpLabelsOn	True	
mpLabelDrawOrder	NhlTDrawOrder	RCSG
MpLabelDrawOrder	"PostDraw"	
mpLabelFontHeightF	NhlTFloat	RCSG
MpLabelFontHeightF	<dynamic>	
mpLabelFontColor	NhlTColorIndex	RCSG
MpLabelFontColor	True	

Composite resources

MapTransObj resources

You can access all **MapTransObj** resources except for the following disabled resource:

- mpPreserveAspectRatio

The **MapTransObj** 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 attempts to set *pmTickMarkDisplayMode* to a value other `NoCreate` will result in a warning, since 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, and to choose individually or by group which of the 583 areas defined in the **MapPlot** database you want to appear. You can outline areas, you can fill areas, or you can *mask* areas, allowing previously rendered plot elements to remain visible.

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. Using these facilities, you can 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 Grenwich 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* to 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 on by default. Turn it off by setting *mpPerimOn* to False. The drawing order is specified using *mpPerimDrawOrder*.

Labels
: Labels are turned on by default. Turn them off by setting *mpLabelsOn* to False. 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 MapTransObj transformation object. Note that it is possible to overlay another plot object on a **MapPlot** object using one of the projections provided by the **MapTransObj**, 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*, *mpGridAndLimbOn*, *mpPerimOn*, and *mpLabelsOn* to False.

The map projections provided by the **MapTransObj** 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.

The MapPlot database

The **MapPlot** database currently contains 583 areas uniquely identified by name. Although the names are unique, the areas they refer to are not necessarily. An area may overlap one or more other areas' extents. For instance, the area "North-and-South-America" overlaps the continental portion of all countries of the Americas, and "US-Lake-Superior" provides a higher resolution outline of basically the same area as "Lake-Superior". Consider the area name as the primary key to the data.

The current version of the database is several years out of date. Since it can be expected to change from time to time, both to accommodate the current political realities and also to incorporate higher resolution data, **MapPlot** provides a means of maintaining compatibility, based on a string resource, *mpDataBaseVersion*, that allows the user to specify the version of the database required.

Area names

You can retrieve a list of all the area names in the **MapPlot** database using 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 elements of this 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.

Pre-defined string constants

Somewhat related to the area types are the pre-defined string constants that specify areas by catagory. 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 catagories described by these string constants can overlap. The catagory "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 catagory. The string "NullArea" specifies no area at all; it is the default value of all the area specifier array resources.

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 group of areas by catagory 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 areas in the **MapPlot** database are actually a concatentation of three distinct boundary sets. The first set contains geophysical boundaries; the second is a union of the geophysical set with interior national boundaries; the third contains (at a higher resolution) the boundaries of the U.S. States. You can specify that **MapPlot** draw any combination of the three sets (including none of them) using resources of type NhlTMapBoundarySets. For outlines and fill these resources are *mpOutlineBoundarySets* and *mpFillBoundarySets*.

The string array resources allow individual specification of areas and area catagories. 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*. The specifier strings you use for these resources may be any of the **MapPlot** pre-defined string constants, explicit names from the **MapPlot** database, or matching substrings. All specifier strings are case-insensitive.

Substring matching

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.

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 catagories, 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 catagory.

Area fill

Since area fill is more compute intensive, and may not be appropriate for many simple plotting tasks, **MapPlot** turns it off by default. Set *mpFillOn* to 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 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 ocean.

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 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 to 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. 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".

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 explcitly 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 `NoColor` (-1); for *mpSpecifiedFillPatterns* the special fill pattern index `NoPattern` (-1); 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 catagories 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. The resource *mpGridSpacingF* specifies the distance in degrees between grid lines both for longitude and for latitude. 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 to False. When labels are turned on the text "NP", "SP", "EQ", "GM", and "ID" is written at the positions of respectively 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 **MapTransObj** 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

5: 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.

6: The resource *mpAreaTypes* needs to be made into an enumerated resource.

See also

- NhlCreate function
- NhlSetValues function
- NhlGetValues function
- MapTransObj object

- PlotManager object
- Title object
- LabelBar object
- Legend object
- Transform object
- View object

Copyright

Copyright 1987,1988,1989,1991,1993,1994,1995
University Corporation for Atmospheric Research
All Rights Reserved

$Revision: 1.17 $ $Date: 1995/07/24 23:22:10 $

MapPlot class resource descriptions

mpDataBaseVersion

This resource defines the version of the map database used by the **MapPlot** object. Currently it is read-only, since only one version of the map database now exists. As new versions of the map database become available, you will be able to set this resource to access a particular version of the database.

Default: "NCARG_4.0"

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 the **MapPlot** database. There is a unique name for each distinct area. You can also set this resource to rename the area names, perhaps to another language than English. You must set the complete list of names at the same time. The easiest way to change the names of only a few areas would be to perform a **GetValues** first, change the names of the selected areas, then do a **SetValues**. Note that 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. Also, keep in mind that the order of areas is fixed: a particular element of the array always refers to the same area in the MapPlot database regardless of the name you give it.

Default: See heading "Area Name" in **MapPlot** database table

mpAreaTypes

The areas in the **MapPlot** database are currently classified into eight *area types*. These types, which cannot be modified, are as follows:

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

The read-only array resource *mpAreaTypes* allows you to retrieve the integer value of the type of each area in the **MapPlot** database. Each element gives the type of the area named by the corresponding element of the *mpAreaNames* array.

V4.0 Status Note 6

Default: See heading "Type" in **MapPlot** database table

mpFixedAreaGroups

MapPlot maintains two arrays that assign area group numbers to each area in the map 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. This 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*.

Default: See heading "Fixed Group" in **MapPlot** database table

mpDynamicAreaGroups

MapPlot maintains two arrays that assign area group numbers to each area in the map 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. This 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 only 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 define any number of area groups up to a maximum of 255.

Default: See heading "Dynamic Group" in **MapPlot** database table

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 256. 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 pre-draw phase and anything drawn during the draw and post-draw phases.

`Draw`

Draw the area outlines during the standard draw; the lines will overlay any elements drawn during the pre-draw 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 post-draw 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 post-draw 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 (although the database is several years out of date). 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 U.S. States and inland water features (using data of higher resolution than the national and geophysical area

outlines). A few areas outside the U.S. are included as well: the Bahamas, and a few lakes extending into Canadian territory. You may add other geophysical and/or national areas by name using the *mpOutlineSpecifiers* resource.

`GeophysicalAndUSStates`

MapPlot will draw area outlines of all global geophyical feature and also U.S. States and inland water features. 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. There is no effect if area names are specified using the *mpOutlineSpecifiers* resource, since all possible area outlines appear regardless.

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*. Among the specifiers, 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 sharing a common substring.

Default: A single element array containing the value "NullArea".

mpAreaMaskingOn

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

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*. Among the specifiers 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 sharing a common substring.

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.

Default: A single element array containing the value "NullArea".

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 pre-draw phase and anything drawn during the draw and post-draw phases.

`Draw`

Draw area fill during the standard draw; the fill overlays anything drawn during the pre-draw phase but underlies any other **MapPlot** elements drawn during the draw phase and anything drawn during the post-draw phase.

`PostDraw`

Draw area fill after the standard draw; the fill overlays anything drawn during the pre-draw and draw phases but underlies any other **MapPlot** elements drawn during the post-draw 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 individually fill the states of the U.S. States and inland water features (using data of higher resolution than the national and geophysical area outlines). A few areas outside the U.S. are included as well: the Bahamas, and a few lakes extending into Canadian territory. In addition, you may fill other geophysical and/or national areas by name using the *mpFillAreaSpecifiers* resource.

`GeophysicalAndUSStates`

MapPlot will draw area fill for all global geophyical feature and also individually fill the U.S. states and inland water features. 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.

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 reource 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: 106,104,57,104,19,23,41,53,60,88.

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 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 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 facter 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*. 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.

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 do not explicitly set fill attributes of areas specified in the *mpFillAreaSpecifiers*, either by not setting the specified attribute resources, or by specifying individual elements as "unset", **MapPlot** uses its normal method of setting fill attributes based on area group assignments. The group numbers are 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 any individual area by prepending an exclamation point ('!') to the area's name.

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.

Default: A single element array containing the value "NullArea".

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

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 ('!').

Default: `GeophysicalPriority` Default: NULL

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:

mpSpecifiedFillScales

You may use this array resource to set explicitly 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 perceived 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 value 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: 106

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 value greater than `Transparent` (-1). 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: 104

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 value greater than `Transparent` (-1). 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: 57

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 value greater than `Transparent` (-1). 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: 104

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 *USStateslLineColor*.

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 *USStateslLineDashPattern*.

Default: 0

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 *USStateslLineDashSegLenF*. **MapPlot** sets the default value of *mpNationalLineDashSegLenF* dynamically based upon the ratio of the actual plot

viewport width to a reference viewport width of 0.5 and a reference dash segment length of 0.075.

Default: 0.075 (for a viewport width of 0.5)

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 *USStateslLineThicknessF*.

Default: 1.0

mpUSStateLineColor

This resource sets the HLU index of a color used to render interior 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 *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: 0

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 upon the ratio of the actual plot viewport width to a reference viewport width of 0.5 and a reference dash segment length of 0.075.

Default: 0.075 (for a viewport width of 0.5)

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: 0

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 upon the ratio of the actual plot viewport width to a reference viewport width of 0.5 and a reference dash segment length of 0.075.

Default: 0.075 (for a viewport width of 0.5)

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

MapPlotfill drawn during the pre-draw phase but underlie other pre-draw phase **MapPlot** elements and anything drawn during the draw and post-draw phases.

`Draw`
Draw grid lines during the standard draw; the grid lines overlay anything drawn during the pre-draw phase and **MapPlot** fill drawn during the draw phase but underlie other draw phase **MapPlot** elements and anything drawn during the post-draw phase.

`PostDraw`
Draw grid lines after the standard draw; the grid lines overlay anything drawn during the pre-draw and draw phases and **MapPlot** fill drawn during the post-draw phase but underlie other post-draw phase **MapPlot** elements.

Default: `PostDraw`

mpGridSpacingF

This resource specifies the spacing in degrees between **MapPlot** grid lines representing latitude and longitude.

Default: 0.1

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`

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: 0

mpGridLineDashSegLenF

This resource specifies the length of each segment of the dash patterns used 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 upon the ratio of the actual plot viewport width to a reference viewport width of 0.5 and a reference dash segment length of 0.075.

Default: 0.075 (for a viewport width of 0.5)

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: 0

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 upon the ratio of the actual plot viewport width to a reference viewport width of 0.5 and a reference dash segment length of 0.075.

Default: 0.075 (for a viewport width of 0.5)

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 **MapTransObj** resource, *mpEllipticalBoundary* this perimeter may have a rectangular or elliptical shape.

Default: True

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 pre-draw phase except **MapPlot** labels but underlies anything drawn during the draw and post-draw phases.

`Draw`

Draw the perimeter during the standard draw; the perimeter overlays anything drawn during the pre-draw phase and any other **MapPlot** elements (except **MapPlot** labels) drawn during the draw phase but underlies anything drawn during the post-draw phase.

`PostDraw`

Draw the perimeter after the standard draw; the perimeter overlays anything drawn during the pre-draw and draw phases and all other **MapPlot** elements (except **MapPlot** labels) drawn during the post-draw 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: 0

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 upon the ratio of the actual plot viewport width to a reference viewport width of 0.5 and a reference dash segment length of 0.075.

Default: 0.075 (for a viewport width of 0.5)

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 date line ("ID") North Pole ("NP") and South Pole ("SP").

Default: True

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 pre-draw phase but underlie anything drawn during the draw and post-draw phases.

`Draw`
: Draw labels during the standard draw; the labels overlay anything drawn during the pre-draw phase and any other **MapPlot** elements drawn during the draw phase but underlie anything drawn during the post-draw phase.

`PostDraw`
: Draw labels after the standard draw; the labels overlay anything drawn during the pre-draw and draw phases and all other **MapPlot** elements drawn during the post-draw 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>

mpLabelFontColor

This resource specifies the hlu color index used to draw the characters used for **MapPlot** labels.

Default: True

$Revision: 1.11 $ $Date: 1995/07/24 18:21:49 $

MapTransObj class

The **MapTransObj** object manages forward and backward transformations in the coordinate space defined by various map projections.

Synopsis

```
Header file:            ncarg/hlu/MapTransObj.h
Class name:             mapTransObjClass
Class pointer:          <Not referenceable>
Fortran class function: <Not referenceable>
Supeclass:              TransObj
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 **MapTransObj** resources only through objects that instantiate a **MapTransObj** child object. These include:

- **MapPlot** object

MapTransObj Resource Set		
NAME	TYPE	ACCESS

	CLASS	DEFAULT	
mpProjection	NhlTProjection		RCSG
MpProjection		"CylindricalEquidistant"	
mpPreserveAspectRatio	NhlTBoolean		RCSG
MpPreserveAspectRatio		True	
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	NhlTFloat	RCSG
MpRightCornerLatF	0.0	
mpRightCornerLonF	NhlTFloat	RCSG
MpRightCornerLonF	0.0	
mpLeftPointLatF	NhlTFloat	RCSG
MpLeftPointLatF	0.0	
mpLeftPointLonF	NhlTFloat	RCSG
MpLeftPointLonF	0.0	
mpRightPointLatF	NhlTFloat	RCSG
MpRightPointLatF	0.0	
mpRightPointLonF	NhlTFloat	RCSG
MpRightPointLonF	0.0	
mpBottomPointLatF	NhlTFloat	RCSG
MpBottomPointLatF	0.0	
mpBottomPointLonF	NhlTFloat	RCSG
MpBottomPointLonF	0.0	
mpTopPointLatF	NhlTFloat	RCSG
MpTopPointLatF	0.0	
mpTopPointLonF	NhlTFloat	RCSG
MpTopPointLonF	0.0	

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 MpEllipticalBoundary	NhlTBoolean False	RCSG

Composite resources

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

Superclass resources

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

- **TransObj** resources

Description

The **MapTransObj** transforms a two-dimensional rectangular data coordinate space representing latitude along the vertical axis and longitude along the horizontal axis into any of 10 different map projections. The data space extent may coincide with, encompass, or arbitrarily intersect the projected extent.

You do not create objects of the **MapTransObj** class directly. You may, however, set its resources via objects that instantiate a **MapTransObj** child. Currently, the **MapPlot** object

is the only HLU library object with a **MapTransObj** child; it allows access to almost all **MapTransObj** resources, depending on it to provide transformations of data representing geophysical and political regions of the earth. However, by using the **MapPlot** as a base plot, you can provide map transformations for other plot object types.

You can force the map projection to maintain its true aspect ratio by setting the boolean resource *mpPreserveAspectRatio* True. This will cause **MapTransObj** center the projected area within the supplied viewport. Since in this case the projected area's boundaries probably do not coincide with the supplied viewport boundaries, the **MapTransObj** supplies four read-only resources that allow you to retrieve the edges of the actual projected area. These are *mpLeftMapPosF*, *mpRightMapPosF*, *mpBottomMapPosF*, and *mpTopMapPosF*.

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 **MapTransObj** 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 **MapTransObj** 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 **MapTransObj** 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

MapTransObj 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 **MapTransObj** keeps the the window coordinate system resources updated to their current values no matter which map limit mode is currently in effect.

When the conic projection, `LambertConformal`, is set, the **MapTransObj** ignores the projection center and rotation resources. Instead, use *mpLambertParallel1F* and *mpLambertParallel2F* to set the two latitudinal edges of the cone and *mpLambertMeridianF* its longitudinal center. If the projection is NhlSATELLITE, the **MapTransObj** 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.

You can also set a resource, *mpEllipticalBoundary*, that limits the perimeter of the viewable area to an ellipse inscribed within the rectangular viewport.

Support functions

The **MapTransObj** 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 MapTransObj 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
- TransObj object
- Obj object

Copyright

Copyright 1987,1988,1989,1991,1993,1994,1995
University Corporation for Atmospheric Research
All Rights Reserved

$Revision: 1.16 $ $Date: 1995/07/24 18:21:49 $

MapTransObj class resource descriptions

mpProjection

This resource of type NhlProjection 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`

mpPreserveAspectRatio

When this boolean resource is set True, the **MapTransObj** forces the projected area to maintain the true aspect ratio required by the projection. The projected area will be centered in the viewport provided, and will fill as much of the area as it can. If False, the projected area will stretch in one direction or the other as much as necessary to fill the supplied viewport.

This resource may be intercepted or disabled by:

- **MapPlot**

Default: True

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.

V4.0 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 Date Line.

V4.0 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 **MapTransObj** determines the extent of the projected globe that is mapped into the viewport area. It has 8 possible settings:

`MaximalArea`

The maximum viewable area of the globe allowed by the projection is mapped into the viewport area.

`LatLon`

MapTransObj 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, **MapTransObj** 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* **MapTransObj** 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, **MapTransObj** 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, **MapTransObj** issues a warning and resets *mpLimitMode* to NhlMAXIMALAREA.

V4.0 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, **MapTransObj** 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* **MapTransObj** issues a warning and exchanges the values. The **MapTransObj** 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, **MapTransObj** 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* **MapTransObj** 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, the **MapTransObj** 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`

MapTransObj 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`

MapTransObj 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`

MapTransObj 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. If you set one of the *mpLeftNPCF* to a value greater than *mpRightNPCF* or *mpBottomNPCF* to a value greater than *mpTopNPCF* **MapTransObj** issues a warning and exchanges the values. The **MapTransObj** 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 NhlLATLON, 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 should not be less than -180.0.

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 should not be greater than 180.0.

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, the **MapTransObj** 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, the **MapTransObj** 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, the **MapTransObj** 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 *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, the **MapTransObj** 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 *mpLambertParallel1F* are equal. Normally, *mpLambertParallel1F* and *mpLambertParallel1F* should not be more than 90.0 degrees apart.

V4.0 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 *mpLambertParallel1F* are equal. Normally, *mpLambertParallel1F* and *mpLambertParallel1F* should not be more than 90.0 degrees apart.

V4.0 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 as *mpCenterLonF* has for the other projections.

V4.0 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 towards 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

$Revision: 1.11 $ $Date: 1995/07/24 23:22:12 $

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 superclass. 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 1987,1988,1989,1991,1993,1994,1995
University Corporation for Atmospheric Research
All Rights Reserved

$Revision: 1.13 $ $Date: 1995/07/24 18:21:50 $

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"

$Revision: 1.6 $ $Date: 1995/07/23 18:38:50 $

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 1987,1988,1989,1991,1993,1994,1995
University Corporation for Atmospheric Research
All Rights Reserved

$Revision: 1.9 $ $Date: 1995/07/23 18:38:51 $

Obj class resource descriptions

The Obj class has no public resources.

$Revision: 1.4 $ $Date: 1995/07/23 18:38:52 $

<<<<<<< PSWorkstation.obj ======= f>>>>>>> 1.9

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:          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"		
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 1987,1988,1989,1991,1993,1994,1995
University Corporation for Atmospheric Research
All Rights Reserved

$Revision: 1.8 $ $Date: 1995/07/23 18:38:53 $

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"

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 PostScript output in DPI (Dots Per Inch). It is important to realize that printers that have a lower resolution, can print PostScript of a higher resolution without problems. However, if the PostScript is generated with a lower resolution then the printer, then 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.

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

$Revision: 1.5 $ $Date: 1995/07/23 18:38:53 $

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,    /* "Conditional" */
        NhlCONDITIONAL  = 2     /* "Always"      */
} 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 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	0.2	
pmLabelBarHeightF	NhlTFloat	RCSG
PmLabelBarHeightF	0.5	
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	0.45	
pmLegendHeightF	NhlTFloat		RCSG
	PmLegendHeightF	0.175	
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 an **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 3 **Title** resources, as follows:

- tiMainFontHeightF
  PlotManager sets the default value of *tiMainFontHeightF* dynamically based upon the ratio of the actual plot viewport width to a reference viewport width of 0.5 and a reference font height of 0.02.
- tiXAxisFontHeightF
  PlotManager sets the default value of *tiXAxisFontHeightF* dynamically based upon the ratio of the actual plot viewport width to a reference viewport width of 0.5 and a reference font height of 0.02.
- tiYAxisFontHeightF
  PlotManager sets the default value of *tiYAxisFontHeightF* dynamically based upon the ratio of the actual plot viewport height to a reference viewport height of 0.5 and a reference font height of 0.02.

LabelBar resources

If *pmLabelBarDisplayMode* has the value `NoCreate` when an **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 this 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 an **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**, 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 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 discussed below involving annotation plots) that annotations do not overlap each other. And if you decide to change the viewport side where an 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 accomodate.

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 towards increasing NDC values, while the direction of the orthogonal axis is outwards 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. Annotation 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 an 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 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 (towards 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 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 9 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 *towards* rather than away from the viewport center. The parallel axis is still directed towards 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 know as *data-tracking annotations*. Their location is set 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 1987,1988,1989,1991,1993,1994,1995
University Corporation for Atmospheric Research
All Rights Reserved

$Revision: 1.12 $ $Date: 1995/07/24 23:22:12 $

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 is 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 elements. 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 NhlAnnotationDisplayMode 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**
- **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 NhlAnnotationDisplayMode 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**
- **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 NhlAnnotationDisplayMode 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**
- **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 of the setting of certain resources belonging to the **LabelBar**.

Default: 0.2

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 of the setting of certain resources belonging to the **LabelBar**.

Default: 0.5

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 NhlPosition 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, which 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 parallelling 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 parallelling 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 parallelling 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 parallelling 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 towards 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 NhlAnnotationDisplayMode 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**
- **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 of the setting of certain resources belonging to the **Legend**.

Default: 0.45

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 of the setting of certain resources belonging to the **Legend**.

Default: 0.175

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 NhlPosition 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, which 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 parallelling 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 parallelling 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 parallelling 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 parallelling 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 towards 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

$Revision: 1.9 $ $Date: 1995/07/24 23:22:13 $

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	
sfSubsetByIndex	NhlTBoolean		RCSG
	SfSubsetByIndex	False	
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>	
sfXCStride	NhlTInteger		RCSG
	SfXCStride	1	

sfYCStride	NhlTInteger	RCSG
SfYCStride	1	

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** object encapsulates the functionality required to pass a two-dimensional scalar field data array to a receiving plot object. Currently the only object in the HLU library that can act as a receiver of scalar field data is the **ContourPlot** object. The **ScalarField** object requires one resource to be set: the two-dimensional array resource, *sfDataArray*, containing 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 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 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.

ScalarField can pre-process the array in a number of ways before handing it off to the receiving object. Instead of sending the whole array you can pass any contiguous rectangular subset of the array, specified either by array index or by location in the data coordinate space. 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. In addition, you can 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.

Support functions

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

Status

1: Currently, once you have set either the *sfXArray* or the *sfYArray*, there is no way to return the **ScalarField** object to a state in which the coordinates are *not* treated as irregularly gridded. The only work-around is to create a new **ScalarField** object.

See also

- NhlCreate function
- NhlSetValues function
- NhlGetValues function
- NhlAddData function
- ContourPlot object
- DataItem object
- Base object

Copyright

Copyright 1987,1988,1989,1991,1993,1994,1995
University Corporation for Atmospheric Research
All Rights Reserved

$Revision: 1.14 $ $Date: 1995/07/24 18:21:53 $

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 without error. The fastest varying dimension (second dimension in C, 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** receivers (currently just the **ContourPlot** object), it accepts data

arrays containing data of any of the following types:

- NhlTByte
- NhlTCharacter
- NhlTShort
- NhlTInteger
- NhlTLong
- NhlTFloat
- NhlTDouble

You may set *sfDataArray* arrays of type NhlTFloat using **NhlRLSetMDFloatArray** and of type NhlTInteger using **NhlRLSetMDIntegerArray**. The other types must be set using **NhlRLSetMDArray**.

Default: NULL

sfXArray

The **ScalarField** object dynamically determines whether the data array is to be treated as a rectangular array irregularly spaced along the X axis by the presence of this array resource. If *sfXArray* is NULL, the data is treated as regularly spaced along the X Axis. If you set *sfXArray* 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 *sfXArray* resource fails either of these tests, the data array is treated as if it were regularly spaced along the X-Axis. Otherwise, **ScalarField** accepts the values of *sfXArray* as sequentially representing the location along the X-Axis of each successive column in the data array. You can specify *sfXArray* using a single-dimensioned array of any of the types allowed by *sfDataArray*. This implies that the first element of the *sfXArray* specifies the location in data coordinates of the first column of the data grid, and the last element specifies the location of the last column.

V4.0 Status Note 1

Default: NULL

sfYArray

The **ScalarField** object dynamically determines whether the data array is to be treated as a rectangular array irregularly spaced along the Y axis by the presence of this array resource. If *sfYArray* is NULL, the data is treated as regularly spaced along the Y Axis. If you set *sfYArray* 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 *sfYArray* resource fails either of these tests, the data array is treated as if it were regularly spaced along the Y-Axis. Otherwise, **ScalarField** accepts the values of *sfYArray* as sequentially representing the location along the Y-Axis of each successive row in the data array. You can specify *sfYArray* using a single-dimensioned array of any of the types allowed by *sfDataArray*. This implies that the first element of the *sfYArray* specifies the location in data coordinates of the first row of the data grid, and the last element specifies the location of the last row.

V4.0 Status Note 1

Default: NULL

sfSubsetByIndex

The **ScalarField** object allows you to pass only a rectangular subset of the data array to the plot object receiver. If the boolean resource *sfSubsetByIndex* is set True, then you specify the subset based on the data array's indexes. Specify the indexes using the resources *sfXCStartIndex*, *sfXCEndIndex*, *sfYCStartIndex*, and *sfYCEndIndex*. If *sfSubsetByIndex* is False, subsets are specified in terms of the data extents. In this case you would use the resources *sfXCStartSubsetV*, *sfXCEndSubsetV*, *sfYCStartSubsetV*, and *sfYCEndSubsetV* to specify a subset of the data.

Default: False

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 may save 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 a subset of an array irregular in either dimension is specified, the subsetted portion of the array must be copied.

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 occurences 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 *mpDataArray* 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 *mpDataArray* 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 *mpDataArray* resource.

Default: <Dynamic>

sfXCStartV

If the array resource *sfXArray* is set and found to be valid, **ScalarField** ignores the *sfXCStartV* resource, since it determines the starting data location based on the value of the first element of *sfXArray*. Otherwise, *sfXCStartV* specifies the location along the X Axis of the first column of the 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 *sfXCStartV* is unspecified, **ScalarField** sets the starting data location using the value of *sfXCSubsetStartV*, if set, in place of *sfXCStartV*, or to the value 0.0 otherwise. You may set *sfXCStartV* using any of the data types **ScalarField** accepts for the *mpDataArray* resource.

Default: <Dynamic>

sfXCEndV

If the array resource *sfXArray* is set and found to be valid, **ScalarField** ignores the *sfXCEndV* resource, since it determines the ending data location based on the value of the last element of *sfXArray*. Otherwise, *sfXCEndV* specifies the location along the X Axis of the last column of the 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 *sfXCEndV* is unspecified, **ScalarField** sets the ending data location using the value of *sfXCSubsetEndV*, if set, in place of *sfXCEndV*, or to the size of the X-Axis data array dimension minus one otherwise. You may set *sfXCEndV* using any of the data types **ScalarField** accepts for the *mpDataArray* resource.

Default: <Dynamic>

sfYCStartV

If the array resource *sfYArray* is set and found to be valid, **ScalarField** ignores the *sfYCStartV* resource, since it determines the starting data location based on the value of the first element of *sfYArray*. Otherwise, *sfYCStartV* specifies the location along the Y Axis of the first row of the 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 *sfXCStartV* is unspecified, **ScalarField** sets the starting data location using the value of *sfYCSubsetStartV*, if set, in place of *sfYCStartV*, or to the value 0.0 otherwise. You may set *sfYCStartV* using any of the data types **ScalarField** accepts for the *mpDataArray* resource.

Default: <Dynamic>

sfYCEndV

If the array resource *sfYArray* is set and found to be valid, **ScalarField** ignores the *sfYCEndV* resource, since it determines the ending data location based on the value of the last element of *sfYArray*. Otherwise, *sfYCEndV* specifies the location along the Y

Axis of the last row of the 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 *sfYCEndV* is unspecified, **ScalarField** sets the ending data location using the value of *sfYCSubsetEndV*, if set, in place of *sfYCEndV*, or to the size of the Y-Axis data array dimension minus one otherwise. You may set *sfYCEndV* using any of the data types **ScalarField** accepts for the *mpDataArray* resource.

Default: <Dynamic>

sfXCStartSubsetV

If the boolean resource *sfSubsetByIndex* is False, *sfXCStartSubsetV*, when set, specifies the starting location along the X-Axis of a rectagular sub-array of the data array. *sfXCStartSubsetV* applies both to regular and irregular coordinate grids. If the direction of the data grid implied by the magnitude of *sfXCStartSubsetV* relative to *sfXCEndSubsetV* is opposite the direction established by *sfXCStartV* and *sfXCEndV* or the first and last elements of *sfXArray*, **ScalarField** issues a warning and proceeds as if neither *sfXCStartSubsetV* nor *sfXCEndSubsetV* had been set. Otherwise, if the value of *sfXCStartSubsetV* is outside the range of the data coordinates, it is quietly ignored.

Since the data array grid consists of data located at discrete points, the location specified by *sfXCStartSubsetV* will likely fall in between two data locations. 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 plot object, may represent a slightly larger area of the data coordinate space than requested. Note that if the grid is irregularly spaced along either the X or the Y Axes, **ScalarField** must copy the subsetted portion of the data array, regardless of the setting of the *sfCopyData* resource. You may set *sfXCSubsetStartV* using any of the data types **ScalarField** accepts for the *mpDataArray* resource.

Default: <Dynamic>

sfXCEndSubsetV

If the boolean resource *sfSubsetByIndex* is False, *sfXCEndSubsetV*, when set, specifies the ending location along the X-Axis of a rectagular sub-array of the data array. *sfXCEndSubsetV* applies both to regular and irregular coordinate grids. If the direction of the data grid implied by the magnitude of *sfXCEndSubsetV* relative to *sfXCStartSubsetV* is opposite the direction established by *sfXCStartV* and *sfXCEndV* or the first and last elements of *sfXArray*, **ScalarField** issues a warning and proceeds as if neither *sfXCStartSubsetV* nor *sfXCEndSubsetV* had been set. Otherwise, if the value of *sfXCEndSubsetV* is outside the range of the data coordinates, it is quietly ignored.

Since the data array grid consists of data located at discrete points, the location specified by *sfXCEndSubsetV* will likely fall in between two data locations. 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 plot object, may represent a slightly larger area of the data coordinate space than requested. Note that if the grid is irregularly spaced along either the X or the Y Axes, **ScalarField** must copy the subsetted portion of the data array, regardless of the setting of the *sfCopyData* resource. You may set *sfXCEndSubsetV* using any of the

data types **ScalarField** accepts for the *mpDataArray* resource.

Default: <Dynamic>

sfYCStartSubsetV

If the boolean resource *sfSubsetByIndex* is False, *sfYCStartSubsetV*, when set, specifies the starting location along the Y-Axis of a rectagular sub-array of the data array. *sfYCStartSubsetV* applies both to regular and irregular coordinate grids. If the direction of the data grid implied by the magnitude of *sfYCStartSubsetV* relative to *sfYCEndSubsetV* is opposite the direction established by *sfYCStartV* and *sfYCEndV* or the first and last elements of *sfYArray*, **ScalarField** issues a warning and proceeds as if neither *sfYCStartSubsetV* nor *sfYCEndSubsetV* had been set. Otherwise, if the value of *sfYCStartSubsetV* is outside the range of the data coordinates, it is quietly ignored.

Since the data array grid consists of data located at discrete points, the location specified by *sfYCStartSubsetV* will likely fall in between two data locations. 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 plot object, may represent a slightly larger area of the data coordinate space than requested. Note that if the grid is irregularly spaced along either the X or the Y Axes, **ScalarField** must copy the subsetted portion of the data array, regardless of the setting of the *sfCopyData* resource. You may set *sfYCStartSubsetV* using any of the data types **ScalarField** accepts for the *mpDataArray* resource.

Default: <Dynamic>

sfYCEndSubsetV

If the boolean resource *sfSubsetByIndex* is False, *sfYCEndSubsetV*, when set, specifies the ending location along the Y-Axis of a rectagular sub-array of the data array. *sfYCEndSubsetV* applies both to regular and irregular coordinate grids. If the direction of the data grid implied by the magnitude of *sfYCEndSubsetV* relative to *sfYCStartSubsetV* is opposite the direction established by *sfYCStartV* and *sfYCEndV* or the first and last elements of *sfYArray*, **ScalarField** issues a warning and proceeds as if neither *sfYCStartSubsetV* nor *sfYCEndSubsetV* had been set. Otherwise, if the value of *sfYCEndSubsetV* is outside the range of the data coordinates, it is quietly ignored.

Since the data array grid consists of data located at discrete points, the location specified by *sfYCEndSubsetV* will likely fall in between two data locations. 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 plot object, may represent a slightly larger area of the data coordinate space than requested. Note that if the grid is irregularly spaced along either the X or the Y Axes, **ScalarField** must copy the subsetted portion of the data array, regardless of the setting of the *sfCopyData* resource. You may set *sfYCEndSubsetV* using any of the data types **ScalarField** accepts for the *mpDataArray* resource.

Default: <Dynamic>

sfXCStartIndex

If the boolean resource *sfSubsetByIndex* is True, *sfXCStartIndex*, when set, specifies the smaller index along the X-Axis dimension that bounds a rectagular sub-array of the

data array. *sfXCStartIndex* applies both to regular and irregular coordinate grids. Note that if the grid is irregularly spaced along either the X or the Y Axes, **ScalarField** must copy the subsetted portion of the data array, regardless of the setting of the *sfCopyData* resource. The specification of *sfStartIndex* follows C-language conventions: the first index is 0 and the last is the size of the dimension minus 1.

V4.0 NOTE: Range checking for this resource is currently inadaequate.

Default: <Dynamic>

sfXCEndIndex

If the boolean resource *sfSubsetByIndex* is True, *sfXCEndIndex*, when set, specifies the larger index along the X-Axis dimension that bounds a rectagular sub-array of the data array. *sfXCEndIndex* applies both to regular and irregular coordinate grids. This is true regardless of the direction of the data coordinate X-Axis. Note that if the grid is irregularly spaced along either the X or the Y Axes, **ScalarField** must copy the subsetted portion of the data array, regardless of the setting of the *sfCopyData* resource. The specification of *sfEndIndex* follows C-language conventions: the first index is 0 and the last is the size of the dimension minus 1.

V4.0 NOTE: Range checking for this resource is currently inadaequate.

Default: <Dynamic>

sfYCStartIndex

If the boolean resource *sfSubsetByIndex* is True, *sfYCStartIndex*, when set, specifies the smaller index along the Y-Axis dimension that bounds a rectagular sub-array of the data array. *sfYCStartIndex* applies both to regular and irregular coordinate grids. This is true regardless of the direction of the data coordinate Y-Axis. Note that if the grid is irregularly spaced along either the X or the Y Axes, **ScalarField** must copy the subsetted portion of the data array, regardless of the setting of the *sfCopyData* resource. The specification of *sfEndIndex* follows C-language conventions: the first index is 0 and the last is the size of the dimension minus 1.

V4.0 NOTE: Range checking for this resource is currently inadaequate.

Default: <Dynamic>

sfYCEndIndex

If the boolean resource *sfSubsetByIndex* is True, *sfYCEndIndex*, when set, specifies the larger index along the Y-Axis dimension that bounds a rectagular sub-array of the data array. *sfYCEndIndex* applies both to regular and irregular coordinate grids. This is true regardless of the direction of the data coordinate Y-Axis. Note that if the grid is irregularly spaced along either the Y or the Y Axes, **ScalarField** must copy the subsetted portion of the data array, regardless of the setting of the *sfCopyData* resource. The specification of *sfEndIndex* follows C-language conventions: the first index is 0 and the last is the size of the dimension minus 1.

V4.0 NOTE: Range checking for this resource is currently inadaequate. ...description to be added...

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.

V4.0 NOTE: Range checking for this resource is currently inadaequate.

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 in effect. The resulting array is passed to the receiving plot object.

V4.0 NOTE: Range checking for this resource is currently inadaequate.

Default: 1

$Revision: 1.8 $ $Date: 1995/07/23 18:38:57 $

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
TxAngleF		0.0	
txJust	NhlTJustification		RCSG
TxJust		NhlCENTERCENTER	
txDirection	NhlTTextDirection		RCSG
TxDirection		NhlACROSS	
Text Appearance Resources			
NAME	TYPE		ACCESS
CLASS		DEFAULT	
txString	NhlTString		RCSG
TxString		<dynamic>	
txFont	NhlTFont		RCSG
Font		pwritx	
txFontQuality	NhlTFontQuality		RCSG
TxFontQuality		"high"	
txFontColor	NhlTColorIndex		RCSG
TxFontColor		NhlFOREGROUND	

txFontHeightF	NhlTFloat	RCSG
TxFontHeightF	.05	
txFontAspectF	NhlTFloat	RCSG
TxFontAspectF	1.3125	
txFontThicknessF	NhlTFloat	RCSG
TxFontThicknessF	1.0	
txConstantSpacingF	NhlTFloat	RCSG
TxConstantSpacingF	0.0	
txFuncCode	NhlTCharacter	RCSG
TxFuncCode	:	

Text Bounding Box Resources		
NAME	TYPE	ACCESS
CLASS	DEFAULT	
txPerimOn	NhlTBoolean	RCSG
TxPerimOn	False	
txPerimColor	NhlTColorIndex	RCSG
TxPerimColor	NhlFOREGROUND	
txPerimThicknessF	NhlTFloat	RCSG
TxPerimThicknessF	1.0	
txPerimDashPattern	NhlTDashIndex	RCSG
TxPerimDashPattern	0	
txPerimDashLengthF	NhlTFloat	RCSG
TxPerimDashLengthF	0.15	
txPerimSpaceF	NhlTFloat	RCSG
TxPerimSpaceF	0.5	
txBackgroundFillColor	NhlTColorIndex	RCSG
TxBackgroundFillColor	NhlTRANSPARENT	

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 upon 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* to 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 1987,1988,1989,1991,1993,1994,1995
University Corporation for Atmospheric Research
All Rights Reserved

$Revision: 1.18 $ $Date: 1995/07/24 18:21:54 $

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 counter-clockwise and 0 degrees is horizontal on the screen.

Default: 0.0

txFont

When the *txFontQuality* resource has the value NhlHIGH, this resource of type NhlFont specifies the font used for the **TextItem**. The fonts may be catagorized 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 NhlMEDIUM or NhlLOW, 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 NhlJustification sets the justification point of the **TextItem**.

Default: NhlCENTERCENTER

txFontQuality

This resource of type NhlFontQuality determines the quality of the font used to draw the **TextItem**. There are three choices:

NhlHIGH

Draw characters using any of the stoked or filled fonts.

NhlMEDIUM

Draw characters using a 94 character stroked font that is simpler than the NhlHIGH quality fonts, resulting in somewhat smaller metafiles. The *txFont* resource is ignored.

NhlLOW

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 NhlMEDIUM text. The *txFont* resource is ignored.

Default: NhlHIGH

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 - 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 NhlTextDirection specifies the direction of the text drawn by the **TextItem**. There are three choices of which two are actually the same:

NhlDOWN
: Each character is placed below the previous character in the text string.

NhlUP
: Each character is placed below the previous character in the text string.

NhlACROSS
: 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: NhlACROSS

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: NhlFOREGROUND

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 segement 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 NhlTRANSPARENT (-1).

Default: NhlTRANSPARENT

$Revision: 1.10 $ $Date: 1995/07/23 18:39:00 $

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		
Resources for drawing the grid boundary		
NAME	**TYPE**	**ACCESS**
CLASS	**DEFAULT**	
tmXBBorderOn	NhlTBoolean	RCSG
TmXBBorderOn	True	
tmXTBorderOn	NhlTBoolean	RCSG
TmXTBorderOn	True	
tmYLBorderOn	NhlTBoolean	RCSG
TmYLBorderOn	True	
tmYRBorderOn	NhlTBoolean	RCSG
TmYRBorderOn	True	

tmBorderThicknessF	NhlTFloat	RCSG
TmBorderThicknessF	2.0	
tmBorderLineColor	NhlTColorIndex	RCSG
TmBorderLineColor	"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
TmMajorGridLineColors	"Foreground"	
tmYMajorGridLineColor	NhlTColorIndex	RCSG
TmMajorGridLineColors	"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
TmMinorGridLineColors	"Foreground"	
tmYMinorGridLineColor	NhlTColorIndex	RCSG
TmMinorGridLineColors	"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), or 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	NhlTTickMarkMode	RCSG
TmXTMode	"Automatic"	
tmYLMode	NhlTTickMarkMode	RCSG
TmYLMode	"Automatic"	
tmYRMode	NhlTTickMarkMode	RCSG
TmYRMode	"Automatic"	
tmXBMaxTicks	NhlTInteger	RCSG
TmXBMaxTicks	7	
tmXTMaxTicks	NhlTInteger	RCSG
TmXTMaxTicks	7	
tmYLMaxTicks	NhlTInteger	RCSG
TmYLMaxTicks	7	
tmYRMaxTicks	NhlTInteger	RCSG
TmYRMaxTicks	7	
tmXBAutoPrecision	NhlTBoolean	RCSG
TmAutoPrecision	True	
tmXTAutoPrecision	NhlTBoolean	RCSG
TmAutoPrecision	True	
tmYLAutoPrecision	NhlTBoolean	RCSG
TmAutoPrecision	True	
tmYRAutoPrecision	NhlTBoolean	RCSG
TmAutoPrecision	True	
tmXBPrecision	NhlTInteger	RCSG
TmPrecisions	4	
tmXTPrecision	NhlTInteger	RCSG
TmPrecisions	4	

tmYLPrecision	NhlTInteger	RCSG
TmPrecisions	4	
tmYRPrecision	NhlTInteger	RCSG
TmPrecisions	4	
tmXBFormat	NhlTString	RCSG
TmFormat	"0@*+^sg"	
tmXTFormat	NhlTString	RCSG
TmFormat	"0@*+^sg"	
tmYLFormat	NhlTString	RCSG
TmFormat	"0@*+^sg"	
tmYRFormat	NhlTString	RCSG
TmFormat	"0@*+^sg"	
tmXBStyle	NhlTTickMarkStyle	RCSG
TmXBStyle	"Linear"	
tmXTStyle	NhlTTickMarkStyle	RCSG
TmXTStyle	"Linear"	
tmYLStyle	NhlTTickMarkStyle	RCSG
TmYLStyle	"Linear"	
tmYRStyle	NhlTTickMarkStyle	RCSG
TmYRStyle	"Linear"	
The following Data resources relate to an axis scale mode of "Automatic", "Manual", or "Explicit".		
tmXBDataLeftF	NhlTFloat	RCSG
TmXBDataLeftF	0.0	
tmXBDataRightF	NhlTFloat	RCSG
TmXBDataRightF	1.0	
tmXTDataLeftF	NhlTFloat	RCSG
TmXTDataLeftF	0.0	
tmXTDataRightF	NhlTFloat	RCSG
TmXTDataRightF	1.0	
tmYLDataBottomF	NhlTFloat	RCSG
TmYLDataBottomF	0.0	
tmYLDataTopF	NhlTFloat	RCSG
TmYLDataTopF	1.0	
tmYRDataBottomF	NhlTFloat	RCSG
TmYRDataBottomF	0.0	
tmYRDataTopF	NhlTFloat	RCSG
TmYRDataTopF	1.0	
The following 3 resources per scaled axis are required for a scale mode of either "Manual" or "Explicit".		
tmXBTickStartF	NhlTFloat	RCSG

TmXBTickStartF	0.0	
tmXBTickEndF	NhlTFloat	RCSG
TmXBTickEndF	0.0	
tmXBTickSpacingF	NhlTFloat	RCSG
TmXBTickSpacingF	0.0	
tmXTTickStartF	NhlTFloat	RCSG
TmXTTickStartF	0.0	
tmXTTickEndF	NhlTFloat	RCSG
TmXTTickEndF	0.0	
tmXTTickSpacingF	NhlTFloat	RCSG
TmXTTickSpacingF	0.0	
tmYLTickStartF	NhlTFloat	RCSG
TmYLTickStartF	0.0	
tmYLTickEndF	NhlTFloat	RCSG
TmYLTickEndF	0.0	
tmYLTickSpacingF	NhlTFloat	RCSG
TmYLTickSpacingF	0.0	
tmYRTickStartF	NhlTFloat	RCSG
TmYRTickStartF	0.0	
tmYRTickEndF	NhlTFloat	RCSG
TmYRTickEndF	0.0	
tmYRTickSpacingF	NhlTFloat	RCSG
TmYRTickSpacingF	0.0	
tmXBIrrTensionF	NhlTFloat	RCSG
TmXBIrrTensionF	2.0	
tmXTIrrTensionF	NhlTFloat	RCSG
TmXTIrrTensionF	2.0	
tmYLIrrTensionF	NhlTFloat	RCSG
TmYLIrrTensionF	2.0	
tmYRIrrTensionF	NhlTFloat	RCSG
TmYRIrrTensionF	2.0	
tmXBIrregularPoints	NhlTFloatGenArray	RCSG
TmXBIrregularPoints	NULL	
tmXTIrregularPoints	NhlTFloatGenArray	RCSG
TmXTIrregularPoints	NULL	
tmYLIrregularPoints	NhlTFloatGenArray	RCSG
TmYLIrregularPoints	NULL	
tmYRIrregularPoints	NhlTFloatGenArray	RCSG
TmYRIrregularPoints	NULL	
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
tmXTValues TmXTValues	NhlTFloatGenArray NULL	RCSG
tmXTLabels TmXTLabels	NhlTStringGenArray NULL	RCSG
tmYLValues TmYLValues	NhlTFloatGenArray NULL	RCSG
tmYLLabels TmYLLabels	NhlTStringGenArray NULL	RCSG
tmYRValues TmYRValues	NhlTFloatGenArray NULL	RCSG
tmYRLabels TmYRLabels	NhlTStringGenArray 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
tmXBLabelFont Font	NhlTFont "pwritx"	RCSG

tmXTLabelFont Font	NhlTFont "pwritx"	RCSG
tmYLLabelFont Font	NhlTFont "pwritx"	RCSG
tmYRLabelFont Font	NhlTFont "pwritx"	RCSG
tmXBLabelFontHeightF TmLabelFontHeightsF	NhlTFloat 0.02	RCSG
tmXTLabelFontHeightF TmLabelFontHeightsF	NhlTFloat 0.02	RCSG
tmYLLabelFontHeightF TmLabelFontHeightsF	NhlTFloat 0.02	RCSG
tmYRLabelFontHeightF TmLabelFontHeightsF	NhlTFloat 0.02	RCSG
tmXBLabelFontColor TmLabelFontColors	NhlTColorIndex "Foreground"	RCSG
tmXTLabelFontColor TmLabelFontColors	NhlTColorIndex "Foreground"	RCSG
tmYLLabelFontColor TmLabelFontColors	NhlTColorIndex "Foreground"	RCSG
tmYRLabelFontColor TmLabelFontColors	NhlTColorIndex "Foreground"	RCSG
tmXBLabelFontAspectF TmLabelFontAspectsF	NhlTFloat 1.3125	RCSG
tmXTLabelFontAspectF TmLabelFontAspectsF	NhlTFloat 1.3125	RCSG
tmYLLabelFontAspectF TmLabelFontAspectsF	NhlTFloat 1.3125	RCSG
tmYRLabelFontAspectF TmLabelFontAspectsF	NhlTFloat 1.3125	RCSG
tmXBLabelJust TmXLabelJusts	NhlTJustification "TopCenter"	RCSG
tmXTLabelJust TmXLabelJusts	NhlTJustification "BottomCenter"	RCSG
tmYLLabelJust TmYLabelJusts	NhlTJustification "CenterRight"	RCSG
tmYRLabelJust TmYLabelJusts	NhlTJustification "TopLeft"	RCSG
tmXBLabelAngleF TmXLabelAnglesF	NhlTFloat 0.0	RCSG
tmXTLabelAngleF	NhlTFloat	RCSG

TmXLabelAnglesF	0.0	
tmYLLabelAngleF	NhlTFloat	RCSG
TmYLabelAnglesF	0.0	
tmYRLabelAngleF	NhlTFloat	RCSG
TmYLabelAnglesF	0.0	
tmXBLabelDirection	NhlTTextDirection	RCSG
TmXLabelDirections	"Across"	
tmXTLabelDirection	NhlTTextDirection	RCSG
TmXLabelDirections	"Across"	
tmYLLabelDirection	NhlTTextDirection	RCSG
TmYLabelDirections	"Across"	
tmYRLabelDirection	NhlTTextDirection	RCSG
TmYLabelDirections	"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	3	
tmXTMinorPerMajor	NhlTInteger	RCSG
TmXMinorPerMajor	3	
tmYLMinorPerMajor	NhlTInteger	RCSG
TmYMinorPerMajor	3	
tmYRMinorPerMajor	NhlTInteger	RCSG
TmYMinorPerMajor	3	

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
TmMajorLineColors	"Foreground"	
tmXTMajorLineColor	NhlTColorIndex	RCSG
TmMajorLineColors	"Foreground"	
tmYLMajorLineColor	NhlTColorIndex	RCSG
TmMajorLineColors	"Foreground"	
tmYRMajorLineColor	NhlTColorIndex	RCSG
TmMajorLineColors	"Foreground"	
tmXBMajorLengthF	NhlTFloat	RCSG
TmMajorLengthsF	.02	
tmXTMajorLengthF	NhlTFloat	RCSG
TmMajorLengthsF	.02	
tmYLMajorLengthF	NhlTFloat	RCSG
TmMajorLengthsF	.02	
tmYRMajorLengthF	NhlTFloat	RCSG
TmMajorLengthsF	.02	
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
TmMinorLineColors	"Foreground"	
tmXTMinorLineColor	NhlTColorIndex	RCSG
TmMinorLineColors	"Foreground"	
tmYLMinorLineColor	NhlTColorIndex	RCSG
TmMinorLineColors	"Foreground"	
tmYRMinorLineColor	NhlTColorIndex	RCSG
TmMinorLineColors	"Foreground"	
tmXBMinorLengthF	NhlTFloat	RCSG
TmMinorLengthsF	.01	
tmXTMinorLengthF	NhlTFloat	RCSG
TmMinorLengthsF	.01	
tmYLMinorLengthF	NhlTFloat	RCSG
TmMinorLengthsF	.01	
tmYRMinorLengthF	NhlTFloat	RCSG
TmMinorLengthsF	.01	
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 superclasses:

- **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 tickmarks, 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 *tmYR* and *tmY* 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 to 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. Currently minor tick marks must be turned off to use the `Explicit` mode but in the future a resource will be available for supplying data locations for the minor tick marks. If you do not set the values array **TickMark** issues a warning message and reverts to `Automatic` mode. If you do not set the labels array, **TickMark** also issues a warning, but in this case it simply does not draw any labels along the affected 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 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 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: "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 1987,1988,1989,1991,1993,1994,1995
University Corporation for Atmospheric Research
All Rights Reserved

$Revision: 1.21 $ $Date: 1995/07/24 18:21:55 $

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 (signalled 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 devices 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 tickmarks. The resources affected are:

tmXTStyle, *tmXTMode*, *tmXTMinorPerMajor*, *tmXTNoMinor*, *tmXTDataLeftF*, *tmXTDataRightF*, *tmXTTickStartF*, *tmXTTickEndF*, *tmXTMaxTicks*, *tmXTTickSpacing*, *tmXTValues*, *tmXTLabels*, *tmXTMajorThicknessF*, *tmXTMajorLineColor*, *tmXTMajorLengthF*, *tmXTMajorOutwardLengthF*, *tmXTMinorThicknessF*, *tmXTMinorLineColor*, *tmXTMinorLengthF*, *tmXTMinorOutwardLength*, *tmXTLabelFont*, *tmXTLabelFontHeightF*, *tmXTLabelFontColor*, *tmXTLabelFontAspectF*, *tmXTLabelAngleF*, *tmXTLabelDirection*, *tmXTLabelDeltaF*, *tmXTIrregularPoints*, *tmXTIrrTensionF*, *tmXTLabelStride*, *tmXTAutoPrecision*, *tmXTPrecision* and *tmXTFormat*.

All other Top **TickMark** resources are unaffected.

Default: True

tmXMajorGrid

Setting to True turns on the X Axis major grid; False turns it off. Grid lines will extend the bot tom 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 to 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 w ill 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 devices 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 minor the grid. Value is a multiple of the devices 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 to True turns on bottom tick marks; False turns them off.

Default: True

tmXTOn

Setting to True turns on top tick marks; False turns them off.

Default: True

tmXBLabelsOn

Setting to True turns on tick mark labels if *tmXBOn* is set; False turns them off.

Default: True

tmXTLabelsOn

Setting to True turns on tick mark labels if *tmXTOn* is set; False turns them off.

Default: False

tmXBBorderOn

Setting to True turns bottom border on; False turns it off. Does not turn off bottom ticks.

Default: True

tmXTBorderOn

Setting to 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 *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.

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 *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.

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 an 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 *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**

V4.0 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 an 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 *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**

V4.0 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 (signalled by the '.' character) is set explicitly in the *tmXBFormat* string (according to the rules of 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 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 (signalled by the '.' character) is set explicitly in the *tmXTFormat* string (according to the rules of 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 p*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 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. If `Log` style is used only 1, 4, and 8 are valid minor per major numbers. If `Explicit` mode is used this resource is ignored.

Default: 3

tmXTMinorPerMajor

Sets the number of minor tick marks per major tick mark for the top x axis. If `Log` style is used only 1, 4, and 8 are valid minor per major numbers. If `Explicit` mode is used this resource is ignored.

Default: 3

tmXBMinorOn

Setting this resource to True turns on minor tick marks; False turns them off.

Default: True

tmXTMinorOn

Setting this resource to True turns on minor tick marks; False turns them off.

Default: True

tmXBLabelStride

Used to skip tick mark labels. The value represent how many tickmarks to skip be tween tick mark labels when drawing the tick mark labels. Will cause the tick mark obj ect to draw first then skip.

Default: 0

tmXTLabelStride

Used to skip tick mark labels. The value represent how many tickmarks to skip be tween tick mark labels when drawing the tick mark labels. Will cause the tick mark obj ect to draw first then skip.

Default: 0

tmXBDataLeftF

Sets the data value that maps to the left hand 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 hand 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 tickmarks. 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.

V4.0 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 an element is NULL (no string) then no tick label is draw under the corresponding tick mark.

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 values scales proportionally to changes in height.

Default: .02

tmXBMajorOutwardLengthF

Length of the major X Axis bottom tick marks that will be drawn on the outside o f 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 mult iple of the current devices minimum linewidth.

Default: 1.0

tmXBMinorLineColor

Sets color index from bottom minor tick marks.

Default: 1

tmXBMinorLengthF

Sets the length in NDC coordinates for x bottom tick marks. This value scales proportionally to changes in height.

Default: .01

tmXBMinorOutwardLengthF

Length of minor X Axis bottom tick marks that will be draw on the outside of the tick mark border. Value is in NDC coordinates. This values scales proportionally to c hanges in height.

Default: 0.0

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 a bottom X Axis labels.

Default: 1.3125

tmXBLabelJust

Sets the justification point for all bottom X Axis labels

Default: `TopCenter`

tmXBLabelAngleF

Sets the counter clockwise 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 v alue multiplied by the *tmXBFontHeightF* represents the amount, in NDC coordinates, tha t the bottom tick marks will be displaced from their default location in the y dir ection. Positive value moves labels in positive direction in NDC coordinate and negative in the negative direction.

Default: 0.0

tmXTDataLeftF

Sets the data value that maps to the left hand 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 hand 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 dra wn 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 tickmarks. 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. 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.

Default: NULL

tmXTLabels

Used only for `Explicit` mode tick marks. This resource is an array of strings tha t correspond to the values in *tmXTValues*. Each element of this resource is placed under the corresponding tick location in *tmXTValues*. If an element is NULL (no string) then no tick label is drawn under the corresponding tick mark.

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 values scales proportionally to changes in height.

Default: .02

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 values 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 devices minimum linewidth.

Default: 1.0

tmXTMinorLineColor

Sets color index from top minor tick marks.

Default: 1

tmXTMinorLengthF

Sets the length in NDC coordinates for x top tick marks. This values scales proportionally to changes in height.

Default: .01

tmXTMinorOutwardLengthF

Length of minor X Axis top tick marks that will be draw on the outside of the tick mark border. Value is in NDC coordinates. This values scales proportionally to changes in height.

Default: 0.0

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 a top X Axis labels.

Default: 1.3125

tmXTLabelJust

Sets the justification point for all top X Axis labels.

Default: `BottomCenter`

tmXTLabelAngleF

Sets the counter clockwise 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, tha t the top tick marks will be displaced from their default location in the y direct ion. Positive value moves labels in positive direction in NDC coordinate and negative in the negative direction.

Default: 0.0

tmYUseLeft

Causes the right tick marks to use the resources for the left tickmarks. The resources affected are

tmYRStyle, *tmYRMode*, *tmYRMinorPerMajor*, *tmYRNoMinor*, *tmYRDataBottomF*, *tmYRDataTopF*, *tmYRTickStartF*, *tmYRTickEndF*, *tmYRMaxTicks*, *tmYRTickSpacing*, *tmYRValues*, *tmYRLabels*, *tmYRMajorThicknessF*, *tmYRMajorLineColor*, *tmYRMajorLengthF*, *tmYRMajorOutwardLengthF*, *tmYRMinorThicknessF*,

tmYRMinorLineColor, *tmYRMinorLengthF*, *tmYRMinorOutwardLength*, *tmYRLabelFont*, *tmYRLabelFontHeightF*, *tmYRLabelFontColor*, *tmYRLabelFontAspectF*, *tmYRLabelAngleF*, *tmYRLabelDirection*, *tmYRLabelDeltaF*, *tmYRIrregularPoints*, *tmYRIrrTensionF*, *tmYRLabelStride*, *tmYRAutoPrecision*, *tmYRPrecision*, and *tmYRFormat*.

All other Right **TickMark** resources are unaffected.

Default: True

tmYLOn

Setting to True turns on the left tick marks and False turns them off.

Default: True

tmYROn

Setting to True turns on the right tick marks and False turns them off.

Default: True

tmYLLabelsOn

Setting to True turns left labels on if *tmYLOn* is set. False turns them off.

Default: True

tmYRLabelsOn

Setting to True turns right labels on if *tmYROn* is set. False turns them off.

Default: False

tmYLBorderOn

Setting to True turns left border on and False turns it off. Tick marks will still be drawn unless *tmYLOn* is False.

Default: True

tmYRBorderOn

Setting to 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 *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.

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 *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.

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 an 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; large values (greater than ~5.0) imply a tight approximation.

This resource may be intercepted or disabled by:

- **PlotManager**

V4.0 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 an 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; large values (greater than ~5.0) imply a tight approximation.

This resource may be intercepted or disabled by:

- **PlotManager**

V4.0 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 (signalled by the '.' character) is set explicitly in the *tmYLFormat* string (according to the rules of 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 p*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 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 (signalled by the '.' character) is set explicitly in the *tmYRFormat* string (according to the rules of 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 p*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 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 to 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 to 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 marks. For `Log` style plots this value must be either 1,4, or 8.

Default: 3

tmYRMinorPerMajor

Sets the number of minor tick marks per major tick marks. For `Log` style plots this value must be either 1,4, or 8.

Default: 3

tmYLMinorOn

Turns on minor tickmarks when set to True; False turns them off.

Default: True

tmYRMinorOn

Turns on minor tickmarks when set to True; False turns them off.

Default: True

tmYLLabelStride

Used to skip tick mark labels. The value represents how many tickmarks 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 tickmarks 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 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 tickmarks that will be chosen.

Default: 7

tmYLTickSpacingF

Used for `Manual` mode ticks only. Specifies the spacing between major tickmarks. 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.

V4.0 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 NULL is used in place of a string no label will appear at that tick mark location.

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 tickmarks.

Default: 1

tmYLMajorLengthF

Sets the length of the Y Axis left major tick marks in NDC coordinates. This values scales proportionally to changes in width.

Default: .02

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 tickmarks.

Default: 1.0

tmYLMinorLineColor

Sets the color index for the Y Axis minor tickmarks.

Default: 1

tmYLMinorLengthF

Sets minor tick mark length for the Y Axis minor tick marks. This values scales proportionally to changes in width.

Default: .01

tmYLMinorOutwardLengthF

Sets the length of the Y Axis left minor tick marks that will be drawn outside of the viewport. This values scales proportionally to changes in width.

Default: 0.0

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 height/width for each character of the Y Axis left tick mark labels .

Default: 1.3125

tmYLLabelJust
Sets font justification point for Y Axis left tick mark labels.

Default: `CenterRight`

tmYLLabelAngleF
Sets the counter clockwise rotation for all Y Axis tickmark 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. Positive value moves labels in positive direction in NDC coordinate and negative in the negative direction.

Default: 0.0

tmYRDataTopF
The data value that will be mapped to the top right corner of 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 tickmarks that will be

drawn.

Default: 7

tmYRTickSpacingF

Used for `Manual` mode ticks only. Specifies the spacing between major tickmarks. 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. 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.

V4.0 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 NULL is used in place of a string no label will appear at that tick mark location.

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 values scales proportionally to changes in width.

Default: .02

tmYRMajorOutwardLengthF

Sets the length of the Y Axis right major tick marks that will be drawn outside of the

current viewport. This value scale proportionally to changes in width.

Default: 0.0

tmYRMinorThicknessF

Sets the length of the Y Axis minor tickmarks in NDC coordinates. This value scale 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 value scales proportionally to changes in width.

Default: .01

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

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 values 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

tmYRLabelJust

Sets the justification point for the right Y Axis labels

Default: `CenterLeft`

tmYRLabelAngleF

Sets the counter-clockwise 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. Positive value moves labels in positive direction in NDC coordinate and negative in the negative direction.

Default: 0.0

$Revision: 1.13 $ $Date: 1995/07/24 23:22:14 $

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	TYPE	ACCESS	
CLASS	DEFAULT		
tiDeltaF	NhlTFloat	RCSG	
TiDeltaF	1.5		
tiUseMainAttributes	NhlTBoolean	RCSG	
TiUseMainAttributes	False		
tiMainOn	NhlTBoolean	RCSG	
TiMainOn	<Dynamic>		

tiMainString	NhlTString	RCSG
TiMainString	"Main"	
tiMainSide	NhlTPosition	RCSG
TiMainSide	NhlTOP	
tiMainPosition	NhlTPosition	RCSG
TiMainPosition	NhlCENTER	
tiMainOffsetXF	NhlTFloat	RCSG
TiMainOffsetXF	0.0	
tiMainOffsetYF	NhlTFloat	RCSG
TiMainOffsetYF	0.0	
tiMainJust	NhlTJustification	RCSG
TiTitleJust	"CenterCenter"	
tiMainFont	NhlTFont	RCSG
Font	0	
tiMainFontColor	NhlTColorIndex	RCSG
TiTitleFontColors	"Foreground	
tiMainFontHeightF	NhlTFloat	RCSG
TiTitleFontHeightsF	0.025	
tiMainFontAspectF	NhlTFloat	RCSG
TiTitleFontAspectsF	1.3125	
tiMainConstantSpacingF	NhlTFloat	RCSG
TiTitleConstantSpacingsF	0.0	
tiMainFontThicknessF	NhlTFloat	RCSG
TiTitleFontThicknessF	1.0	
tiMainFuncCode	NhlTCharacter	RCSG
TiTitleFuncCodes	:	
tiMainFontQuality	NhlTFontQuality	RCSG
TiTitleFontQualities	NhlHIGH	
tiMainAngleF	NhlTFloat	RCSG
TiTitleAnglesF	0.00	
tiMainDirection	NhlTTextDirection	RCSG
TiMainDirection	NhlACROSS	
tiXAxisOn	NhlTBoolean	RCSG
TiXAxisOn	<Dynamic>	
tiXAxisString	NhlTString	RCSG
TiXAxisString	"XAxis"	
tiXAxisSide	NhlTPosition	RCSG
TiXAxisSide	NhlBOTTOM	
tiXAxisPosition	NhlTPosition	RCSG
TiXAxisPosition	NhlCENTER	

tiXAxisOffsetXF	NhlTFloat	RCSG
TiXAxisOffsetXF	0.0	
tiXAxisOffsetYF	NhlTFloat	RCSG
TiXAxisOffsetYF	0.0	
tiXAxisJust	NhlTColorIndex	RCSG
TiTitleJust	"CenterCenter"	
tiXAxisFont	NhlTFont	RCSG
Font	0	
tiXAxisFontColor	NhlTColorIndex	RCSG
TiTitleFontColors	"Foreground"	
tiXAxisFontHeightF	NhlTFloat	RCSG
TiTitleFontHeightsF	0.025	
tiXAxisFontAspectF	NhlTFloat	RCSG
TiTitleFontAspectsF	1.3125	
tiXAxisConstantSpacingF	NhlTFloat	RCSG
TiTitleConstantSpacingsF	0.0	
tiXAxisFontThicknessF	NhlTFloat	RCSG
TiTitleFontThicknessF	1.0	
tiXAxisFuncCode	NhlTCharacter	RCSG
TiTitleFuncCodes	:	
tiXAxisFontQuality	NhlTFontQuality	RCSG
TiTitleFontQualities	NhlHIGH	
tiXAxisAngleF	NhlTFloat	RCSG
TiTitleAnglesF	0.0	
tiXAxisDirection	NhlTTextDirection	RCSG
TiXAxisDirection	NhlACROSS	
tiYAxisOn	NhlTBoolean	RCSG
TiYAxisOn	<Dynamic>	
tiYAxisString	NhlTString	RCSG
TNhlTStringiYAxisString	"YAxis"	
tiYAxisSide	NhlTPosition	RCSG
TiYAxisSide	NhlLEFT	
tiYAxisPosition	NhlTPosition	RCSG
TiYAxisPosition	NhlCENTER	
tiYAxisOffsetXF	NhlTFloat	RCSG
TiYAxisOffsetXF	0.0	
tiYAxisOffsetYF	NhlTFloat	RCSG
TiYAxisOffsetYF	0.0	
tiYAxisJust	NhlTJustification	RCSG
TiTitleJust	"CenterCenter"	
tiYAxisFont	NhlTFont	RCSG

	Font	0	
tiYAxisFontColor	TiTitleFontColors	NhlTColorIndex "Foreground"	RCSG
tiYAxisFontHeightF	TiTitleFontHeightsF	NhlTFloat 0.025	RCSG
tiYAxisFontAspectF	TiTitleFontAspectsF	NhlTFloat 1.3125	RCSG
tiYAxisConstantSpacingF	TiTitleConstantSpacingsF	NhlTFloat 0.0	RCSG
tiYAxisFontThicknessF	TiTitleFontThicknessF	NhlTFloat 1.0	RCSG
tiYAxisFuncCode	TiTitleFuncCodes	NhlTCharacter :	RCSG
tiYAxisFontQuality	TiTitleFontQualities	NhlTFontQuality NhlHIGH	RCSG
tiYAxisAngleF	TiTitleAnglesF	NhlTFloat 90.0	RCSG
tiYAxisDirection	TiYAxisDirection	NhlTTextDirection NhlACROSS	RCSG

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

The **Title** object class draws titles around the viewport defined by the *vpXF*, *vpYF*, *vpWidthF*, and *vpHeightF*. There are three titles drawn by the **Title** object, the *Main* title, the *X-Axis* title and the *Y-Axis* title. The Main title by default 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.

The names of the resources used to configure the text attributes of the titles are formed in a consistent way from the corresponding **TextItem** resources. The name of the individual title is prepended to the **TextItem** resource name. For example, the **TextItem** resource for setting the string to draw is *txString*. For the three individual titles, the **Title** object provides the

string resources *tiMainString*, *tiXAxisString*, and *tiYAxisString*. All other text attribute resources for the titles are constructed in the same fashion. However, note that the **Title** object has its own resources for positioning the titles relative to its viewport, and therefore does not recognize the **TextItem** positioning resources, *txPosXF* and *txPosYF*. Also the **Title** object does not support perimeter outlines or fill of the area occupied by the text string.

The resources for adjusting the position of the titles are *tiDeltaF*, *tiMainPosition*, *tiMainSide*, *tiMainOffsetXF*, *tiMainOffsetYF*. Similarly named resources are available for the X-Axis and Y-Axis as well. *tiDeltaF* applies to all titles. It is a multiplier that defines how far from the viewport edge titles should be placed. The distance is computed by multiplying *tiDeltaF* by the appropriate font height. *tiMainPosition* can be set to NhlLEFT, NhlCENTER or NhlRIGHT specifying what part of the viewport the Main title will be centered above or below. *tiMainSide* determines what side of the viewport the main title is placed. It is a resource of type NhlPosition, that can be set to either of the values NhlTOP or NhlBOTTOM. Similar resources apply to the X-Axis and Y-Axis. The resources *tiMainOffsetXF* and *tiMainOffsetYF* are available to displace text in NDC coordinates from its default position.

When the X-Axis and the Main title share the same side of the viewport the **Title** object makes sure the two titles will not overlap. The Main title is always placed outside of the X-Axis title when they are on the same side.

Support functions

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

Status

1: The *tiUseMainAttributes* resource currently does not copy the values of *tiMainFontQuality* and *tiMainFontColor* as one might expect it should. It *does* copy the value of *tiMainAngleF* even though the default value for the Y-Axis is different from the default values for the Main and X-Axis titles. This needs attention.

See also

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

Copyright

Copyright 1987,1988,1989,1991,1993,1994,1995
University Corporation for Atmospheric Research
All Rights Reserved

$Revision: 1.12 $ $Date: 1995/07/24 18:21:56 $

Title class resource descriptions

tiDeltaF

Sets the default offset for all of the titles. This value multiplied by the appropriate *ti...FontHeightF* resource value gives the amount, in NDC, that the titles will be offset. Positive values move the titles away from the viewport center. For example, a positive value would cause the Main title to move upwards when *tiMainSide* is set to NhlTOP.

Default: 1.5

tiUseMainAttributes

When this boolean resource is set to True, the **Title** object copies the value of Main title resources that set text attributes to the corresponding X-Axis and Y-Axis title resources. The resources copied include *tiMainFont*, *tiMainFontHeightF*, *tiMainFontAspectF*, *tiMainConstantSpacingF*, *tiMainFontThicknessF*, *tiMainFuncCode*, *tiMainJust*, and *tiMainAngleF*.

V4.0 Status Note 1

Default: False

---

tiMainOn

Setting this boolean resource to 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 NhlPosition specifies the side of the viewport relative to which the justification point of the Main title is located. There are two choices:
NhlTOP

Locate the justification point of the Main title relative to the top edge of the viewport.

NhlBOTTOM

Locate the justification point of the Main title relative to the bottom edge of the viewport.

If you set *tiMainPosition* to NhlLEFT, NhlCENTER, or NhlRIGHT **Title** issues a warning message and reverts to the default value, NhlTOP.

Default: NhlTOP

tiMainPosition

This resource of type NhlPosition specifies the base horizontal location of the justification point of the Main title with respect to the viewport boundaries. There are three choices:

NhlLEFT

Locate the base position of the justification point of the Main title along the line of the left edge of the viewport.

NhlCENTER

Locate the base position of the justification point of the Main title along the line of the horizontal center of the viewport.

NhlRIGHT

Locate the base position of justification point of the Main title along the line of the left 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 NhlBOTTOM or NhlTOP **Title** issues a warning message and reverts to the default value, NhlCENTER.

Default: NhlCENTER

tiMainOffsetXF

Sets an offset in NDC coordinates by which the Main title will be displaced in a direction parallel to X-Axis from its base location. This value does not scale in response to changes to the viewport size.

Default: 0.0

tiMainOffsetYF

Sets an 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 over any offset due to *tiDeltaF*. It does not scale in response to change to the viewport size.

Default: 0.0

tiMainJust

This resource of type NhlJustification sets the Justification point of the Main title.

Default: NhlCENTERCENTER

tiMainFont

This resource of type NhlFont 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' 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 *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 *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 *txFont* specifies a filled font (font indexes 21, 22, 25 - 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: NhlHIGH

tiMainAngleF

Angle of counter-clockwise rotation of the Main title.

Default: 0.00

tiMainDirection

This resource of type NhlTextDirection specifies the direction of the text used for the Main title. There are three choices of which two are actually the same:

NhlDOWN

Each character is placed below the previous character in the text string.

NhlUP

Each character is placed below the previous character in the text string.

NhlACROSS

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: NhlACROSS

---

tiXAxisOn

Setting this boolean resource to 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 explicit 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 NhlPosition specifies the side of the viewport relative to which the justification point of the X-Axis title is located. There are two choices:

NhlTOP

Locate the justification point of the X-Axis title relative to the top edge of the viewport.

NhlBOTTOM

Locate the justification point of the X-Axis title relative to the bottom edge of the viewport.

If you set *tiXAxisPosition* to NhlLEFT, NhlCENTER, or NhlRIGHT **Title** issues a warning message and reverts to the default value, NhlBOTTOM.

Default: NhlBOTTOM

tiXAxisPosition

This resource of type NhlPosition specifies the base horizontal location of the justification point of the X-Axis title with respect to the viewport boundaries. There are three choices:

NhlLEFT

Locate the base position of the justification point of the X-Axis title along the line of the left edge of the viewport.

NhlCENTER

Locate the base position of the justification point of the X-Axis title along the line of the horizontal center of the viewport.

NhlRIGHT

Locate the base position of the justification point of the X-Axis title along the line of the left 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 NhlBOTTOM or NhlTOP **Title** issues a warning message and reverts to the default value, NhlCENTER.

Default: NhlCENTER

tiXAxisOffsetXF

Sets an offset in NDC coordinates by which the X-Axis title will be displaced in a direction parallel to X-Axis from its base location. This value does not scale in response to changes to the viewport size.

Default: 0.0

tiXAxisOffsetYF

Sets an 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 over any offset due to *tiDeltaF*. It does not scale in response to change to the viewport size.

Default: 0.0

tiXAxisJust

This resource of type NhlJustification sets the Justification point of the X-Axis title.

Default: NhlCENTERCENTER

tiXAxisFont

This resource of type NhlFont 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' 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 *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 *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 *txFont* specifies a filled font (font indexes 21, 22, 25 - 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: NhlHIGH

tiXAxisAngleF

Sets the counter-clockwise rotation angle for the X-Axis title.

Default: 0.0

tiXAxisDirection

This resource of type NhlTextDirection specifies the direction of the text used for the X-Axis title. There are three choices of which two are actually the same:

NhlDOWN

Each character is placed below the previous character in the text string.

NhlUP

Each character is placed below the previous character in the text string.

NhlACROSS

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: NhlACROSS

---

tiYAxisOn

Setting this boolean resource to 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 explicit 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 NhlPosition specifies the side of the viewport relative to which the justification point of the Y-Axis title is located. There are two choices:

NhlLEFT
: Locate the justification point of the Y-Axis title relative to the left edge of the viewport.

NhlRIGHT
: Locate the justification point of the Y-Axis title relative to the right edge of the viewport.

If you set *tiYAxisPosition* to NhlBOTTOM, NhlCENTER, or NhlTOP **Title** issues a warning message and reverts to the default value, NhlLEFT.

Default: NhlLEFT

tiYAxisPosition

This resource of type NhlPosition specifies the base vertical location of the justification point of the Y-Axis title with respect to the viewport boundaries. There are three choices:

NhlBOTTOM
: Locate the base position of the justification point of the Y-Axis title along the line of the bottom edge of the viewport.

NhlCENTER
: Locate the base position of the justification point of the Y-Axis title along the line of the horizontal center of the viewport.

NhlTOP
: 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 NhlRIGHT or NhlLEFT **Title** issues a warning message and reverts to the default value, NhlCENTER.

Default: NhlCENTER

tiYAxisOffsetYF

Sets an offset in NDC coordinates by which the Y-Axis title will be displaced in a direction parallel to Y-Axis from its base location. This value does not scale in response to changes to the viewport size.

Default: 0.0

tiYAxisOffsetXF

Sets an 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 over any offset due to *tiDeltaF*. It does not scale in response to change to the viewport size.

Default: 0.0

tiYAxisJust

This resource of type NhlJustification sets the Justification point of the Y-Axis title.

Default: NhlCENTERCENTER

tiYAxisFont

This resource of type NhlFont 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' 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 *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 *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 *txFont* specifies a filled font (font indexes 21, 22, 25 - 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: NhlHIGH

tiYAxisAngleF

Sets the counter-clockwise rotation angle for the Y-Axis title.

Default: 90.0

tiYAxisDirection

This resource of type NhlTextDirection specifies the direction of the text used for the Y-Axis title. There are three choices of which two are actually the same:

NhlDOWN
: Each character is placed below the previous character in the text string.

NhlUP
: Each character is placed below the previous character in the text string.

NhlACROSS
: 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: NhlACROSS

$Revision: 1.10 $ $Date: 1995/07/24 23:22:15 $

TransObj class

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

Synopsis

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

Resources

Local resources

TransObj Resource Set		
NAME	TYPE	ACCESS
CLASS	DEFAULT	
trOutOfRangeF	NhlTFloat	RCSG
TrOutOfRangeF	1.0e12	

Composite resources

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

Superclass resources

The superclass of the **TransObj** object class does not define any resources.

Description

Objects belonging to the **TransObj** class provide forward and reverse transformations for **Transform** class plot objects. There are currently three subclasses of the **TransObj** class:

- The **LogLinTransObj** provides transformations to and from a linear or logarithmic coordinate space. Either or both coordinate axes may be logarithmic.
- The **IrregularTransObj** provides transformations to and from an irregularly-gridded rectangular coordinate space.
- The **MapTransObj** provides transformations to and from any of 10 different standard map projections.

You do not instantiate any of these objects directly but several **Transform** class plot objects allow you to access their resources.

Support functions

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

Status

See also

- LogLinTransObj object
- IrregularTransObj object
- MapTransObj object
- Transform object
- Obj object

Copyright

Copyright 1987,1988,1989,1991,1993,1994,1995
University Corporation for Atmospheric Research
All Rights Reserved

$Revision: 1.9 $ $Date: 1995/07/23 18:39:05 $

TransObj class resource descriptions

trOutOfRangeF

This resource specifies the value that is to be returned by the low- level library routines for coordinates that transform to an out of range value. You must be careful that it be set to the same value for all **TransObj** class objects belonging to plot objects that are part of the same composite overlay. This includes all member plot objects and the base plot object. In general, it is best not to modify this resource; just leave it set to its default value.

Default: 1.0e12

$Revision: 1.5 $ $Date: 1995/07/23 18:39:06 $

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	
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. Subclasses of the **Transform** class may contain the **PlotManager** class as a composite class member. A transform instantiated with a active **PlotManager** is is known as

a plot object.

All subclasses of **Transform** include as a composite class member at least one subclass of the **TransObj** class to perform the forward and reverse transformations. There are currently three **TransObj** classes available to the subclasses of **Transform**:

- The **LogLinTransObj** provides transformations to and from a linear or logarithmic coordinate space. Either or both coordinate axes may be logarithmic.
- The **IrregularTransObj** provides transformations to and from an irregularly-gridded rectangular coordinate space.
- The **MapTransObj** provides transformations to and from any of 10 different standard map projections.

The **Transform** subclasses may include any or all of these available **TransObj** class members.

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.

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.

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

1: The status of the immediate mode drawing functions, **DataPolyline** and **NDCPolyline**, is not yet resolved.

See also

- **TransObj** object
- **LogLinTransObj** object
- **IrregularTransObj** object
- **MapTransObj** object
- **PlotManager** object

Copyright

Copyright 1987,1988,1989,1991,1993,1994,1995
University Corporation for Atmospheric Research
All Rights Reserved

$Revision: 1.20 $ $Date: 1995/07/24 18:21:58 $

Transform class resource descriptions

tfPlotManagerOn

This boolean resource specifies whether a transform should actually instantiate its **PlotManager** composite class member. It is a create-only resource; if the **PlotManager** is not instantiated when the transform is created it cannot be instantiated later. If the **PlotManager** is activated, the transform becomes a plot object. Otherwise, it is called a simple transform. The primary reason to create a simple transform would be to conserve memory and to make the program run a bit more efficiently.

Default: True

$Revision: 1.8 $ $Date: 1995/07/23 18:39:08 $

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		
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.

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, 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 1987,1988,1989,1991,1993,1994,1995
University Corporation for Atmospheric Research
All Rights Reserved

$Revision: 1.14 $ $Date: 1995/07/23 18:39:09 $

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

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 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

$Revision: 1.7 $ $Date: 1995/07/23 18:39:10 $

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	TYPE	ACCESS
CLASS	DEFAULT	
wsMaximumSize	NhlTLong	RCSG
WsMaximumSize	16777216	
wsThresholdSize	NhlTLong	RCSG
WsThresholdSize	1048576	
wsCurrentSize	NhlTLong	RCSG
WsCurrentSize	0	

Composite resources

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

Superclass resources

The superclasses of the **Workstation** object class do 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 1987,1988,1989,1991,1993,1994,1995
University Corporation for Atmospheric Research
All Rights Reserved

$Revision: 1.12 $ $Date: 1995/07/24 18:21:58 $

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: 1048576

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

$Revision: 1.6 $ $Date: 1995/07/24 18:21:59 $

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:              NhlTColorIndexGenArray
Definition:             An array of NhlTColorIndex elements.

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	NhlTFloatGenArray	RCSG
WkColorMap	<Dynamic>	
wkColorMapLen	NhlTInteger	G
WkColorMapLen	<Dynamic>	

wkBackgroundColor	NhlTFloatGenArray	RCSG
WkBackgroundColor	(/0.0,0.0,0.0/)	
wkForegroundColor	NhlTFloatGenArray	RCSG
WkForegroundColor	<Dynamic>	
wkDashTableLength	NhlTInteger	G
WkDashTableLength	<Dynamic>	
wkFillTableLength	NhlTInteger	G
WkFillTableLength	<Dynamic>	
wkMarkerTableLength	NhlTInteger	G
WkMarkerTableLength	<Dynamic>	
wkGksWorkId	NhlTInteger	G
WkGksWorkId	<Dynamic>	
wkDashPattern	NhlTDashIndex	RCSG
WkDashPattern	NhlSOLIDLINE	
wkLineDashSegLenF	NhlTFloat	RCSG
WkLineDashSegLenF	0.15	
wkLineColor	NhlTColorIndex	RCSG
WkLineColor	NhlFOREGROUND	
wkLineThicknessF	NhlTFloat	RCSG
WkLineThicknessF	1.0	
wkLineLabel	NhlTString	RCSG
WkLineLabel	NULL	
wkLineLabelFont	NhlTFont	RCSG
WkLineLabelFont	"pwritx"	
wkLineLabelFontColor	NhlTColorIndex	RCSG
WkLineLabelFontColor	NhlFOREGROUND	
wkLineLabelFontHeightF	NhlTFloat	RCSG
WkLineLabelFontHeightF	0.0125	
wkLineLabelFontAspectF	NhlTFloat	RCSG
WkLineLabelFontAspectF	1.0	
wkLineLabelFontThicknessF	NhlTFloat	RCSG
WkLineLabelFontThicknessF	1.0	
wkLineLabelFontQuality	NhlTFontQuality	RCSG
WkLineLabelFontQuality	NhlHIGH	
wkLineLabelConstantSpacingF	NhlTFloat	RCSG
WkLineLabelConstantSpacingF	0.0	
wkLineLabelFuncCode	NhlTCharacter	RCSG
WkLineLabelFuncCode	:	
wkMarkerIndex	NhlTMarkerIndex	RCSG
WkMarkerIndex	"*"	
wkMarkerColor	NhlTColorIndex	RCSG

	WkMarkerColor	NhlFOREGROUND	
wkMarkerSizeF	NhlTFloat		RCSG
	WkMarkerSizeF	0.007	
wkMarkerXOffset	NhlTFloat		RCSG
	WkMarkerXOffset	0.0	
wkMarkerYOffset	NhlTFloat		RCSG
	WkMarkerYOffset	0.0	
wkMarkerThicknessF	NhlTFloat		RCSG
	WkMarkerThicknessF	1.0	

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 1987,1988,1989,1991,1993,1994,1995
University Corporation for Atmospheric Research
All Rights Reserved

$Revision: 1.16 $ $Date: 1995/07/23 18:39:13 $

Workstation class resource descriptions

wkColorMap

Use this resource to set or retrieve the colormap for a **Workstation** object. The resource is a two-dimensional array of type NhlTFloat. The fast-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.

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 dynamicly determined based upon 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.

As a convenience, some pre-defined colormap's have been setup. To set the *wkColorMap* to one of the pre-defined colormaps, each pre-defined colormap has been given a string name that the *wkColorMap* resource can be set to. The pre-defined colormaps are as follows:

"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 8 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 8 entries. The first two entries default as described above. The next 6 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 defined 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 5 levels of red, 7 levels of green and 5 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.

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 is a one dimensional array containing three elements of type NhlTFloat that specify (in order) the red, green, and blue components of the **Workstation** background color. 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). 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 is a one dimensional array containing three elements of type NhlTFloat that specify (in order) the red, green, and blue components of the **Workstation** foreground color. 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 brackedted by a GKS activate workstation call and a GKS deactivate workstation call.

Default: <Dynamic>
This value is determined by the low-level GKS library.

wkDashPattern

This resource is used to set the dash pattern index for *immediate mode* drawing functions like NhlNDCPolyline.

Default: NhlSOLIDLINE

wkLineDashSegLenF

This resource is used to set the length in NDC of the dash pattern for *immediate mode* drawing functions like NhlNDCPolyline. This is the length before the dash pattern repeats.

Default: 0.15

wkLineColor

This resource is used to set the color index for the lines drawn with *immediate mode* drawing functions like NhlNDCPolyline.

Default: NhlFOREGROUND

wkLineThicknessF

This resource is used to set the line thickness for the lines drawn with *immediate mode* drawing functions like NhlNDCPolyline.

Default: 1.0

wkLineLabel

This resource is used to set the line label for the lines drawn with *immediate mode* drawing functions like NhlNDCPolyline. If you don’t want line labels set this resource to NULL using the C interface, or a Zero length string from the FORTRAN interface or NCL.

Default: NULL

wkLineLabelFont

This resource is used to set the line label font for the line labels drawn with *immediate mode* drawing functions like NhlNDCPolyline.

Default: "pwritex"

wkLineLabelFontColor

This resource is used to set the line label color index for the line labels drawn with *immediate mode* drawing functions like NhlNDCPolyline.

Default: NhlFOREGROUND

wkLineLabelFontHeightF

This resource is used to set the line label font height for the line labels drawn with *immediate mode* drawing functions like NhlNDCPolyline.

Default: 0.0125

wkLineLabelFontAspectF

This resource is used to set the line label font aspect ratio for the line labels drawn with *immediate mode* drawing functions like NhlNDCPolyline.

Default: 1.0

wkLineLabelFontThicknessF

This resource is used to set the line label font thickness for the line labels drawn with *immediate mode* drawing functions like NhlNDCPolyline.

Default: 1.0

wkLineLabelFontQuality

This resource is used to set the line label font quality for the line labels drawn with *immediate mode* drawing functions like NhlNDCPolyline.

Default: NhlHIGH

wkLineLabelConstantSpacingF

This resource is used to set the constant spacing value for the line labels drawn with *immediate mode* drawing functions like NhlNDCPolyline. Values less than 0.0 result in an error and are replaced with the default value.

Default: 0.0

wkLineLabelFuncCode

This resource is used to set the function code used when parsing line labels drawn with *immediate mode* drawing functions like NhlNDCPolyline.

Default: ':'

wkMarkerIndex

This resource is used to set the marker index for markers drawn with *immediate mode* drawing functions like ??????????.

Default: "*"

wkMarkerColor

This resource is used to set the color index for markers drawn with *immediate mode* drawing functions like ??????????.

Default: NhlFOREGROUND

wkMarkerSizeF

This resource is used to set the marker height for markers drawn with *immediate mode* drawing functions like ??????????.

Default: 0.007

wkMarkerXOffset

This resource is used to set the X offset for markers drawn with *immediate mode*

drawing functions like ??????????.

Default: 0.0

wkMarkerYOffset

This resource is used to set the Y offset for markers drawn with *immediate mode* drawing functions like ??????????.

Default: 0.0

wkMarkerThicknessF

This resource is used to set the thickness of the lines used to render markers drawn with *immediate mode* drawing functions like ??????????.

Default: 1.0

$Revision: 1.11 $ $Date: 1995/07/23 18:39:14 $

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"
                                NhlPRIVATE      = 1,    /* "Private"
                        } NhlXColorMode;
```

Resources

Local resources

```
+-----------------------------------------------------------+
|                      XWorkstation Resource Set            |
```

NAME	TYPE	ACCESS
CLASS	DEFAULT	
wkWindowId	NhlTInteger	RCG
WkWindowId	NULL	
wkXColorMode	NhlTXColorMode	RCG
WkXColorMode	"share"	
wkPause	NhlTBoolean	RCSG
WkPause	<Dynamic>	

Composite resources

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

Superclass resources

You can set all resources defined by the superclasses 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 to 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 1987,1988,1989,1991,1993,1994,1995
University Corporation for Atmospheric Research
All Rights Reserved

$Revision: 1.13 $ $Date: 1995/07/24 18:21:59 $

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.

Default: "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.

$Revision: 1.9 $ $Date: 1995/07/23 18:39:16 $

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,LogLinTransObj,IrregularTransObj

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	
xyXStyle	NhlTTickMarkStyle	RCSG
XyXStyle	NhlLINEAR	

xyYStyle	NhlTTickMarkStyle	RCSG
XyYStyle	NhlLINEAR	
xyXIrregularPoints	NhlTFloatGenArray	RCSG
XyXIrregularPoints	NULL	
xyYIrregularPoints	NhlTFloatGenArray	RCSG
XyYIrregularPoints	NULL	
xyXIrrTensionF	NhlTFloat	RCSG
XyXIrrTensionF	2.0	
xyYIrrTensionF	NhlTFloat	RCSG
XyYIrrTensionF	2.0	
xyComputeXMin	NhlTBoolean	RCSG
XyComputeExtent	<dynamic>	
xyComputeXMax	NhlTBoolean	RCSG
XyComputeExtent	<dynamic>	
xyComputeYMin	NhlTBoolean	RCSG
XyComputeExtent	<dynamic>	
xyComputeYMax	NhlTBoolean	RCSG
XyComputeExtent	<dynamic>	

Data-specific resources

The following resources are not set directly to the **XyPlot** object. The programmer sets them by first adding data to the **XyPlot**'s *xyCoordData* resource using the NhlAddData function. Then the **XyDataSpec** object ids associated with the **DataItem** objects that have been added can be retrieved from the *xyCoordDataSpec* resource.

The **XyDataSpec** class also inherits all the **DataSpec** resources.

XyDataSpec resource set		
NAME	TYPE	ACCESS
CLASS	DEFAULT	
xyDashPattern	NhlTDashIndex	RSG
XyDashPattern	NhlSOLIDLINE	
xyDashPatterns	NhlTDashIndexGenArray	RSG
XyDashPatterns	NULL	
xyMonoDashPattern	NhlTBoolean	RSG
XyMonoDashPattern	False	
xyMarkLineMode	NhlTMarkLineMode	RSG
XyMarkLineMode	NhlLINES	
xyMarkLineModes	NhlTMarkLineModeGenArray	RSG
XyMarkLineModes	NULL	

xyMonoMarkLineMode	NhlTBoolean	RSG
XyMonoMarkLineMode	False	
xyExplicitLegendLabels	NhlTStringGenArray	RSG
XyExplicitLegendLabels	NULL	
xyLineColor	NhlTColorIndex	RSG
XyLineColor	NhlFOREGROUND	
xyLineColors	NhlTColorIndexGenArray	RSG
XyLineColors	NULL	
xyMonoLineColor	NhlTBoolean	RSG
XyMonoLineColor	False	
xyLineDashSegLenF	NhlTFloat	RCSG
XyLineDashSegLenF	<Dynamic>	
xyLineLabelFontHeightF	NhlTFloat	RCSG
XyLineLabelFontHeightF	<Dynamic>	
xyLineLabelFontColor	NhlTColorIndex	RSG
XyLineLabelFontColor	NhlFOREGROUND	
xyLineLabelFontColors	NhlTColorIndexGenArray	RSG
XyLineLabelFontColors	NULL	
xyMonoLineLabelFontColor	NhlTBoolean	RSG
XyMonoLineLabelFontColor	False	
xyLabelMode	NhlTLineLabelMode	RSG
XyLabelMode	NhlNOLABELS	
xyExplicitLabels	NhlTStringGenArray	RSG
XyExplicitLabels	NULL	
xyLineThicknessF	NhlTFloat	RSG
XyLineThicknessF	1.0	
xyLineThicknesses	NhlTFloatGenArray	RSG
XyLineThicknesses	NULL	
xyMonoLineThickness	NhlTBoolean	RSG
XyMonoLineThickness	False	
xyMarker	NhlTMarkerIndex	RSG
XyMarker	"default"	
xyMarkers	NhlTMarkerIndexGenArray	RSG
XyMarkers	NULL	
xyMonoMarker	NhlTBoolean	RSG
XyMonoMarker	False	
xyMarkerColor	NhlTColorIndex	RSG
XyMarkerColor	NhlFOREGROUND	
xyMarkerColors	NhlTColorIndexGenArray	RSG
XyMarkerColors	NULL	
xyMonoMarkerColor	NhlTBoolean	RSG

XyMonoMarkerColor	False	
xyMarkerSizeF	NhlTFloat	RSG
XyMarkerSizeF	0.01	
xyMarkerSizes	NhlTFloatGenArray	RSG
XyMarkerSizes	NULL	
xyMonoMarkerSize	NhlTBoolean	RSG
XyMonoMarkerSize	False	
xyMarkerThicknessF	NhlTFloat	RSG
XyMarkerThicknessF	1.0	
xyMarkerThicknesses	NhlTFloatGenArray	RSG
XyMarkerThicknesses	NULL	
xyMonoMarkerThickness	NhlTBoolean	RSG
XyMonoMarkerThickness	False	

Composite resources

LogLinTransObj resources

If the **XyPlot** *xyXStyle* and *xyYStyle* resources indicate the use of a logarithmic or linear coordinate system, then all the **LogLinTransObj** object resources are available. **XyPlot** modifies some of the **LogLinTransObj** 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.

IrregularTransObj resources

If the **XyPlot** *xyXStyle* and *xyYStyle* resources indicate the use of an irregular coordinate system, then some the **IrregularTransObj** object resources are available. **XyPlot** disables the following resources:

- trXAxisType
- trXCoordPoints

- trXInterPoints
- trXTensionF
- trXSamples
- trYAxisType
- trYCoordPoints
- trYInterPoints
- trYTensionF
- trYSamples

XyPlot modifies **IrregularTransObj** 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.

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 NhlCONDITIONAL.

pmTickMarkDisplayMode

The default value of *pmTickMarkDisplayMode* is set to NhlCONDITIONAL.

pmLegendDisplayMode

The default value of *pmLegendDisplayMode* is set to NhlNEVER.

pmLabelBarDisplayMode

The *pmLabelBarDisplayMode* resource is hard-coded to its default value of NhlNOCREATE 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, **LogLinTransObj**, and **IrregularTransObj** 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 indices, colors of the lines, line labels, and line label colors for curves. **XyPlot** allows the user to configure Marker indices, 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 colors for each element of the *xyCoordData* resource.

The *xyLabelMode* and *xyExplicitLabels* resources control how each curve is labeled. The curve line labels can be any string. However, line labels longer than 6 or 7 characters can cause problems and spaces are not drawn. It is not possible to configure the rest of the text attributes for the line labels.

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 are 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.

See also

- NhlCreate function
- NhlDestroy function
- NhlSetValues function
- NhlGetValues function
- NhlAddData function
- DataComm object
- Title object
- TickMark object

Copyright

Copyright 1987,1988,1989,1991,1993,1994,1995
University Corporation for Atmospheric Research
All Rights Reserved

$Revision: 1.22 $ $Date: 1995/07/24 23:22:16 $

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,LogLinTransObj,IrregularTransObj
```

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	

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. The programmer sets them by first adding data to the **XyPlot**'s *xyCoordData* resource using the NhlAddData function. Then the **XyDataSpec** object ids associated with the **DataItem** objects that have been added can be retrieved from the *xyCoordDataSpec* resource.

The **XyDataSpec** class also inherits all the **DataSpec** resources.

XyDataSpec resource set		
NAME CLASS	TYPE DEFAULT	ACCESS
xyDashPattern XyDashPattern	NhlTDashIndex NhlSOLIDLINE	RSG
xyDashPatterns XyDashPatterns	NhlTDashIndexGenArray NULL	RSG
xyMonoDashPattern XyMonoDashPattern	NhlTBoolean False	RSG
xyMarkLineMode XyMarkLineMode	NhlTMarkLineMode NhlLINES	RSG

xyMarkLineModes	NhlTMarkLineModeGenArray	RSG
XyMarkLineModes	NULL	
xyMonoMarkLineMode	NhlTBoolean	RSG
XyMonoMarkLineMode	False	
xyExplicitLegendLabels	NhlTStringGenArray	RSG
XyExplicitLegendLabels	NULL	
xyLineColor	NhlTColorIndex	RSG
XyLineColor	NhlFOREGROUND	
xyLineColors	NhlTColorIndexGenArray	RSG
XyLineColors	NULL	
xyMonoLineColor	NhlTBoolean	RSG
XyMonoLineColor	False	
xyLineDashSegLenF	NhlTFloat	RCSG
XyLineDashSegLenF	<Dynamic>	
xyLineLabelFontHeightF	NhlTFloat	RCSG
XyLineLabelFontHeightF	<Dynamic>	
xyLineLabelFontColor	NhlTColorIndex	RSG
XyLineLabelFontColor	NhlFOREGROUND	
xyLineLabelFontColors	NhlTColorIndexGenArray	RSG
XyLineLabelFontColors	NULL	
xyMonoLineLabelFontColor	NhlTBoolean	RSG
XyMonoLineLabelFontColor	False	
xyLabelMode	NhlTLineLabelMode	RSG
XyLabelMode	NhlNOLABELS	
xyExplicitLabels	NhlTStringGenArray	RSG
XyExplicitLabels	NULL	
xyLineThicknessF	NhlTFloat	RSG
XyLineThicknessF	1.0	
xyLineThicknesses	NhlTFloatGenArray	RSG
XyLineThicknesses	NULL	
xyMonoLineThickness	NhlTBoolean	RSG
XyMonoLineThickness	False	
xyMarker	NhlTMarkerIndex	RSG
XyMarker	"default"	
xyMarkers	NhlTMarkerIndexGenArray	RSG
XyMarkers	NULL	
xyMonoMarker	NhlTBoolean	RSG
XyMonoMarker	False	
xyMarkerColor	NhlTColorIndex	RSG
XyMarkerColor	NhlFOREGROUND	
xyMarkerColors	NhlTColorIndexGenArray	RSG

XyMarkerColors	NULL	
xyMonoMarkerColor	NhlTBoolean	RSG
XyMonoMarkerColor	False	
xyMarkerSizeF	NhlTFloat	RSG
XyMarkerSizeF	0.01	
xyMarkerSizes	NhlTFloatGenArray	RSG
XyMarkerSizes	NULL	
xyMonoMarkerSize	NhlTBoolean	RSG
XyMonoMarkerSize	False	
xyMarkerThicknessF	NhlTFloat	RSG
XyMarkerThicknessF	1.0	
xyMarkerThicknesses	NhlTFloatGenArray	RSG
XyMarkerThicknesses	NULL	
xyMonoMarkerThickness	NhlTBoolean	RSG
XyMonoMarkerThickness	False	

Composite resources

LogLinTransObj resources

If the **XyPlot** *xyXStyle* and *xyYStyle* resources indicate the use of a logarithmic or linear coordinate system, then all the **LogLinTransObj** object resources are available. **XyPlot** modifies some of the **LogLinTransObj** 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.

IrregularTransObj resources

If the **XyPlot** *xyXStyle* and *xyYStyle* resources indicate the use of an irregular coordinate system, then some the **IrregularTransObj** object resources are available. **XyPlot** disables the following resources:

- trXAxisType
- trXCoordPoints
- trXInterPoints
- trXTensionF
- trXSamples
- trYAxisType
- trYCoordPoints
- trYInterPoints
- trYTensionF
- trYSamples

XyPlot modifies **IrregularTransObj** 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.

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 NhlCONDITIONAL.

pmTickMarkDisplayMode

The default value of *pmTickMarkDisplayMode* is set to NhlCONDITIONAL.

pmLegendDisplayMode

The default value of *pmLegendDisplayMode* is set to NhlNEVER.

pmLabelBarDisplayMode

The *pmLabelBarDisplayMode* resource is hard-coded to its default value of NhlNOCREATE 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, **LogLinTransObj**, and **IrregularTransObj** 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 indices, colors of the lines, line labels, and line label colors for curves. **XyPlot** allows the user to configure Marker indices, 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 colors for each element of the *xyCoordData* resource.

The *xyLabelMode* and *xyExplicitLabels* resources control how each curve is labeled. The curve line labels can be any string. However, line labels longer than 6 or 7 characters can cause problems and spaces are not drawn. It is not possible to configure the rest of the text attributes for the line labels.

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 are 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.

See also

- NhlCreate function
- NhlDestroy function
- NhlSetValues function
- NhlGetValues function
- NhlAddData function
- DataComm object
- Title object
- TickMark object

Copyright

Copyright 1987,1988,1989,1991,1993,1994,1995
University Corporation for Atmospheric Research
All Rights Reserved

$Revision: 1.22 $ $Date: 1995/07/24 23:22:16 $

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

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 tmX[TB]Mode can be set to EXPLICIT and strings representing the alternate scale provided.

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 tmY[TB]Mode can be set to EXPLICIT and strings representing the alternate scale provided.

Default: LINEAR

xyXIrregularPoints

An array of floats that specify a discrete representation of an irregular coordinate system. The values in this array must be monotonic 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 monotonic 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: NhlSOLIDLINE

xyDashPatterns
An array of integer dash pattern indices to use when drawing the corresponding curves in the associated data object. If there are more curves than the number of dash pattern indices 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: NhlLINES

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: NhlFOREGROUND

xyLineColors

An array of color indices to use when drawing the corresponding curves in the associated data object. If there are more curves than the number of color indices 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 NhlLETTERED or NhlCUSTOM. This value is only used if xyMonoLineLabelFontColor is True, or the xyLineLabelFontColors array doesn't specifiy enough values.

Default: NhlFOREGROUND

xyLineLabelFontColors
This resource indicates what color index to draw the line labels in if xyLabelMode is set to NhlLETTERED or NhlCUSTOM. 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 line-width 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 indices 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: NhlFOREGROUND

xyMarkerColors

An array of color indices 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 indices 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 NDC.

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

$Revision: 1.17 $ $Date: 1995/07/25 02:23:14 $

HLU API - alphabetically

NHLPERROR

The Fortran names of these functions are **NhlFPError**, **NhlFErrGetId**, **NhlFErrNumMsgs**, **NhlFErrGetMsg**, **NhlFErrCleanMsgs**, **NhlFErrSpringMsg**, and **NhlFErrFPrintMsg**. There are no Fortran bindings for **NHLPEROR** and **NhlErrAddTable**.

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.

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.

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

The Fortran names of these functions are **NhlFPError**, **NhlFErrGetId**, **NhlFErrNumMsgs**, **NhlFErrGetMsg**, **NhlFErrCleanMsgs**, **NhlFErrSpringMsg**, and **NhlFErrFPrintMsg**. There are no Fortran bindings for **NHLPEROR** and **NhlErrAddTable**.

Error handling functions for the HLU library.

NhlErrClearMsgs

The Fortran names of these functions are **NhlFPError**, **NhlFErrGetId**,

NhlFErrNumMsgs, **NhlFErrGetMsg**, **NhlFErrCleanMsgs**, **NhlFErrSpringMsg**, and **NhlFErrFPrintMsg**. There are no Fortran bindings for **NHLPEROR** and **NhlErrAddTable**.

Error handling functions for the HLU library.

NhlErrFPrintMsg

The Fortran names of these functions are **NhlFPError**, **NhlFErrGetId**, **NhlFErrNumMsgs**, **NhlFErrGetMsg**, **NhlFErrCleanMsgs**, **NhlFErrSpringMsg**, and **NhlFErrFPrintMsg**. There are no Fortran bindings for **NHLPEROR** and **NhlErrAddTable**.

Error handling functions for the HLU library.

NhlErrGetId

The Fortran names of these functions are **NhlFPError**, **NhlFErrGetId**, **NhlFErrNumMsgs**, **NhlFErrGetMsg**, **NhlFErrCleanMsgs**, **NhlFErrSpringMsg**, and **NhlFErrFPrintMsg**. There are no Fortran bindings for **NHLPEROR** and **NhlErrAddTable**.

Error handling functions for the HLU library.

NhlErrGetMsg

The Fortran names of these functions are **NhlFPError**, **NhlFErrGetId**, **NhlFErrNumMsgs**, **NhlFErrGetMsg**, **NhlFErrCleanMsgs**, **NhlFErrSpringMsg**, and **NhlFErrFPrintMsg**. There are no Fortran bindings for **NHLPEROR** and **NhlErrAddTable**.

Error handling functions for the HLU library.

NhlErrNumMsgs

The Fortran names of these functions are **NhlFPError**, **NhlFErrGetId**, **NhlFErrNumMsgs**, **NhlFErrGetMsg**, **NhlFErrCleanMsgs**, **NhlFErrSpringMsg**, and **NhlFErrFPrintMsg**. There are no Fortran bindings for **NHLPEROR** and **NhlErrAddTable**.

Error handling functions for the HLU library.

NhlErrSPrintMsg

The Fortran names of these functions are **NhlFPError**, **NhlFErrGetId**, **NhlFErrNumMsgs**, **NhlFErrGetMsg**, **NhlFErrCleanMsgs**, **NhlFErrSpringMsg**, and **NhlFErrFPrintMsg**. There are no Fortran bindings for **NHLPEROR** and **NhlErrAddTable**.

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 routines.

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 these functions are **NhlFOpen** and **NhlFInitialize**.

HLU library initialization functions.

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.

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 routines.

Memory allocation functions.

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.

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 routine.

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

NhlOpen

The Fortran names of these functions are **NhlFOpen** and **NhlFInitialize**.

HLU library initialization functions.

NhlPError

The Fortran names of these functions are **NhlFPError**, **NhlFErrGetId**, **NhlFErrNumMsgs**, **NhlFErrGetMsg**, **NhlFErrCleanMsgs**, **NhlFErrSpringMsg**, and **NhlFErrFPrintMsg**. There are no Fortran bindings for **NHLPEROR** and **NhlErrAddTable**.

Error handling functions for the HLU library.

NhlRLClear

The Fortran names of these functions are **NhlFRLCreate**, **NhlFRLDestroy**, **NhlFRLClear**, **NhlFRLUnSet**, and **NhlFRLIsSet**. *ResList* support functions.

NhlRLCreate

The Fortran names of these functions are **NhlFRLCreate**, **NhlFRLDestroy**, **NhlFRLClear**, **NhlFRLUnSet**, and **NhlFRLIsSet**. *ResList* support functions.

NhlRLDestroy

The Fortran names of these functions are **NhlFRLCreate**, **NhlFRLDestroy**, **NhlFRLClear**, **NhlFRLUnSet**, and **NhlFRLIsSet**. *ResList* support functions.

NhlRLGet

The Fortran names of these functions are **NhlFRLGetInteger**, **NhlFRLGetFloat**, **NhlFRLGetString**. There is no Fortran binding for **NhlRLGet**.

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

NhlRLGetArray

The Fortran names of these functions are **NhlFRLGetIntegerArray**, **NhlFRLGetFloatArray**, **NhlFRLGetStringArray**. There is no Fortran binding for **NhlRLGetArray**.

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

NhlRLGetFloat

The Fortran names of these functions are **NhlFRLGetInteger**, **NhlFRLGetFloat**, **NhlFRLGetString**. There is no Fortran binding for **NhlRLGet**.

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

NhlRLGetFloatArray

The Fortran names of these functions are **NhlFRLGetIntegerArray**, **NhlFRLGetFloatArray**, **NhlFRLGetStringArray**. There is no Fortran binding for **NhlRLGetArray**.

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

NhlRLGetInteger

The Fortran names of these functions are **NhlFRLGetInteger**,

NhlFRLGetFloat, **NhlFRLGetString**. There is no Fortran binding for **NhlRLGet**.

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

NhlRLGetIntegerArray

The Fortran names of these functions are **NhlFRLGetIntegerArray**, **NhlFRLGetFloatArray**, **NhlFRLGetStringArray**. There is no Fortran binding for **NhlRLGetArray**.

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

NhlRLGetMDArray

The Fortran names of these functions are **NhlFRLGetMDIntegerArray**, **NhlFRLGetMDFloatArray**. There is no Fortran binding for **NhlRLGetMDArray**.

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

NhlRLGetMDFloatArray

The Fortran names of these functions are **NhlFRLGetMDIntegerArray**, **NhlFRLGetMDFloatArray**. There is no Fortran binding for **NhlRLGetMDArray**.

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

NhlRLGetMDIntegerArray

The Fortran names of these functions are **NhlFRLGetMDIntegerArray**, **NhlFRLGetMDFloatArray**. There is no Fortran binding for **NhlRLGetMDArray**.

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

NhlRLGetString

The Fortran names of these functions are **NhlFRLGetInteger**, **NhlFRLGetFloat**, **NhlFRLGetString**. There is no Fortran binding for **NhlRLGet**.

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

NhlRLGetStringArray

The Fortran names of these functions are **NhlFRLGetIntegerArray**, **NhlFRLGetFloatArray**, **NhlFRLGetStringArray**. There is no Fortran binding for **NhlRLGetArray**.

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

NhlRLIsSet

The Fortran names of these functions are **NhlFRLCreate**, **NhlFRLDestroy**, **NhlFRLClear**, **NhlFRLUnSet**, and **NhlFRLIsSet**. *ResList* support functions.

NhlRLSet

The Fortran names of these functions are **NhlFRLSetInteger**, **NhlFRLSetFloat**, **NhlFRLSetString**. There is no Fortran binding for **NhlRLSet**.

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

NhlRLSetArray

The Fortran names of these functions are **NhlFRLSetIntegerArray**, **NhlFRLSetFloatArray**, **NhlFRLSetStringArray**. There is no Fortran binding for **NhlRLSetArray**.

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

NhlRLSetFloat

The Fortran names of these functions are **NhlFRLSetInteger**, **NhlFRLSetFloat**, **NhlFRLSetString**. There is no Fortran binding for **NhlRLSet**.

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

NhlRLSetFloatArray

The Fortran names of these functions are **NhlFRLSetIntegerArray**, **NhlFRLSetFloatArray**, **NhlFRLSetStringArray**. There is no Fortran binding for **NhlRLSetArray**.

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

NhlRLSetInteger

The Fortran names of these functions are **NhlFRLSetInteger**, **NhlFRLSetFloat**, **NhlFRLSetString**. There is no Fortran binding for **NhlRLSet**.

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

NhlRLSetIntegerArray

The Fortran names of these functions are **NhlFRLSetIntegerArray**, **NhlFRLSetFloatArray**, **NhlFRLSetStringArray**. There is no Fortran binding for **NhlRLSetArray**.

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

NhlRLSetMDArray

The Fortran names of these functions are **NhlFRLSetMDIntegerArray**, **NhlFRLSetMDFloatArray**. There is no Fortran binding for **NhlRLSetMDArray**.

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

NhlRLSetMDFloatArray

The Fortran names of these functions are **NhlFRLSetMDIntegerArray**, **NhlFRLSetMDFloatArray**. There is no Fortran binding for **NhlRLSetMDArray**.

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

NhlRLSetMDIntegerArray

The Fortran names of these functions are **NhlFRLSetMDIntegerArray**, **NhlFRLSetMDFloatArray**. There is no Fortran binding for **NhlRLSetMDArray**.

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

NhlRLSetString

The Fortran names of these functions are **NhlFRLSetInteger**, **NhlFRLSetFloat**, **NhlFRLSetString**. There is no Fortran binding for **NhlRLSet**.

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

NhlRLSetStringArray

The Fortran names of these functions are **NhlFRLSetIntegerArray**, **NhlFRLSetFloatArray**, **NhlFRLSetStringArray**. There is no Fortran binding for **NhlRLSetArray**.

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

NhlRLUnSet

The Fortran names of these functions are **NhlFRLCreate**, **NhlFRLDestroy**, **NhlFRLClear**, **NhlFRLUnSet**, and **NhlFRLIsSet**. *ResList* support functions.

NhlRealloc

No corresponding Fortran routines.

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.

$Revision: 1.4 $ $Date: 1995/07/23 20:18:13 $

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.

$Revision: 1.3 $ $Date: 1995/07/19 17:47:12 $

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
: Identifier of the plot object to which the **View** object is to be added as an annotation.

view_id

Identifier of the **View** object to be added as an annotation of the plot object.
anno_manager_id (Fortran only)
If successful, identifier of new **AnnoManager** object; else error code.

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 1987,1988,1989,1991,1993,1994,1995
University Corporation for Atmospheric Research
All Rights Reserved

$Revision: 1.11 $ $Date: 1995/07/22 20:03:49 $

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
Identifies the **DataComm** object with the data resource.
res_name
Identifies the data resource to set in the **DataComm** object.
ditemid
Identifies the **DataItem** object id to associate with the *res_name* resource in the *dcommid* **DataComm** object.
id (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 1987,1988,1989,1991,1993,1994,1995
University Corporation for Atmospheric Research
All Rights Reserved

$Revision: 1.11 $ $Date: 1995/07/22 20:03:50 $

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
: Identifies the plot object to be used as the base plot.

transform_id
: Identifies the **Transform** object to be overlaid on the base plot.

after_id
: 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 (Fortran only)
: Error code.

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 1987,1988,1989,1991,1993,1994,1995
University Corporation for Atmospheric Research
All Rights Reserved

$Revision: 1.8 $ $Date: 1995/07/22 20:03:51 $

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 (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 **App** object that is currently being used as the *default parent*. Otherwise, the value will be NhlFATAL.

See also

- **App object**

Copyright

Copyright 1987,1988,1989,1991,1993,1994,1995
University Corporation for Atmospheric Research
All Rights Reserved

$Revision: 1.4 $ $Date: 1995/07/22 20:03:51 $

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
: Id that identifies the graphical object to move to another Workstation class parent.

work_id
: Id that identifies the workstation object to move the graphical object to.

ierr (Fortran only)
: Error code.

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 1987,1988,1989,1991,1993,1994,1995
University Corporation for Atmospheric Research
All Rights Reserved

$Revision: 1.11 $ $Date: 1995/07/22 20:03:52 $

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
: Specifies the object identifier.

name_ret (Fortran only)
: On output, the class name of the object specified by the *id* argument.

ierr (Fortran only)
: Error code.

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, and the **NhlFClassName** Fortran subroutine returns the error in *ierr*.

See also

- NhlCreate

Copyright

Copyright 1987,1988,1989,1991,1993,1994,1995
University Corporation for Atmospheric Research
All Rights Reserved

$Revision: 1.8 $ $Date: 1995/07/22 20:03:52 $

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
    Integer identifier of an instance of a **Workstation** class object
ierr (Fortran only)
    Error code.

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; continuing the program may result in core dump

---

See Also

- **NhlDraw**
- **NhlUpdateWorkstation**
- **NhlFrame**
- **Workstation object**
- **XWorkstation object**
- **NcgmWorkstation object**

Copyright

Copyright 1987,1988,1989,1991,1993,1994,1995
University Corporation for Atmospheric Research
All Rights Reserved

$Revision: 1.8 $ $Date: 1995/07/22 20:03:53 $

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.

Copyright

Copyright 1987,1988,1989,1991,1993,1994,1995
University Corporation for Atmospheric Research
All Rights Reserved

$Revision: 1.6 $ $Date: 1995/07/22 20:03:53 $

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, ier
      integer ret_id, parent_id, rlist_id, ierr
      character*(*) name
      external lc
```

Arguments

ret_id
: 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
: Sets the name of the object being created.

lc
: Identifies the class of object to create, i.e. the *type* of object to create.

parent_id
: 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
: 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`.

ierr (Fortran only)
: Error code.

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 1987,1988,1989,1991,1993,1994,1995
University Corporation for Atmospheric Research
All Rights Reserved

$Revision: 1.14 $ $Date: 1995/07/22 20:03:54 $

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,
                float   *x,
                float   *y,
                int     n,
                )
```

Fortran Synopsis

```
      subroutine NhlFDataPolyline(pid, x, y, n, ierr)
      integer pid, n, ierr
      real x(n), y(n)
```

Arguments

pid
: Id of a **Transform** class plot object with an established data transformation.

x
: Array of X-Axis coordinates values in data coordinate space.

y
: Array of Y-Axis coordinates values in data coordinate space.

n
: Number of elements in the coordinate arrays *x* and *y*.

ierr (Fortran only)
: Error code.

Description

This function draws an immediate mode polyline in the current data coordinate space of the **Transform** class plot object specified by the parameter *pid*. An *immediate mode* polyline is a polyline that appears as soon as the low-level point buffer is flushed after the function is called. It is not connected with the plot object's **Draw** method.

V4.0 NOTE: There is currently no public interface for setting the line attributes of immediate mode polylines.

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

Copyright

Copyright 1987,1988,1989,1991,1993,1994,1995
University Corporation for Atmospheric Research
All Rights Reserved

$Revision: 1.6 $ $Date: 1995/07/22 20:03:54 $

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_r
```

Arguments

pid
: Id of a **Transform** class plot object with an established data transformation.

x
: Array of X-Axis coordinates values in data coordinate space.

y
: Array of Y-Axis coordinates values in data coordinate space.

n
: Number of elements in each of the arrays *x*, *y*, *xout* and *yout*.

xout
: Array for X-Axis output values in NDC space. The user is responsible for providing the appropriate amount of storage.

yout
: Array for Y-Axis output values in NDC space. The user is responsible for providing the appropriate amount of storage.

xmissing
: 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
: 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 (Fortran only)
: Flag for whether or not any missing values exist for the X-Axis input array; if 1, then missing values exists.

iymissing (Fortran only)
: Flag for whether or not any missing values exist for the Y-Axis input array; if 1, then missing value exists..

status
: 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
: 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 (Fortran only)
: Error code.

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 1987,1988,1989,1991,1993,1994,1995
University Corporation for Atmospheric Research
All Rights Reserved

$Revision: 1.8 $ $Date: 1995/07/22 20:03:55 $

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
: Identifies the HLU object to destroy. Returned from the NhlCreate function.

ierr (Fortran only)
: Error code.

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 1987,1988,1989,1991,1993,1994,1995
University Corporation for Atmospheric Research
All Rights Reserved

$Revision: 1.9 $ $Date: 1995/07/22 20:03:55 $

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
Identifies the HLU object to draw. Returned from the NhlCreate function.

ierr (Fortran only)
Error code.

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 1987,1988,1989,1991,1993,1994,1995
University Corporation for Atmospheric Research
All Rights Reserved

$Revision: 1.9 $ $Date: 1995/07/22 20:03:56 $

NhlPError

NhlPError, NHLPERROR, NhlErrGetId, NhlErrNumMsgs, NhlErrGetMsg, NhlErrCleanMsgs, NhlErrAddTable, NhlErrSPrintMsg, NhlErrFPrintMsg

The Fortran names of these functions are **NhlFPError**, **NhlFErrGetId**, **NhlFErrNumMsgs**, **NhlFErrGetMsg**, **NhlFErrCleanMsgs**, **NhlFErrSpringMsg**, and **NhlFErrFPrintMsg**. There are no Fortran bindings for **NHLPEROR** 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, format)
      character*(*) severity, format
      integer errnum

      subroutine NhlFErrGetId(id_ret)
      integer id_ret

      subroutine NhlFErrNumMsgs(nummsg)
      integer nummsg

      subroutine NhlFErrGetMsg(msgnum, severity, emsg, errorno, sysmsg
     +                         line, fname, ierr)
      integer msgnum, severity, 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
: Specifies the severity level of the error message being reported.

errnum
: Specifies a unique error number that references an error message from an installed error table.

format
: Format string like printf.

...
: Arguments for format string aka printf.

msgnum
: Identifies the index of the error message to retrieve.

msgptr (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.

msg (C only)
: Pointer to an NhlErrMsg structure that fully describes an error message.

start (C only)
: Index to assign to the first error message in the table being installed.

tlen (C only)
: Length of the error table being installed.

etable (C only)
: Pointer to an error table to install.

buffer
: Character buffer to place the formatted message into.

fp (C only)
: File pointer to print formatted error message to.

nummsg (Fortran only)
: Number of error messages currently buffered.

id_ret (Fortran only)
: Id for *Error object*.

iunit (Fortran only)
: Unit number to print formatted error message to.

emsg (Fortran only)
: Actual text of the error message.

errorno (Fortran only)
: Error number.

sysmsg (Fortran only)
: Actual text in the error table indexed by *errorno*.

line (Fortran only)
: Line number the error message was reported from, or 0.

fname (Fortran only)
: File name the error message was reported from, or NULL.

ierr (Fortran only)
: Error code.

Types

```
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 1987,1988,1989,1991,1993,1994,1995
University Corporation for Atmospheric Research
All Rights Reserved

$Revision: 1.12 $ $Date: 1995/07/22 20:04:08 $

NhlPError

NhlPError, NHLPERROR, NhlErrGetId, NhlErrNumMsgs, NhlErrGetMsg, NhlErrCleanMsgs, NhlErrAddTable, NhlErrSPrintMsg, NhlErrFPrintMsg

The Fortran names of these functions are **NhlFPError**, **NhlFErrGetId**, **NhlFErrNumMsgs**, **NhlFErrGetMsg**, **NhlFErrCleanMsgs**, **NhlFErrSpringMsg**, and **NhlFErrFPrintMsg**. There are no Fortran bindings for **NHLPEROR** 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, format)
      character*(*) severity, format
      integer errnum

      subroutine NhlFErrGetId(id_ret)
      integer id_ret

      subroutine NhlFErrNumMsgs(nummsg)
      integer nummsg

      subroutine NhlFErrGetMsg(msgnum, severity, emsg, errorno, sysmsg
     +                         line, fname, ierr)
      integer msgnum, severity, 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
: Specifies the severity level of the error message being reported.

errnum
: Specifies a unique error number that references an error message from an installed error table.

format
: Format string like printf.

...
: Arguments for format string aka printf.

msgnum
: Identifies the index of the error message to retrieve.

msgptr (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.

msg (C only)
: Pointer to an NhlErrMsg structure that fully describes an error message.

start (C only)
: Index to assign to the first error message in the table being installed.

tlen (C only)
: Length of the error table being installed.

etable (C only)
: Pointer to an error table to install.

buffer
: Character buffer to place the formatted message into.

fp (C only)
: File pointer to print formatted error message to.

nummsg (Fortran only)
: Number of error messages currently buffered.

id_ret (Fortran only)
: Id for *Error object*.

iunit (Fortran only)
: Unit number to print formatted error message to.

emsg (Fortran only)
: Actual text of the error message.

errorno (Fortran only)
: Error number.

sysmsg (Fortran only)
: Actual text in the error table indexed by *errorno*.

line (Fortran only)
: Line number the error message was reported from, or 0.

fname (Fortran only)

File name the error message was reported from, or NULL.

ierr (Fortran only)

Error code.

Types

```
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 1987,1988,1989,1991,1993,1994,1995
University Corporation for Atmospheric Research
All Rights Reserved

$Revision: 1.12 $ $Date: 1995/07/22 20:04:08 $

NhlPError

NhlPError, NHLPERROR, NhlErrGetId, NhlErrNumMsgs, NhlErrGetMsg, NhlErrCleanMsgs, NhlErrAddTable, NhlErrSPrintMsg, NhlErrFPrintMsg

The Fortran names of these functions are **NhlFPError**, **NhlFErrGetId**, **NhlFErrNumMsgs**, **NhlFErrGetMsg**, **NhlFErrCleanMsgs**, **NhlFErrSpringMsg**, and **NhlFErrFPrintMsg**. There are no Fortran bindings for **NHLPEROR** 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, format)
      character*(*) severity, format
      integer errnum

      subroutine NhlFErrGetId(id_ret)
      integer id_ret

      subroutine NhlFErrNumMsgs(nummsg)
      integer nummsg

      subroutine NhlFErrGetMsg(msgnum, severity, emsg, errorno, sysmsg
     +                         line, fname, ierr)
      integer msgnum, severity, 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
: Specifies the severity level of the error message being reported.

errnum
: Specifies a unique error number that references an error message from an installed error table.

format
: Format string like printf.

...
: Arguments for format string aka printf.

msgnum
: Identifies the index of the error message to retrieve.

msgptr (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.

msg (C only)
: Pointer to an NhlErrMsg structure that fully describes an error message.

start (C only)
: Index to assign to the first error message in the table being installed.

tlen (C only)
: Length of the error table being installed.

etable (C only)
: Pointer to an error table to install.

buffer
: Character buffer to place the formatted message into.

fp (C only)
: File pointer to print formatted error message to.

nummsg (Fortran only)
: Number of error messages currently buffered.

id_ret (Fortran only)
: Id for *Error object*.

iunit (Fortran only)
: Unit number to print formatted error message to.

emsg (Fortran only)
: Actual text of the error message.

errorno (Fortran only)
: Error number.

sysmsg (Fortran only)
: Actual text in the error table indexed by *errorno*.

line (Fortran only)
Line number the error message was reported from, or 0.
fname (Fortran only)
File name the error message was reported from, or NULL.
ierr (Fortran only)
Error code.

Types

```
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 1987,1988,1989,1991,1993,1994,1995
University Corporation for Atmospheric Research
All Rights Reserved

$Revision: 1.12 $ $Date: 1995/07/22 20:04:08 $

NhlPError

NhlPError, NHLPERROR, NhlErrGetId, NhlErrNumMsgs, NhlErrGetMsg, NhlErrCleanMsgs, NhlErrAddTable, NhlErrSPrintMsg, NhlErrFPrintMsg

The Fortran names of these functions are **NhlFPError**, **NhlFErrGetId**, **NhlFErrNumMsgs**, **NhlFErrGetMsg**, **NhlFErrCleanMsgs**, **NhlFErrSpringMsg**, and **NhlFErrFPrintMsg**. There are no Fortran bindings for **NHLPEROR** 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, format)
      character*(*) severity, format
      integer errnum

      subroutine NhlFErrGetId(id_ret)
      integer id_ret

      subroutine NhlFErrNumMsgs(nummsg)
      integer nummsg

      subroutine NhlFErrGetMsg(msgnum, severity, emsg, errorno, sysmsg
     +                         line, fname, ierr)
      integer msgnum, severity, 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
: Specifies the severity level of the error message being reported.

errnum
: Specifies a unique error number that references an error message from an installed error table.

format
: Format string like printf.

...
: Arguments for format string aka printf.

msgnum
: Identifies the index of the error message to retrieve.

msgptr (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.

msg (C only)
: Pointer to an NhlErrMsg structure that fully describes an error message.

start (C only)
: Index to assign to the first error message in the table being installed.

tlen (C only)
: Length of the error table being installed.

etable (C only)
: Pointer to an error table to install.

buffer
: Character buffer to place the formatted message into.

fp (C only)
: File pointer to print formatted error message to.

nummsg (Fortran only)
: Number of error messages currently buffered.

id_ret (Fortran only)
: Id for *Error object*.

iunit (Fortran only)
: Unit number to print formatted error message to.

emsg (Fortran only)
: Actual text of the error message.

errorno (Fortran only)
: Error number.

sysmsg (Fortran only)
Actual text in the error table indexed by *errorno*.
line (Fortran only)
Line number the error message was reported from, or 0.
fname (Fortran only)
File name the error message was reported from, or NULL.
ierr (Fortran only)
Error code.

Types

```
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 1987,1988,1989,1991,1993,1994,1995
University Corporation for Atmospheric Research
All Rights Reserved

$Revision: 1.12 $ $Date: 1995/07/22 20:04:08 $

NhlPError

NhlPError, NHLPERROR, NhlErrGetId, NhlErrNumMsgs, NhlErrGetMsg, NhlErrCleanMsgs, NhlErrAddTable, NhlErrSPrintMsg, NhlErrFPrintMsg

The Fortran names of these functions are **NhlFPError**, **NhlFErrGetId**, **NhlFErrNumMsgs**, **NhlFErrGetMsg**, **NhlFErrCleanMsgs**, **NhlFErrSpringMsg**, and **NhlFErrFPrintMsg**. There are no Fortran bindings for **NHLPEROR** 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, format)
      character*(*) severity, format
      integer errnum

      subroutine NhlFErrGetId(id_ret)
      integer id_ret

      subroutine NhlFErrNumMsgs(nummsg)
      integer nummsg

      subroutine NhlFErrGetMsg(msgnum, severity, emsg, errorno, sysmsg
```

```
+                           line, fname, ierr)
 integer msgnum, severity, 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
: Specifies the severity level of the error message being reported.

errnum
: Specifies a unique error number that references an error message from an installed error table.

format
: Format string like printf.

...
: Arguments for format string aka printf.

msgnum
: Identifies the index of the error message to retrieve.

msgptr (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.

msg (C only)
: Pointer to an NhlErrMsg structure that fully describes an error message.

start (C only)
: Index to assign to the first error message in the table being installed.

tlen (C only)
: Length of the error table being installed.

etable (C only)
: Pointer to an error table to install.

buffer
: Character buffer to place the formatted message into.

fp (C only)
: File pointer to print formatted error message to.

nummsg (Fortran only)
: Number of error messages currently buffered.

id_ret (Fortran only)
: Id for *Error object*.

iunit (Fortran only)
: Unit number to print formatted error message to.

emsg (Fortran only)

Actual text of the error message.

errorno (Fortran only)
Error number.

sysmsg (Fortran only)
Actual text in the error table indexed by *errorno*.

line (Fortran only)
Line number the error message was reported from, or 0.

fname (Fortran only)
File name the error message was reported from, or NULL.

ierr (Fortran only)
Error code.

Types

```
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 1987,1988,1989,1991,1993,1994,1995
University Corporation for Atmospheric Research
All Rights Reserved

$Revision: 1.12 $ $Date: 1995/07/22 20:04:08 $

NhlPError

NhlPError, NHLPERROR, NhlErrGetId, NhlErrNumMsgs, NhlErrGetMsg, NhlErrCleanMsgs, NhlErrAddTable, NhlErrSPrintMsg, NhlErrFPrintMsg

The Fortran names of these functions are **NhlFPError**, **NhlFErrGetId**, **NhlFErrNumMsgs**, **NhlFErrGetMsg**, **NhlFErrCleanMsgs**, **NhlFErrSpringMsg**, and **NhlFErrFPrintMsg**. There are no Fortran bindings for **NHLPEROR** 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, format)
      character*(*) severity, format
      integer errnum

      subroutine NhlFErrGetId(id_ret)
      integer id_ret

      subroutine NhlFErrNumMsgs(nummsg)
```

```
      integer nummsg

      subroutine NhlFErrGetMsg(msgnum, severity, emsg, errorno, sysmsg
     +                         line, fname, ierr)
      integer msgnum, severity, 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
: Specifies the severity level of the error message being reported.

errnum
: Specifies a unique error number that references an error message from an installed error table.

format
: Format string like printf.

...
: Arguments for format string aka printf.

msgnum
: Identifies the index of the error message to retrieve.

msgptr (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.

msg (C only)
: Pointer to an NhlErrMsg structure that fully describes an error message.

start (C only)
: Index to assign to the first error message in the table being installed.

tlen (C only)
: Length of the error table being installed.

etable (C only)
: Pointer to an error table to install.

buffer
: Character buffer to place the formatted message into.

fp (C only)
: File pointer to print formatted error message to.

nummsg (Fortran only)
: Number of error messages currently buffered.

id_ret (Fortran only)
: Id for *Error object*.

iunit (Fortran only)
Unit number to print formatted error message to.
emsg (Fortran only)
Actual text of the error message.
errorno (Fortran only)
Error number.
sysmsg (Fortran only)
Actual text in the error table indexed by *errorno*.
line (Fortran only)
Line number the error message was reported from, or 0.
fname (Fortran only)
File name the error message was reported from, or NULL.
ierr (Fortran only)
Error code.

Types

```
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 1987,1988,1989,1991,1993,1994,1995
University Corporation for Atmospheric Research
All Rights Reserved

$Revision: 1.12 $ $Date: 1995/07/22 20:04:08 $

NhlPError

NhlPError, NHLPERROR, NhlErrGetId, NhlErrNumMsgs, NhlErrGetMsg, NhlErrCleanMsgs, NhlErrAddTable, NhlErrSPrintMsg, NhlErrFPrintMsg

The Fortran names of these functions are **NhlFPError**, **NhlFErrGetId**, **NhlFErrNumMsgs**, **NhlFErrGetMsg**, **NhlFErrCleanMsgs**, **NhlFErrSpringMsg**, and **NhlFErrFPrintMsg**. There are no Fortran bindings for **NHLPEROR** 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, format)
      character*(*) severity, format
      integer errnum
```

```
      subroutine NhlFErrGetId(id_ret)
      integer id_ret

      subroutine NhlFErrNumMsgs(nummsg)
      integer nummsg

      subroutine NhlFErrGetMsg(msgnum, severity, emsg, errorno, sysmsg
     +                         line, fname, ierr)
      integer msgnum, severity, 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
: Specifies the severity level of the error message being reported.

errnum
: Specifies a unique error number that references an error message from an installed error table.

format
: Format string like printf.

...
: Arguments for format string aka printf.

msgnum
: Identifies the index of the error message to retrieve.

msgptr (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.

msg (C only)
: Pointer to an NhlErrMsg structure that fully describes an error message.

start (C only)
: Index to assign to the first error message in the table being installed.

tlen (C only)
: Length of the error table being installed.

etable (C only)
: Pointer to an error table to install.

buffer
: Character buffer to place the formatted message into.

fp (C only)
: File pointer to print formatted error message to.

nummsg (Fortran only)

Number of error messages currently buffered.

id_ret (Fortran only)
Id for *Error object*.

iunit (Fortran only)
Unit number to print formatted error message to.

emsg (Fortran only)
Actual text of the error message.

errorno (Fortran only)
Error number.

sysmsg (Fortran only)
Actual text in the error table indexed by *errorno*.

line (Fortran only)
Line number the error message was reported from, or 0.

fname (Fortran only)
File name the error message was reported from, or NULL.

ierr (Fortran only)
Error code.

Types

```
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 1987,1988,1989,1991,1993,1994,1995
University Corporation for Atmospheric Research
All Rights Reserved

$Revision: 1.12 $ $Date: 1995/07/22 20:04:08 $

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
Integer identifier of an instance of a **Workstation** class object
ierr (Fortran only)
Error code

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 1987,1988,1989,1991,1993,1994,1995
University Corporation for Atmospheric Research
All Rights Reserved

$Revision: 1.6 $ $Date: 1995/07/22 20:03:56 $

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
: Specifies the size of the memory to allocate.

data
: Specifies a pointer to memory previously returned from NhlMalloc or NhlRealloc.

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 NhlNOERROR if there is no error, or NhlWARNING if there is a detectable error.

Copyright

Copyright 1987,1988,1989,1991,1993,1994,1995
University Corporation for Atmospheric Research

All Rights Reserved

$Revision: 1.7 $ $Date: 1995/07/22 20:04:04 $

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
: Integer identifier of an instance of a **Workstation** class object

ci
: Integer color index in the range 1 through 255 of a currently allocated color.

ierr (Fortran only)
: Error code.

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 1987,1988,1989,1991,1993,1994,1995
University Corporation for Atmospheric Research
All Rights Reserved

$Revision: 1.6 $ $Date: 1995/07/22 20:03:57 $

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>

typedef struct _NhlBoundingBox {
        int     set;
        float   t;
        float   b;
        float   l;
        float   r;
} NhlBoundingBox;

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
: Id of object whose bounding box is requested.

thebox
: 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
: Id of object whose bounding box is requested.

top
On output contains the NDC value of the top edge of the bounding box of the **View** class object.

bottom
On output contains the NDC value of the bottom edge of the bounding box of the **View** class object.

left
On output contains the NDC value of the left edge of the bounding box of the **View** class object.

right
On output contains the NDC value of the right edge of the bounding box of the **View** class object.

ierr
Error code.

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 1987,1988,1989,1991,1993,1994,1995
University Corporation for Atmospheric Research
All Rights Reserved

$Revision: 1.6 $ $Date: 1995/07/22 20:03:58 $

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
: Integer identifier of an instance of a **Workstation** class object

hlu_ci
: Integer color index in the range 1 through 255 of a currently allocated color in the given *workid* **Workstation** object.

indx_gks_ret (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 1987,1988,1989,1991,1993,1994,1995
University Corporation for Atmospheric Research

All Rights Reserved

$Revision: 1.9 $ $Date: 1995/07/24 18:25:54 $

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
: Id that identifies the object. Returned from the NhlCreate function.

rlist_id
: Id of a ResList that specifies which resources to retrieve and where put the retrieved values.

ierr (Fortran only)
: Error code.

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 1987,1988,1989,1991,1993,1994,1995
University Corporation for Atmospheric Research
All Rights Reserved

$Revision: 1.12 $ $Date: 1995/07/22 20:03:59 $

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

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 1987,1988,1989,1991,1993,1994,1995
University Corporation for Atmospheric Research

All Rights Reserved

$Revision: 1.7 $ $Date: 1995/07/22 20:03:59 $

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.

See also

- App object

Copyright

Copyright 1987,1988,1989,1991,1993,1994,1995
University Corporation for Atmospheric Research
All Rights Reserved

$Revision: 1.8 $ $Date: 1995/07/22 20:04:08 $

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
: Integer identifier of an instance of a **Workstation** class object

ci
: Integer color index in the range 0 through 255.

isalloc (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 1987,1988,1989,1991,1993,1994,1995
University Corporation for Atmospheric Research
All Rights Reserved

$Revision: 1.7 $ $Date: 1995/07/22 20:04:00 $

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
    Specifies an integer id to an existing HLU object.
istatus (Fortran only)
    On output, the status of the **App** 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 status in *istatus*.

Copyright

Copyright 1987,1988,1989,1991,1993,1994,1995
University Corporation for Atmospheric Research
All Rights Reserved

$Revision: 1.7 $ $Date: 1995/07/22 20:04:00 $

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
Specifies an integer id to an existing HLU object.
istatus (Fortran only)
On output, the status of the **DataComm** 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 1987,1988,1989,1991,1993,1994,1995
University Corporation for Atmospheric Research
All Rights Reserved

$Revision: 1.7 $ $Date: 1995/07/22 20:04:01 $

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
: Specifies an integer id to an existing HLU object.

istatus (Fortran only)
: On output, the status of the **DataItem** 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 1987,1988,1989,1991,1993,1994,1995
University Corporation for Atmospheric Research
All Rights Reserved

$Revision: 1.8 $ $Date: 1995/07/22 20:04:01 $

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
Specifies an integer id to an existing HLU object.
istatus (Fortran only)
On output, the status of the **DataSpec** 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 1987,1988,1989,1991,1993,1994,1995
University Corporation for Atmospheric Research
All Rights Reserved

$Revision: 1.7 $ $Date: 1995/07/22 20:04:02 $

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
Specifies an integer id to an existing HLU object.
istatus (Fortran only)
On output, the status of the **Transform** 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 1987,1988,1989,1991,1993,1994,1995
University Corporation for Atmospheric Research
All Rights Reserved

$Revision: 1.7 $ $Date: 1995/07/22 20:04:03 $

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
    Specifies an integer id to an existing HLU object.
istatus (Fortran only)
    On output, the status of the **View** 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 1987,1988,1989,1991,1993,1994,1995

University Corporation for Atmospheric Research
All Rights Reserved

$Revision: 1.7 $ $Date: 1995/07/22 20:04:03 $

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
: Specifies an integer id to an existing HLU object.

istatus (Fortran only)
: On output, the status of the **Workstation** 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 1987,1988,1989,1991,1993,1994,1995
University Corporation for Atmospheric Research
All Rights Reserved

$Revision: 1.7 $ $Date: 1995/07/22 20:04:04 $

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
: Specifies the size of the memory to allocate.

data
: Specifies a pointer to memory previously returned from NhlMalloc or NhlRealloc.

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 NhlNOERROR if there is no error, or NhlWARNING if there is a detectable error.

Copyright

Copyright 1987,1988,1989,1991,1993,1994,1995
University Corporation for Atmospheric Research
All Rights Reserved

$Revision: 1.7 $ $Date: 1995/07/22 20:04:04 $

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,
                float   *x,
                float   *y,
                int     n,
                )
```

Fortran Synopsis

```
      subroutine NhlFNDCPolyline(pid, x, y, n, ierr)
      integer pid, n, ierr
      float x(n), y(n)
```

Arguments

pid
: Id of a **Transform** class plot object with an established data transformation.

x
: Array of X-Axis coordinates values in NDC space.

y
: Array of Y-Axis coordinates values in NDC space.

n
: Number of elements in the coordinate arrays *x* and *y*.

ierr (Fortran only)
: Error code.

Description

This function draws an immediate mode polyline in NDC space. An *immediate mode* polyline is a polyline that appears as soon as the low-level point buffer is flushed after the function is called. It is not connected with the plot object's **Draw** method.

V4.0 NOTE: There is currently no public interface for setting the line attributes of immediate mode polylines.

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

Copyright

Copyright 1987,1988,1989,1991,1993,1994,1995
University Corporation for Atmospheric Research
All Rights Reserved

$Revision: 1.6 $ $Date: 1995/07/22 20:04:05 $

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, sta
     +                         out_of_range, ierr)
           integer pid, n, ixmissing, iymissing, status, ierr
           real x(n), y(n), xout(n), yout(n), xmissing, ymissing, o
```

Arguments

pid

Id of a **Transform** class plot object with an established data transformation.

x

Array of X-Axis coordinates values in NDC space.

y

Array of Y-Axis coordinates values in NDC space.

n

Number of elements in each of the arrays *x*, *y*, *xout* and *yout*.

xout

Array for X-Axis output values in data coordinate space. The user is

responsible for providing the appropriate amount of storage.

yout
Array for Y-Axis output values in data coordinate space. The user is responsible for providing the appropriate amount of storage.

xmissing
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
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 (Fortran only)
Flag for whether or not any missing values exist for the X-Axis input array; if 1, then missing values exists.

iymissing (Fortran only)
Flag for whether or not any missing values exist for the Y-Axis input array; if 1, then missing value exists..

status
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
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 (Fortran only)
Error code.

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 1987,1988,1989,1991,1993,1994,1995
University Corporation for Atmospheric Research
All Rights Reserved

$Revision: 1.7 $ $Date: 1995/07/22 20:04:05 $

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
: Specifies the object identifier.

name_ret (Fortran only)
: The name of the object specified by the *id* argument.

ierr (Fortran only)
: Error code.

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, and the **NhlFName** Fortran subroutine returns the error in *ierr*.

See also

- NhlCreate

Copyright

Copyright 1987,1988,1989,1991,1993,1994,1995
University Corporation for Atmospheric Research
All Rights Reserved

$Revision: 1.7 $ $Date: 1995/07/22 20:04:06 $

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
: Integer identifier of an instance of a **Workstation** class object

red
: Value of the red component of the desired color in the range 0.0 to 1.0 inclusive.

green
: Value of the green component of the desired color in the range 0.0 to 1.0

inclusive.

blue
: Value of the blue component of the desired color in the range 0.0 to 1.0 inclusive.

indx (Fortran only)
: The HLU color index of a newly allocated color with the specified RGB values.

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 1987,1988,1989,1991,1993,1994,1995
University Corporation for Atmospheric Research
All Rights Reserved

$Revision: 1.7 $ $Date: 1995/07/22 20:04:07 $

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
: 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 1987,1988,1989,1991,1993,1994,1995
University Corporation for Atmospheric Research
All Rights Reserved

$Revision: 1.6 $ $Date: 1995/07/22 20:04:07 $

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.

See also

- App object

Copyright

Copyright 1987,1988,1989,1991,1993,1994,1995
University Corporation for Atmospheric Research
All Rights Reserved

$Revision: 1.8 $ $Date: 1995/07/22 20:04:08 $

NhlPError

NhlPError, NHLPERROR, NhlErrGetId, NhlErrNumMsgs, NhlErrGetMsg, NhlErrCleanMsgs, NhlErrAddTable, NhlErrSPrintMsg, NhlErrFPrintMsg

The Fortran names of these functions are **NhlFPError**, **NhlFErrGetId**, **NhlFErrNumMsgs**, **NhlFErrGetMsg**, **NhlFErrCleanMsgs**, **NhlFErrSpringMsg**, and **NhlFErrFPrintMsg**. There are no Fortran bindings for **NHLPEROR** 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, format)
      character*(*) severity, format
      integer errnum

      subroutine NhlFErrGetId(id_ret)
      integer id_ret

      subroutine NhlFErrNumMsgs(nummsg)
      integer nummsg

      subroutine NhlFErrGetMsg(msgnum, severity, emsg, errorno, sy
     +                         line, fname, ierr)
      integer msgnum, severity, 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
: Specifies the severity level of the error message being reported.

errnum
: Specifies a unique error number that references an error message from an installed error table.

format
: Format string like printf.

...
: Arguments for format string aka printf.

msgnum
: Identifies the index of the error message to retrieve.

msgptr (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.

msg (C only)
: Pointer to an NhlErrMsg structure that fully describes an error message.

start (C only)
: Index to assign to the first error message in the table being installed.

tlen (C only)
: Length of the error table being installed.

etable (C only)
: Pointer to an error table to install.

buffer
: Character buffer to place the formatted message into.

fp (C only)
: File pointer to print formatted error message to.

nummsg (Fortran only)
: Number of error messages currently buffered.

id_ret (Fortran only)
: Id for *Error object*.

iunit (Fortran only)
: Unit number to print formatted error message to.

emsg (Fortran only)
: Actual text of the error message.

errorno (Fortran only)
: Error number.

sysmsg (Fortran only)
: Actual text in the error table indexed by *errorno*.

line (Fortran only)
: Line number the error message was reported from, or 0.

fname (Fortran only)
File name the error message was reported from, or NULL.
ierr (Fortran only)
Error code.

Types

```
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 1987,1988,1989,1991,1993,1994,1995
University Corporation for Atmospheric Research
All Rights Reserved

$Revision: 1.12 $ $Date: 1995/07/22 20:04:08 $

NhlPError

NhlPError, NHLPERROR, NhlErrGetId, NhlErrNumMsgs, NhlErrGetMsg, NhlErrCleanMsgs, NhlErrAddTable, NhlErrSPrintMsg, NhlErrFPrintMsg

The Fortran names of these functions are **NhlFPError**, **NhlFErrGetId**, **NhlFErrNumMsgs**, **NhlFErrGetMsg**, **NhlFErrCleanMsgs**, **NhlFErrSpringMsg**, and **NhlFErrFPrintMsg**. There are no Fortran bindings for **NHLPEROR** 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, format)
      character*(*) severity, format
      integer errnum

      subroutine NhlFErrGetId(id_ret)
      integer id_ret

      subroutine NhlFErrNumMsgs(nummsg)
      integer nummsg

      subroutine NhlFErrGetMsg(msgnum, severity, emsg, errorno, sy
     +                         line, fname, ierr)
      integer msgnum, severity, 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
: Specifies the severity level of the error message being reported.

errnum
: Specifies a unique error number that references an error message from an installed error table.

format
: Format string like printf.

...
: Arguments for format string aka printf.

msgnum
: Identifies the index of the error message to retrieve.

msgptr (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.

msg (C only)
: Pointer to an NhlErrMsg structure that fully describes an error message.

start (C only)
: Index to assign to the first error message in the table being installed.

tlen (C only)
: Length of the error table being installed.

etable (C only)
: Pointer to an error table to install.

buffer
: Character buffer to place the formatted message into.

fp (C only)
: File pointer to print formatted error message to.

nummsg (Fortran only)
: Number of error messages currently buffered.

id_ret (Fortran only)
: Id for *Error object*.

iunit (Fortran only)
: Unit number to print formatted error message to.

emsg (Fortran only)
: Actual text of the error message.

errorno (Fortran only)
: Error number.

sysmsg (Fortran only)
Actual text in the error table indexed by *errorno*.
line (Fortran only)
Line number the error message was reported from, or 0.
fname (Fortran only)
File name the error message was reported from, or NULL.
ierr (Fortran only)
Error code.

Types

```
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 1987,1988,1989,1991,1993,1994,1995
University Corporation for Atmospheric Research
All Rights Reserved

$Revision: 1.12 $ $Date: 1995/07/22 20:04:08 $

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
: Specifies the type of ResList to create. Use NhlSETRL for a list being used to set values in an object (**NhlCreate** and **NhlSetValues**). Use NhlGETRL for a list being used to retrieve values from an object. (**NhlGetValues**)

list_id
: Integer identifier for the *ResList*. It is returned from the **NhlRLCreate** function.

resname
: 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 (Fortran only)
: Flag indicating whether or not a resource specified by *resname* is in the *ResList*.

Types

```
typedef enum NhlRLType_{
        NhlSETRL,
        NhlGETRL
} 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 1987,1988,1989,1991,1993,1994,1995
University Corporation for Atmospheric Research
All Rights Reserved

$Revision: 1.11 $ $Date: 1995/07/22 20:04:09 $

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
: Specifies the type of ResList to create. Use NhlSETRL for a list being used to set values in an object (**NhlCreate** and **NhlSetValues**). Use NhlGETRL for a list being used to retrieve values from an object. (**NhlGetValues**)

list_id
: Integer identifier for the *ResList*. It is returned from the **NhlRLCreate** function.

resname
: 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 (Fortran only)
: Flag indicating whether or not a resource specified by *resname* is in the *ResList*.

Types

```
typedef enum NhlRLType_{
        NhlSETRL,
        NhlGETRL
} 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 1987,1988,1989,1991,1993,1994,1995
University Corporation for Atmospheric Research
All Rights Reserved

$Revision: 1.11 $ $Date: 1995/07/22 20:04:09 $

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
: Specifies the type of ResList to create. Use NhlSETRL for a list being used to set values in an object (**NhlCreate** and **NhlSetValues**). Use NhlGETRL for a list being used to retrieve values from an object. (**NhlGetValues**)

list_id
: Integer identifier for the *ResList*. It is returned from the **NhlRLCreate** function.

resname
: 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 (Fortran only)
: Flag indicating whether or not a resource specified by *resname* is in the *ResList*.

Types

```
typedef enum NhlRLType_{
        NhlSETRL,
        NhlGETRL
} 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 1987,1988,1989,1991,1993,1994,1995
University Corporation for Atmospheric Research
All Rights Reserved

$Revision: 1.11 $ $Date: 1995/07/22 20:04:09 $

NhlRLGet

NhlRLGet, NhlRLGetInteger, NhlRLGetFloat, NhlRLGetString

The Fortran names of these functions are **NhlFRLGetInteger**,

NhlFRLGetFloat, **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 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 NhlFRLGetString(list_id, resname, sval, ierr)
      integer list_id, ierr
      character*(*) resname, sval
```

Arguments

list_id
: Id of the *ResList* to retrieve the *resname/val_addr* pair from.

resname
: Specifies the resource name to retrieve the value of.

type (C only)
: Specifies the type *val_addr* points to.

val_addr (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 (Fortran only)
: The integer value retrieved from the *resname* resource when the list is applied to an HLU object using the **NhlFGetValues** function.

rval (Fortran only)
: The real value retrieved from the *resname* resource when the list is applied to an HLU object using the **NhlFGetValues** function.

sval (Fortran only)
: The string retrieved from the *resname* resource when the list is applied to an HLU object using the **NhlFGetValues** function.

ierr (Fortran only)
: Error code.

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 1987,1988,1989,1991,1993,1994,1995
University Corporation for Atmospheric Research
All Rights Reserved

$Revision: 1.9 $ $Date: 1995/07/22 20:04:09 $

NhlRLGetArray

NhlRLGetArray, NhlRLGetIntegerArray, NhlRLGetFloatArray, NhlRLGetStringArray

The Fortran names of these functions are **NhlFRLGetIntegerArray**, **NhlFRLGetFloatArray**, **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 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 NhlFRLGetStringArray(list_id, resname, sarray,
     +                                num_elements, ierr)
      integer list_id, num_elements, ierr
      character*(*) resname, sarray(num_elements)
```

Arguments

list_id
: Id of the *ResList* to retrieve the *name/val_addr* pair from.

resname
: Specifies the resource name to retrieve the value of.

data_addr (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 (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 (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
: 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 (Fortran only)
: The integer array retrieved from the *resname* resource when the list is

applied to an HLU object using the **NhlFGetValues** function.

rarray (Fortran only)
The real array retrieved from the *resname* resource when the list is applied to an HLU object using the **NhlFGetValues** function.

sarray (Fortran only)
The string array retrieved from the *resname* resource when the list is applied to an HLU object using the **NhlFGetValues** function.

ierr (Fortran only)
Error code.

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 1987,1988,1989,1991,1993,1994,1995
University Corporation for Atmospheric Research
All Rights Reserved

$Revision: 1.10 $ $Date: 1995/07/22 20:04:10 $

NhlRLGet

NhlRLGet, NhlRLGetInteger, NhlRLGetFloat, NhlRLGetString

The Fortran names of these functions are **NhlFRLGetInteger**, **NhlFRLGetFloat**, **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 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 NhlFRLGetString(list_id, resname, sval, ierr)
```

```
integer list_id, ierr
character*(*) resname, sval
```

Arguments

list_id
: Id of the *ResList* to retrieve the *resname/val_addr* pair from.

resname
: Specifies the resource name to retrieve the value of.

type (C only)
: Specifies the type *val_addr* points to.

val_addr (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 (Fortran only)
: The integer value retrieved from the *resname* resource when the list is applied to an HLU object using the **NhlFGetValues** function.

rval (Fortran only)
: The real value retrieved from the *resname* resource when the list is applied to an HLU object using the **NhlFGetValues** function.

sval (Fortran only)
: The string retrieved from the *resname* resource when the list is applied to an HLU object using the **NhlFGetValues** function.

ierr (Fortran only)
: Error code.

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 1987,1988,1989,1991,1993,1994,1995
University Corporation for Atmospheric Research
All Rights Reserved

$Revision: 1.9 $ $Date: 1995/07/22 20:04:09 $

NhlRLGetArray

NhlRLGetArray, NhlRLGetIntegerArray, NhlRLGetFloatArray, NhlRLGetStringArray

The Fortran names of these functions are **NhlFRLGetIntegerArray**, **NhlFRLGetFloatArray**, **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 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 NhlFRLGetStringArray(list_id, resname, sarray,
     +                                num_elements, ierr)
      integer list_id, num_elements, ierr
      character*(*) resname, sarray(num_elements)
```

Arguments

list_id
: Id of the *ResList* to retrieve the *name/val_addr* pair from.

resname
: Specifies the resource name to retrieve the value of.

data_addr (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 (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 (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

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 (Fortran only)
: The integer array retrieved from the *resname* resource when the list is applied to an HLU object using the **NhlFGetValues** function.

rarray (Fortran only)
: The real array retrieved from the *resname* resource when the list is applied to an HLU object using the **NhlFGetValues** function.

sarray (Fortran only)
: The string array retrieved from the *resname* resource when the list is applied to an HLU object using the **NhlFGetValues** function.

ierr (Fortran only)
: Error code.

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 1987,1988,1989,1991,1993,1994,1995
University Corporation for Atmospheric Research
All Rights Reserved

$Revision: 1.10 $ $Date: 1995/07/22 20:04:10 $

NhlRLGet

NhlRLGet, NhlRLGetInteger, NhlRLGetFloat, NhlRLGetString

The Fortran names of these functions are **NhlFRLGetInteger**, **NhlFRLGetFloat**, **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 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 NhlFRLGetString(list_id, resname, sval, ierr)
integer list_id, ierr
character*(*) resname, sval
```

Arguments

list_id
: Id of the *ResList* to retrieve the *resname/val_addr* pair from.

resname
: Specifies the resource name to retrieve the value of.

type (C only)
: Specifies the type *val_addr* points to.

val_addr (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 (Fortran only)
: The integer value retrieved from the *resname* resource when the list is applied to an HLU object using the **NhlFGetValues** function.

rval (Fortran only)
: The real value retrieved from the *resname* resource when the list is applied to an HLU object using the **NhlFGetValues** function.

sval (Fortran only)
: The string retrieved from the *resname* resource when the list is applied to an HLU object using the **NhlFGetValues** function.

ierr (Fortran only)
: Error code.

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 1987,1988,1989,1991,1993,1994,1995
University Corporation for Atmospheric Research
All Rights Reserved

$Revision: 1.9 $ $Date: 1995/07/22 20:04:09 $

NhlRLGetArray

NhlRLGetArray, NhlRLGetIntegerArray, NhlRLGetFloatArray, NhlRLGetStringArray

The Fortran names of these functions are **NhlFRLGetIntegerArray**, **NhlFRLGetFloatArray**, **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 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 NhlFRLGetStringArray(list_id, resname, sarray,
     +                                num_elements, ierr)
      integer list_id, num_elements, ierr
      character*(*) resname, sarray(num_elements)
```

Arguments

list_id
: Id of the *ResList* to retrieve the *name/val_addr* pair from.

resname
: Specifies the resource name to retrieve the value of.

data_addr (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 (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 (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
: 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 (Fortran only)
: The integer array retrieved from the *resname* resource when the list is applied to an HLU object using the **NhlFGetValues** function.

rarray (Fortran only)
: The real array retrieved from the *resname* resource when the list is applied to an HLU object using the **NhlFGetValues** function.

sarray (Fortran only)
: The string array retrieved from the *resname* resource when the list is applied to an HLU object using the **NhlFGetValues** function.

ierr (Fortran only)
: Error code.

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 1987,1988,1989,1991,1993,1994,1995
University Corporation for Atmospheric Research
All Rights Reserved

$Revision: 1.10 $ $Date: 1995/07/22 20:04:10 $

NhlRLGetMDArray

NhlRLGetMDArray, NhlRLGetMDIntegerArray, NhlRLGetMDFloatArray

The Fortran names of these functions are **NhlFRLGetMDIntegerArray**, **NhlFRLGetMDFloatArray**. 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
)
```

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
```

Arguments

list_id
: Id of the *ResList* to retrieve the *name/val_addr* pair from.

resname
: Specifies the resource name to retrieve the value of.

data_addr (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 (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 (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 (C only)
: Returns the number of dimensions in the *data* array. This parameter must be a valid pointer to an integer.

len_dimensions (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 (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 (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 (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 (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.

ierr (Fortran only)
: Error code.

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 1987,1988,1989,1991,1993,1994,1995
University Corporation for Atmospheric Research
All Rights Reserved

$Revision: 1.10 $ $Date: 1995/07/22 20:04:10 $

NhlRLGetMDArray

NhlRLGetMDArray, NhlRLGetMDIntegerArray, NhlRLGetMDFloatArray

The Fortran names of these functions are **NhlFRLGetMDIntegerArray**, **NhlFRLGetMDFloatArray**. 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
)
```

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
```

Arguments

list_id
: Id of the *ResList* to retrieve the *name/val_addr* pair from.

resname
: Specifies the resource name to retrieve the value of.

data_addr (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 (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 (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 (C only)
: Returns the number of dimensions in the *data* array. This parameter must be a valid pointer to an integer.

len_dimensions (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 (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 (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 (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 (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.

ierr (Fortran only)
: Error code.

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 1987,1988,1989,1991,1993,1994,1995
University Corporation for Atmospheric Research
All Rights Reserved

$Revision: 1.10 $ $Date: 1995/07/22 20:04:10 $

NhlRLGetMDArray

NhlRLGetMDArray, NhlRLGetMDIntegerArray, NhlRLGetMDFloatArray

The Fortran names of these functions are **NhlFRLGetMDIntegerArray**, **NhlFRLGetMDFloatArray**. 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
)
```

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
```

Arguments

list_id
: Id of the *ResList* to retrieve the *name/val_addr* pair from.

resname
: Specifies the resource name to retrieve the value of.

data_addr (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 (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 (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 (C only)
: Returns the number of dimensions in the *data* array. This parameter must be a valid pointer to an integer.

len_dimensions (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 (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 (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 (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 (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.

ierr (Fortran only)
: Error code.

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 1987,1988,1989,1991,1993,1994,1995
University Corporation for Atmospheric Research
All Rights Reserved

$Revision: 1.10 $ $Date: 1995/07/22 20:04:10 $

NhlRLGet

NhlRLGet, NhlRLGetInteger, NhlRLGetFloat, NhlRLGetString

The Fortran names of these functions are **NhlFRLGetInteger**, **NhlFRLGetFloat**, **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 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 NhlFRLGetString(list_id, resname, sval, ierr)
integer list_id, ierr
character*(*) resname, sval
```

Arguments

list_id
: Id of the *ResList* to retrieve the *resname/val_addr* pair from.

resname
: Specifies the resource name to retrieve the value of.

type (C only)
: Specifies the type *val_addr* points to.

val_addr (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 (Fortran only)
: The integer value retrieved from the *resname* resource when the list is applied to an HLU object using the **NhlFGetValues** function.

rval (Fortran only)
: The real value retrieved from the *resname* resource when the list is applied to an HLU object using the **NhlFGetValues** function.

sval (Fortran only)
: The string retrieved from the *resname* resource when the list is applied to an HLU object using the **NhlFGetValues** function.

ierr (Fortran only)
: Error code.

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 1987,1988,1989,1991,1993,1994,1995
University Corporation for Atmospheric Research
All Rights Reserved

$Revision: 1.9 $ $Date: 1995/07/22 20:04:09 $

NhlRLGetArray

NhlRLGetArray, NhlRLGetIntegerArray, NhlRLGetFloatArray, NhlRLGetStringArray

The Fortran names of these functions are **NhlFRLGetIntegerArray**, **NhlFRLGetFloatArray**, **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 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 NhlFRLGetStringArray(list_id, resname, sarray,
     +                                num_elements, ierr)
      integer list_id, num_elements, ierr
      character*(*) resname, sarray(num_elements)
```

Arguments

list_id

Id of the *ResList* to retrieve the *name/val_addr* pair from.

resname

Specifies the resource name to retrieve the value of.

data_addr (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 (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 (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

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 (Fortran only)

The integer array retrieved from the *resname* resource when the list is applied to an HLU object using the **NhlFGetValues** function.

rarray (Fortran only)

The real array retrieved from the *resname* resource when the list is applied to an HLU object using the **NhlFGetValues** function.

sarray (Fortran only)

The string array retrieved from the *resname* resource when the list is applied to an HLU object using the **NhlFGetValues** function.

ierr (Fortran only)

Error code.

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 1987,1988,1989,1991,1993,1994,1995
University Corporation for Atmospheric Research
All Rights Reserved

$Revision: 1.10 $ $Date: 1995/07/22 20:04:10 $

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
: Specifies the type of ResList to create. Use NhlSETRL for a list being used to set values in an object (**NhlCreate** and **NhlSetValues**). Use NhlGETRL for a list being used to retrieve values from an object. (**NhlGetValues**)

list_id
: Integer identifier for the *ResList*. It is returned from the **NhlRLCreate** function.

resname
: 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 (Fortran only)
: Flag indicating whether or not a resource specified by *resname* is in the *ResList*.

Types

```
typedef enum NhlRLType_{
        NhlSETRL,
        NhlGETRL
} 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 1987,1988,1989,1991,1993,1994,1995
University Corporation for Atmospheric Research
All Rights Reserved

$Revision: 1.11 $ $Date: 1995/07/22 20:04:09 $

NhlRLSet

NhlRLSet, NhlRLSetInteger, NhlRLSetFloat, NhlRLSetString

The Fortran names of these functions are **NhlFRLSetInteger**, **NhlFRLSetFloat**, **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 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 NhlFRLSetString(list_id, resname, sval, ierr)
integer list_id, ierr
character*(*) resname, sval
```

Arguments

list_id
: Id of the *ResList* to add the *name/value* pair to.

resname
: Specifies the resource name to apply the given *value* to when the list is applied to an HLU object.

type (C only)
: Specifies the type for *value*.

value (C only)
: Specifies the value that will be applied to the corresponding *resname* when the list is applied to an HLU object.

ival (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 (Fortran only)
: Specifies the real value that will be applied to the corresponding *resname* resource when the list is applied to an HLU object.

sval (Fortran only)
: Specifies the string that will be applied to the corresponding *resname* resource when the list is applied to an HLU object.

ierr (Fortran only)
: Error code.

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 1987,1988,1989,1991,1993,1994,1995
University Corporation for Atmospheric Research
All Rights Reserved

$Revision: 1.8 $ $Date: 1995/07/22 20:04:11 $

NhlRLSetArray

NhlRLSetArray, NhlRLSetIntegerArray, NhlRLSetFloatArray, NhlRLSetStringArray

The Fortran names of these functions are **NhlFRLSetIntegerArray**, **NhlFRLSetFloatArray**, **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 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 NhlFRLSetStringArray(list_id, resname, sarray,
     +                                num_elements, ierr)
      integer list_id, num_elements, ierr
      character*(*) resname, sarray(num_elements)
```

Arguments

list_id
: Id of the *ResList* to add the *name/value* pair to.

resname
: Specifies the resource name to apply the given *data* to when the list is applied to an HLU object.

data (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 (C only)
: Specifies the type of each element of the *data* array.

size_elements (C only)

Specifies the size in bytes of each element of the *data* array.

num_elements
Specifies the number of elements in the *data* array.

iarray (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 (Fortran only)
Specifies the real array that will be applied to the corresponding *resname* resource when the list is applied to an HLU object.

sarray (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 (Fortran only)
Error code.

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 1987,1988,1989,1991,1993,1994,1995
University Corporation for Atmospheric Research
All Rights Reserved

$Revision: 1.9 $ $Date: 1995/07/22 20:04:11 $

NhlRLSet

NhlRLSet, NhlRLSetInteger, NhlRLSetFloat, NhlRLSetString

The Fortran names of these functions are **NhlFRLSetInteger**, **NhlFRLSetFloat**, **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 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 NhlFRLSetString(list_id, resname, sval, ierr)
integer list_id, ierr
character*(*) resname, sval
```

Arguments

list_id
: Id of the *ResList* to add the *name/value* pair to.

resname
: Specifies the resource name to apply the given *value* to when the list is applied to an HLU object.

type (C only)
: Specifies the type for *value*.

value (C only)
: Specifies the value that will be applied to the corresponding *resname* when the list is applied to an HLU object.

ival (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 (Fortran only)
: Specifies the real value that will be applied to the corresponding *resname* resource when the list is applied to an HLU object.

sval (Fortran only)
: Specifies the string that will be applied to the corresponding *resname* resource when the list is applied to an HLU object.

ierr (Fortran only)
: Error code.

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 1987,1988,1989,1991,1993,1994,1995
University Corporation for Atmospheric Research
All Rights Reserved

$Revision: 1.8 $ $Date: 1995/07/22 20:04:11 $

NhlRLSetArray

NhlRLSetArray, NhlRLSetIntegerArray, NhlRLSetFloatArray, NhlRLSetStringArray

The Fortran names of these functions are **NhlFRLSetIntegerArray**, **NhlFRLSetFloatArray**, **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 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 NhlFRLSetStringArray(list_id, resname, sarray,
     +                                num_elements, ierr)
      integer list_id, num_elements, ierr
      character*(*) resname, sarray(num_elements)
```

Arguments

list_id
: Id of the *ResList* to add the *name/value* pair to.

resname
: Specifies the resource name to apply the given *data* to when the list is applied to an HLU object.

data (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 (C only)
: Specifies the type of each element of the *data* array.

size_elements (C only)
: Specifies the size in bytes of each element of the *data* array.

num_elements
: Specifies the number of elements in the *data* array.

iarray (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 (Fortran only)
: Specifies the real array that will be applied to the corresponding *resname* resource when the list is applied to an HLU object.

sarray (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 (Fortran only)
: Error code.

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 1987,1988,1989,1991,1993,1994,1995
University Corporation for Atmospheric Research
All Rights Reserved

$Revision: 1.9 $ $Date: 1995/07/22 20:04:11 $

NhlRLSet

NhlRLSet, NhlRLSetInteger, NhlRLSetFloat, NhlRLSetString

The Fortran names of these functions are **NhlFRLSetInteger**, **NhlFRLSetFloat**, **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 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 NhlFRLSetString(list_id, resname, sval, ierr)
integer list_id, ierr
character*(*) resname, sval
```

Arguments

list_id
: Id of the *ResList* to add the *name/value* pair to.

resname
: Specifies the resource name to apply the given *value* to when the list is applied to an HLU object.

type (C only)
: Specifies the type for *value*.

value (C only)
: Specifies the value that will be applied to the corresponding *resname* when the list is applied to an HLU object.

ival (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 (Fortran only)
: Specifies the real value that will be applied to the corresponding *resname* resource when the list is applied to an HLU object.

sval (Fortran only)
: Specifies the string that will be applied to the corresponding *resname* resource when the list is applied to an HLU object.

ierr (Fortran only)
: Error code.

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 1987,1988,1989,1991,1993,1994,1995
University Corporation for Atmospheric Research
All Rights Reserved

$Revision: 1.8 $ $Date: 1995/07/22 20:04:11 $

NhlRLSetArray

NhlRLSetArray, NhlRLSetIntegerArray, NhlRLSetFloatArray, NhlRLSetStringArray

The Fortran names of these functions are **NhlFRLSetIntegerArray**, **NhlFRLSetFloatArray**, **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 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 NhlFRLSetStringArray(list_id, resname, sarray,
     +                                num_elements, ierr)
      integer list_id, num_elements, ierr
      character*(*) resname, sarray(num_elements)
```

Arguments

list_id
: Id of the *ResList* to add the *name/value* pair to.

resname
: Specifies the resource name to apply the given *data* to when the list is applied to an HLU object.

data (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 (C only)
: Specifies the type of each element of the *data* array.

size_elements (C only)
: Specifies the size in bytes of each element of the *data* array.

num_elements
: Specifies the number of elements in the *data* array.

iarray (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 (Fortran only)
Specifies the real array that will be applied to the corresponding *resname* resource when the list is applied to an HLU object.

sarray (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 (Fortran only)
Error code.

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 1987,1988,1989,1991,1993,1994,1995
University Corporation for Atmospheric Research
All Rights Reserved

$Revision: 1.9 $ $Date: 1995/07/22 20:04:11 $

NhlRLSetMDArray

NhlRLSetMDArray, NhlRLSetMDIntegerArray, NhlRLSetMDFloatArray

The Fortran names of these functions are **NhlFRLSetMDIntegerArray**, **NhlFRLSetMDFloatArray**. 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
                )
```

Fortran synopsis

```
      subroutine NhlFRLSetMDIntegerArray(list_id, resname, iarray,
     +                   num_dimensions, len_dimensions, ierr)
      integer list_id, iarray(*), num_dimensions, len_dimensions,
      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
```

Arguments

list_id
: Id of the *ResList* to add the *name/value* pair to.

resname
: Specifies the resource name to apply the given *data* to when the list is applied to an HLU object.

data (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 (C only)
: Specifies the type of each element of the *data* array.

size_elements (C only)
: Specifies the size in bytes of each element of the *data* array.

num_dimensions
: Specifies the number of dimensions in the *data* array.

len_dimensions
: An integer array of length *num_dimensions* that specifies the length of each dimensions in the *data* array.

iarray (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 (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.

ierr (Fortran only)
: Error code.

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 1987,1988,1989,1991,1993,1994,1995
University Corporation for Atmospheric Research
All Rights Reserved

$Revision: 1.9 $ $Date: 1995/07/22 20:04:12 $

NhlRLSetMDArray

NhlRLSetMDArray, NhlRLSetMDIntegerArray, NhlRLSetMDFloatArray

The Fortran names of these functions are **NhlFRLSetMDIntegerArray**, **NhlFRLSetMDFloatArray**. 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
                )
```

Fortran synopsis

```
      subroutine NhlFRLSetMDIntegerArray(list_id, resname, iarray,
     +                   num_dimensions, len_dimensions, ierr)
      integer list_id, iarray(*), num_dimensions, len_dimensions,
      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
```

Arguments

list_id
: Id of the *ResList* to add the *name/value* pair to.

resname
: Specifies the resource name to apply the given *data* to when the list is applied to an HLU object.

data (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 (C only)
: Specifies the type of each element of the *data* array.

size_elements (C only)
: Specifies the size in bytes of each element of the *data* array.

num_dimensions
: Specifies the number of dimensions in the *data* array.

len_dimensions
: An integer array of length *num_dimensions* that specifies the length of each dimensions in the *data* array.

iarray (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 (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.
ierr (Fortran only)
Error code.

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 1987,1988,1989,1991,1993,1994,1995
University Corporation for Atmospheric Research
All Rights Reserved

$Revision: 1.9 $ $Date: 1995/07/22 20:04:12 $

NhlRLSetMDArray

NhlRLSetMDArray, NhlRLSetMDIntegerArray, NhlRLSetMDFloatArray

The Fortran names of these functions are **NhlFRLSetMDIntegerArray**, **NhlFRLSetMDFloatArray**. 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
                )
```

Fortran synopsis

```
      subroutine NhlFRLSetMDIntegerArray(list_id, resname, iarray,
     +                   num_dimensions, len_dimensions, ierr)
      integer list_id, iarray(*), num_dimensions, len_dimensions,
      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
```

Arguments

list_id
: Id of the *ResList* to add the *name/value* pair to.

resname
: Specifies the resource name to apply the given *data* to when the list is applied to an HLU object.

data (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 (C only)
: Specifies the type of each element of the *data* array.

size_elements (C only)
: Specifies the size in bytes of each element of the *data* array.

num_dimensions
: Specifies the number of dimensions in the *data* array.

len_dimensions
: An integer array of length *num_dimensions* that specifies the length of each dimensions in the *data* array.

iarray (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 (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.

ierr (Fortran only)
: Error code.

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 1987,1988,1989,1991,1993,1994,1995
University Corporation for Atmospheric Research
All Rights Reserved

$Revision: 1.9 $ $Date: 1995/07/22 20:04:12 $

NhlRLSet

NhlRLSet, NhlRLSetInteger, NhlRLSetFloat, NhlRLSetString

The Fortran names of these functions are **NhlFRLSetInteger**, **NhlFRLSetFloat**, **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 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 NhlFRLSetString(list_id, resname, sval, ierr)
      integer list_id, ierr
      character*(*) resname, sval
```

Arguments

list_id
: Id of the *ResList* to add the *name/value* pair to.

resname
: Specifies the resource name to apply the given *value* to when the list is applied to an HLU object.

type (C only)
: Specifies the type for *value*.

value (C only)
: Specifies the value that will be applied to the corresponding *resname* when the list is applied to an HLU object.

ival (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 (Fortran only)
: Specifies the real value that will be applied to the corresponding *resname* resource when the list is applied to an HLU object.

sval (Fortran only)
: Specifies the string that will be applied to the corresponding *resname* resource when the list is applied to an HLU object.

ierr (Fortran only)

Error code.

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 1987,1988,1989,1991,1993,1994,1995
University Corporation for Atmospheric Research
All Rights Reserved

$Revision: 1.8 $ $Date: 1995/07/22 20:04:11 $

NhlRLSetArray

NhlRLSetArray, NhlRLSetIntegerArray, NhlRLSetFloatArray, NhlRLSetStringArray

The Fortran names of these functions are **NhlFRLSetIntegerArray**, **NhlFRLSetFloatArray**, **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 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 NhlFRLSetStringArray(list_id, resname, sarray,
     +                                num_elements, ierr)
      integer list_id, num_elements, ierr
      character*(*) resname, sarray(num_elements)
```

Arguments

list_id
: Id of the *ResList* to add the *name/value* pair to.

resname
: Specifies the resource name to apply the given *data* to when the list is applied to an HLU object.

data (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 (C only)
: Specifies the type of each element of the *data* array.

size_elements (C only)
: Specifies the size in bytes of each element of the *data* array.

num_elements
: Specifies the number of elements in the *data* array.

iarray (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 (Fortran only)
: Specifies the real array that will be applied to the corresponding *resname* resource when the list is applied to an HLU object.

sarray (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 (Fortran only)
: Error code.

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 1987,1988,1989,1991,1993,1994,1995
University Corporation for Atmospheric Research
All Rights Reserved

$Revision: 1.9 $ $Date: 1995/07/22 20:04:11 $

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
: Specifies the type of ResList to create. Use NhlSETRL for a list being used to set values in an object (**NhlCreate** and **NhlSetValues**). Use NhlGETRL for a list being used to retrieve values from an object. (**NhlGetValues**)

list_id
: Integer identifier for the *ResList*. It is returned from the **NhlRLCreate** function.

resname
: 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 (Fortran only)
: Flag indicating whether or not a resource specified by *resname* is in the *ResList*.

Types

```
typedef enum NhlRLType_{
        NhlSETRL,
        NhlGETRL
} 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 1987,1988,1989,1991,1993,1994,1995
University Corporation for Atmospheric Research
All Rights Reserved

$Revision: 1.11 $ $Date: 1995/07/22 20:04:09 $

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
: Specifies the size of the memory to allocate.

data
: Specifies a pointer to memory previously returned from NhlMalloc or NhlRealloc.

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 NhlNOERROR if there is no error, or NhlWARNING if there is a detectable error.

Copyright

Copyright 1987,1988,1989,1991,1993,1994,1995
University Corporation for Atmospheric Research
All Rights Reserved

$Revision: 1.7 $ $Date: 1995/07/22 20:04:04 $

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, ie
      integer plot_id, anno_manager_id, ierr
```

Arguments

plot_id
: Identifier of the plot object from which the **View** object annotation is to be removed.

anno_manager_id
: Identifier of the **AnnoManager** for the **View** object annotation to be removed.

ierr (Fortran only)
: Error code.

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 1987,1988,1989,1991,1993,1994,1995
University Corporation for Atmospheric Research
All Rights Reserved

$Revision: 1.10 $ $Date: 1995/07/22 20:04:13 $

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
: Identifies the **DataComm** object with the data resource.

res_name
: Identifies the data resource to remove the *ditemid* from in the **DataComm** object.

ditemid
: Identifies the id of the **DataItem** or **DataSpec** object to dissociate from the *res_name* resource in the *dcommid* **DataComm** object.

ierr (Fortran only)
: Error code.

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 1987,1988,1989,1991,1993,1994,1995
University Corporation for Atmospheric Research
All Rights Reserved

$Revision: 1.9 $ $Date: 1995/07/22 20:04:13 $

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, i
integer base_id, overlay_id, restore, ierr
```

Arguments

base_id
: Identifies the base plot from which the overlay is to be removed.

overlay_id
: 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
: 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 (Fortran only)
: Error code.

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 1987,1988,1989,1991,1993,1994,1995
University Corporation for Atmospheric Research
All Rights Reserved

$Revision: 1.7 $ $Date: 1995/07/22 20:04:14 $

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
: Integer identifier of an instance of a **Workstation** class object

ci
: Integer color index in the range 1 through 255 that is to be allocated.

red
: Value of the red component of the desired color in the range 0.0 to 1.0 inclusive.

green
: Value of the green component of the desired color in the range 0.0 to 1.0 inclusive.

blue
: Value of the blue component of the desired color in the range 0.0 to 1.0 inclusive.

ierr (Fortran only)
: Error code.

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 1987,1988,1989,1991,1993,1994,1995
University Corporation for Atmospheric Research
All Rights Reserved

$Revision: 1.6 $ $Date: 1995/07/22 20:04:14 $

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
: Id that identifies the object. Returned from the NhlCreate function.

rlist_id
: Id of a ResList that specifies what resources to modify and what values to modify them to.

ierr (Fortran only)
: Error code.

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 1987,1988,1989,1991,1993,1994,1995
University Corporation for Atmospheric Research
All Rights Reserved

$Revision: 1.13 $ $Date: 1995/07/22 20:04:15 $

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
Identifies the **DataComm** object to update.
ierr (Fortran only)
Error code.

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 1987,1988,1989,1991,1993,1994,1995
University Corporation for Atmospheric Research
All Rights Reserved

$Revision: 1.11 $ $Date: 1995/07/22 20:04:16 $

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
Integer identifier of an instance of a **Workstation** class object
ierr (Fortran only)
Error code.

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 1987,1988,1989,1991,1993,1994,1995
University Corporation for Atmospheric Research
All Rights Reserved

$Revision: 1.7 $ $Date: 1995/07/22 20:04:16 $

Resources - alphabetically

There are hundreds of resources in the full HLU resource table. Therefore, if you click on a specific object in the following list, you will be transferred to the appropriate starting point in the table for resources related to that object.

Object order

- AnnoManager - resources with prefix am
- Application - resources with prefix app
- CoordArrays - resources with prefix ca
- ContourPlot - resources with prefix cn
- CoordArrTable - resources with prefix ct
- DataComm - resources with prefix dc
- Error - resources with prefix err
- LabelBar - resources with prefix lb
- Legend - resources with prefix lg
- MapPlot - resources with prefix mp
- MapTransObj - resources with prefix mp
- PlotManager - resources with prefix pm
- ScalarField - resources with prefix sf
- Title - resources with prefix ti
- TickMark - resources with prefix tm
- Transform - resources with prefix tf
- TransObj - resources with prefix tr
- IrregularTransObj - resources with prefix tr
- LogLinTransObj - resources with prefix tr
- TextItem - resources with prefix tx
- View - resources with prefix vp
- Workstation - resources with prefix wk

- NcgmWorkstation - resources with prefix wk
- PSWorkstation - resources with prefix wk
- XWorkstation - resources with prefix wk
- Workspace - resources with prefix ws
- 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

Application 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

NhlTcnLabelScalingMode

cnLegendLevelFlags

NhlTcnLevelUseModeGenArray

cnLevelCount

NhlTInteger

cnLevelFlag

NhlTcnLevelUseMode

cnLevelFlags

NhlTcnLevelUseModeGenArray

cnLevelSelectionMode

NhlTcnLevelSelectionMode

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
cnRasterModeOn
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

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
mpAreaNames
NhlTStringGenArray
mpAreaTypes
NhlTIntegerGenArray
mpDataBaseVersion
NhlTString
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

mpGridLineColor

NhlTColorIndex

mpGridLineDashPattern

NhlTDashIndex

mpGridLineDashSegLenF

NhlTFloat

mpGridLineThicknessF

NhlTFloat

mpGridMaskMode

NhlTMapGridMaskMode

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

mpAreaMaskingOn

NhlTBoolean

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

MapTransObj 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
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
mpPreserveAspectRatio
NhlTBoolean
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
sfSubsetByIndex
NhlTBoolean
sfXArray
NhlTGenArray
sfXCEndIndex
NhlTInteger
sfXCEndSubsetV
NhlTVariable
sfXCEndV
NhlTVariable
sfXCStartIndex
NhlTInteger
sfXCStartSubsetV
NhlTVariable
sfXCStartV
NhlTVariable

sfXCStride
NhlTInteger
sfYArray
NhlTGenArray
sfYCEndIndex
NhlTInteger
sfYCEndSubsetV
NhlTVariable
sfYCEndV
NhlTVariable
sfYCStartIndex
NhlTInteger
sfYCStartSubsetV
NhlTVariable
sfYCStartV
NhlTVariable
sfYCStride
NhlTInteger

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
tmXBLabelDeltaF
NhlTFloat
tmXBLabelDirection
NhlTTextDirection
tmXBLabelFont
NhlTFont
tmXBLabelFontAspectF
NhlTFloat
tmXBLabelFontColor
NhlTColorIndex
tmXBLabelFontHeightF
NhlTFloat
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
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
tmXTLabelDeltaF
NhlTFloat
tmXTLabelDirection
NhlTTextDirection
tmXTLabelFont
NhlTFont
tmXTLabelFontAspectF
NhlTFloat
tmXTLabelFontColor
NhlTColorIndex
tmXTLabelFontHeightF
NhlTFloat
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
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
tmYLLabelDeltaF
NhlTFloat
tmYLLabelDirection
NhlTTextDirection

tmYLLabelFont
NhlTFont
tmYLLabelFontAspectF
NhlTFloat
tmYLLabelFontColor
NhlTColorIndex
tmYLLabelFontHeightF
NhlTFloat
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
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
tmYRLabelDeltaF
NhlTFloat
tmYRLabelDirection
NhlTTextDirection
tmYRLabelFont
NhlTFont

tmYRLabelFontAspectF
NhlTFloat
tmYRLabelFontColor
NhlTColorIndex
tmYRLabelFontHeightF
NhlTFloat
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
tmYRMode
NhlTTickMarkMode
tmYROn
NhlTBoolean
tmYRPrecision
NhlTInteger
tmYRStyle
NhlTTickMarkStyle
tmYRTickEndF
NhlTFloat

tmYRTickSpacingF
NhlTFloat
tmYRTickStartF
NhlTFloat
tmYRValues
NhlTFloatGenArray
tmYUseLeft
NhlTBoolean

Transform resources with prefix tf

tfPlotManagerOn
NhlTBoolean

TransObj resources with prefix tr

trOutOfRangeF
NhlTFloat

IrregularTransObj resources with prefix tr

trXAxisType
NhlTAxisType
trXCoordPoints
NhlTFloatGenArray
trXInterPoints
NhlTFloatGenArray
trXMaxF
NhlTFloat
trXMinF
NhlTFloat
trXReverse
NhlTBoolean
trXSamples
NhlTInteger
trXTensionF
NhlTFloat
trYAxisType
NhlTAxisType
trYCoordPoints
NhlTFloatGenArray
trYInterPoints
NhlTFloatGenArray
trYMaxF
NhlTFloat
trYMinF

NhlTFloat

trYReverse

NhlTBoolean

trYSamples

NhlTInteger

trYTensionF

NhlTFloat

LogLinTransObj resources with prefix tr

trXLog

NhlTInteger

trXMaxF

NhlTFloat

trXMinF

NhlTFloat

trXReverse

NhlTInteger

trYLog

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

View resources with prefix vp

vpAnnoManagerId

NhlTObjId

vpHeightF

NhlTFloat

vpKeepAspect

NhlTBoolean

vpUseSegments

NhlTBoolean

vpWidthF

NhlTFloat

vpXF

NhlTFloat

vpYF

NhlTFloat

Workstation resources with prefix wk

wkBackgroundColor

NhlTFloatGenArray
wkColorMap
NhlTFloatGenArray
wkColorMapLen
NhlTInteger
wkDashPattern
NhlTDashIndex
wkDashTableLength
NhlTInteger
wkFillTableLength
NhlTInteger
wkForegroundColor
NhlTFloatGenArray
wkGksWorkId
NhlTInteger
wkLineColor
NhlTColorIndex
wkLineDashSegLenF
NhlTFloat
wkLineLabel
NhlTString
wkLineLabelConstantSpacingF
NhlTFloat
wkLineLabelFont
NhlTFont
wkLineLabelFontAspectF
NhlTFloat
wkLineLabelFontColor
NhlTColorIndex
wkLineLabelFontHeightF
NhlTFloat
wkLineLabelFontQuality
NhlTFontQuality
wkLineLabelFontThicknessF
NhlTFloat
wkLineLabelFuncCode
NhlTCharacter
wkLineThicknessF
NhlTFloat
wkMarkerColor
NhlTColorIndex
wkMarkerIndex
NhlTMarkerIndex
wkMarkerSizeF
NhlTFloat
wkMarkerTableLength

NhlTInteger
wkMarkerThicknessF
NhlTFloat
wkMarkerXOffset
NhlTFloat
wkMarkerYOffset
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

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

$Revision: 1.10 $ $Date: 1995/07/24 18:21:53 $

HLU Types Alphabetized

- **NhlTAlternatePlace**
- **NhlTAnnotationDisplayMode**
- **NhlTAxisType**
- **NhlTBoolean**
- **NhlTBooleanGenArray**
- **NhlTByte**
- **NhlTByteGenArray**
- **NhlTcaCastMode**
- **NhlTCharacter**
- **NhlTCharacterGenArray**
- **NhlTcnHighLowLabelOverlapMode**
- **NhlTcnLabelScalingMode**
- **NhlTcnLevelSelectionMode**
- **NhlTcnLevelUseMode**
- **NhlTcnLevelUseModeGenArray**
- **NhlTcnLineLabelPlacementMode**
- **NhlTColorIndex**
- **NhlTColorIndexGenArray**
- **NhlTDashIndex**
- **NhlTDashIndexGenArray**
- **NhlTDouble**
- **NhlTDoubleGenArray**
- **NhlTDrawOrder**
- **NhlTErrorTypes**
- **NhlTFillIndex**
- **NhlTFillIndexGenArray**
- **NhlTFloat**
- **NhlTFloatGenArray**
- **NhlTFont**
- **NhlTFontQuality**
- **NhlTGenArray**
- **NhlTInteger**
- **NhlTIntegerGenArray**
- **NhlTJustification**

- **NhlTlbBoxSizingMode**
- **NhlTlbLabelAlignmentMode**
- **NhlTlgItemPlacementMode**
- **NhlTlgLabelAlignmentMode**
- **NhlTLineLabelMode**
- **NhlTLong**
- **NhlTLongGenArray**
- **NhlTMapBoundarySets**
- **NhlTMapGridMaskMode**
- **NhlTMapLimitMode**
- **NhlTMapShapeMode**
- **NhlTMarkerIndex**
- **NhlTMarkerIndexGenArray**
- **NhlTMarkLineMode**
- **NhlTMarkLineModeGenArray**
- **NhlTObjId**
- **NhlTObjIdGenArray**
- **NhlTOrientation**
- **NhlTPointer**
- **NhlTPointerGenArray**
- **NhlTPosition**
- **NhlTProjection**
- **NhlTPSFormat**
- **NhlTShort**
- **NhlTShortGenArray**
- **NhlTSpecifiedFillPriority**
- **NhlTString**
- **NhlTStringGenArray**
- **NhlTTextDirection**
- **NhlTTickMarkMode**
- **NhlTTickMarkStyle**
- **NhlTVariable**
- **NhlTVisualType**
- **NhlTWorkOrientation**
- **NhlTXColorMode**

$Revision: 1.3 $ $Date: 1995/07/20 22:14:49 $

HLU Enumeration Values Alphabetized

- **NhlABOVEITEMS (1, "AboveItems")**
- **NhlACROSS (1, "Across")**
- **NhlADJUSTVP (8,9, "AdjustVP")**
- **NhlADJUSTVPOMITOVERHL (10,11, "AdjustVPOmitOverHL")**

- **NhlALLBOUNDARIES (5, "AllBoundaries")**
- **NhlALWAYS (1, "Conditional")**
- **NhlANGLES (2, "Angles")**
- **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")**
- **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")**
- **NhlINTEGERLINELABELS (4, "IntegerLineLabels")**
- **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")**
- **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")**
- **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")**
- **NhlWINDOW (7, "Window")**

$Revision: 1.3 $ $Date: 1995/07/20 22:14:37 $

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 are the output field width and the number of significant digits. A C conversion specifier string ends with a one-character conversion

specifier, possibly preceded by a one-character conversion qualifier. HLU conversion specifiers extend this general model with several additional flag characters and 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.

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. Unless the zero flag character ('0') is specified, zeroes to the right of

	all non-zero significant digits are not included in the output string. 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 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).

$Revision: 1.7 $ $Date: 1995/07/11 22:21:05 $

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.

$Revision: 1.3 $ $Date: 1995/07/11 22:21:06 $

Resource database

Overview
Resource file syntax
Resource matching rules

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

retruns 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 Wh
FileName        = <valid filename for operating system>
ResourceSpec    = WhiteSpace ResourceName WhiteSpace ":" WhiteSpa
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 newlin
```

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*" 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 resouce 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 1987,1988,1989,1991,1993,1994,1995
University Corporation for Atmospheric Research
All Rights Reserved

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.

$Revision: 1.1 $ $Date: 1995/07/20 22:13:16 $

MapPlot database table

This table lists the default contents of the **MapPlot** 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." This is the current ordering but it should not be assumed that the areas will always remain in this order.

The second column contains an integer representation of the area type. You can retrieve an array of these values by getting the 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 "geographical" 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-Is
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-Ri
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-Scho
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-I
188	3	2	4	USSR-Ostrov-Malyy-Lyakhovskiy
189	3	2	4	USSR-Ostrov-Novaya-Sibir'
190	3	2	4	USSR-Ostrov-Oktyabr'skoy-Revolyuts
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-Ladoo
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-Cl
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·
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-
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-Isla
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-Islan
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-Isl
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-is
480	6	2	6	US-Minnesota-Lake-of-the-Woods-is
481	6	2	6	US-Minnesota-Lake-of-the-Woods-is
482	6	2	6	US-Minnesota-Lake-of-the-Woods-la
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
```

$Revision: 1.6 $ $Date: 1995/07/24 23:22:11 $

Introduction to NCL

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

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

NCL allows different datasets to be imported into one uniform and consistent data manipulation environment. The primary data format used by NCL is the netCDF data format. This network-and-architecture-transparent format has the ability to store multiple data types of multiple dimensions. Furthermore, the netCDF file contains information describing its content. For example, a given variable can have information about the units it is stored in, missing values, valid data ranges, and so on. This general information facilitates the development of a powerful generalized data manipulation environment like NCL. For more information on netCDF, consult the netCDF User's Guide. For a good understanding of NCL, you should be familiar with netCDF and basic programming language concepts.

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

Language overview

You can think of NCL as a complete programming language. It has types, variables, operators, expressions, conditional statements, and loops. NCL also has features that are not found in common programming languages. These additional features handle the manipulation of metadata and the configuration of the output graphics. NCL can operate in three modes. The first mode of operation is as an interactive command line interpreter where every statement is executed immediately after the user enters a command or expression. The second mode of operation is as a batch command interpreter where an entire NCL script is read in at one time and executed. The third mode is intended to facilitate the batch production of large quantities of output visualizations either for data exploration or video production. NCL is not meant to be a replacement for programming in other structured languages; it is meant to provide an integrated environment where data can be selected, manipulated, and visualized interactively without requiring compilation.

$Revision: 1.5 $ $Date: 1995/07/11 22:23:46 $

NCL reference pages

- Introduction to NCL
- NCL usage help
- NCL functions and procedures
- NCL language reference
  - NCL keyword listing
  - NCL data types overview

        - Basic numeric types
        - Non-numeric types
        - Coercion of types
        - Creating data
    - Variables
        - Creating variables
        - Attributes
        - Dimensions
        - Coordinate variables
        - Subscripts
            - Standard
            - Coordinate
            - Named
        - Files and file variables
    - Expressions
        - Algebraic
        - Logical
        - Arrays
        - Functions
        - Intrinsic functions
    - Statements
        - Blocks
        - if
        - Loops
            - do
            - while
        - Assignment
        - Procedures
        - Function and procedure definitions
        - Visualization blocks
            - create
            - setvalues
            - getvalues
        - load script from file
        - record commands
            - record
            - stop record
- NCL API
- Extending NCL's function set

$Revision: 1.9 $ $Date: 1995/07/11 22:23:59 $

NCL functions and procedures

General NCL functions and procedures

Functions

- addfile
- all
- any
- asciiread
- bsearch
- cbinread
- dimsizes
- fbinread
- getenv
- ismissing
- prompt
- rand
- sizeof

Procedures

- asciiwrite
- cbinwrite
- delete
- fbinwrite
- list_files
- list_filevars
- list_hlus
- list_procfuncs
- list_vars
- print
- qsort
- sleep
- srand
- system

NCL math functions

Functions

- abs
- acos
- asin
- atan
- ceil
- cos
- cosh
- fabs
- floor

- idsfft
- log
- log10
- sin
- sinh
- sqrt
- tan
- tanh

NCL versions of HLU functions and procedures

Functions

- NhlAddAnnotation
- NhlAddData
- NhlClassName
- NhlGetBB
- NhlIsAllocatedColor
- NhlName
- NhlNewColor

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)
- datatondc (NhlDataToNDC)
- destroy (NhlDestroy)
- draw (NhlDraw)
- frame (NhlFrame)
- ndctodata (NhlNDCToData)
- overlay
- update (NhlUpdateWorkstation)
- NhlChangeWorkstation
- NhlClearWorkstation (clear)
- NhlDataPolyline
- NhlDataToNDC (datatondc)
- NhlDestroy (destroy)
- NhlDraw (draw)
- NhlFrame (frame)
- NhlFreeColor
- NhlNDCPolyline
- NhlNDCToData (ndctodata)
- NhlRemoveAnnotation

- NhlRemoveData
- NhlRemoveOverlay
- NhlSetColor
- NhlUpdateData
- NhlUpdateWorkstation (update)

NCL versions of LLU functions and procedures

Functions

- ncargpath

Procedures

- ncargversion

NCL data type conversion functions

- chartodouble
- chartofloat
- chartoint
- chartolong
- chartoshort
- doubletobyte
- doubletochar
- doubletofloat
- doubletoint
- doubletolong
- doubletoshort
- floattobyte
- floattochar
- floattoint
- floattolong
- floattoshort
- inttobyte
- inttochar
- inttoshort
- longtobyte
- longtochar
- longtoint
- longtoshort
- shorttobyte
- shorttochar

$Revision: 1.17 $ $Date: 1995/07/24 22:13:12 $

---

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.

$Revision: 1.6 $ $Date: 1995/07/24 22:11:54 $

all

This function returns True if all the elements of the input parameter are set to True.

Synopsis

```
function all(
        logical_array : logical
)
```

Arguments

logical_array
: Any logical array or scalar of any number of dimensions.

Description

The **all** function returns a single logical scalar value only if all of the elements of the logical input parameter are True. This function is useful for changing multidimensional logical results from conditional expressions into a single scalar value, which is required by the if statement.

$Revision: 1.4 $ $Date: 1995/07/11 22:21:51 $

---

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
: An array of logical values that are either a result of a conditional expression or a variable that references logical values.

Description

The **any** function is one way to condense a multidimensional result of a relational operation into a single scalar logical value for an if statement. **any** returns True if any of the elements in the multidimensional input are True.

$Revision: 1.4 $ $Date: 1995/07/11 22:21:52 $

---

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.

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.

$Revision: 1.4 $ $Date: 1995/07/11 22:21:56 $

---

bsearch

This function is not implemented yet.

Synopsis

```
function bsearch(
)
```

Arguments

ARG1
: ARG1 ...

ARG2
: ARG2 ...

Description

$Revision: 1.4 $ $Date: 1995/07/11 22:23:27 $

---

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.

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.

$Revision: 1.7 $ $Date: 1995/07/11 22:21:54 $

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.

$Revision: 1.4 $ $Date: 1995/07/11 22:22:08 $

---

fbinread

This function is used read in binary files that have been written using an UNFORMATED FORTRAN write.

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.

datatype
: A string representing the type of the data being read.

Description

The **fbinread** function is used to read in binary data. The data must have been written on a machine having the same architecture as the machine on

which they were written. If the specified dimensions do not specify 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.

$Revision: 1.6 $ $Date: 1995/07/11 22:22:11 $

---

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.

$Revision: 1.4 $ $Date: 1995/07/11 22:22:20 $

---

ismissing

This function returns a single scalar logical value of True when its parameter references a missing value (determined by the _FillValue attribute).

Synopsis

```
function ismissing (
        data[1]
)
```

Arguments

data
: Can be any type of NCL scalar data value. Must have a single dimension and a dimension length of one.

Description

The **ismissing** function returns True when the element passed to it contains a missing value. This is useful for filtering missing values.

$Revision: 1.6 $ $Date: 1995/07/26 15:13:42 $

---

prompt

This function is used ...

Synopsis

```
function FUNCNAME(
        ARGUMENTS
)
```

Arguments

ARG1
: ARG1 ...

ARG2
: ARG2 ...

Description

The **FUNCNAME** function ....

$Revision: 1.4 $ $Date: 1995/07/11 22:23:08 $

---

rand

This function is used to generate psuedo random numbers.

Synopsis

```
function rand( )
```

Description

The **rand** function returns a psuedo 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.**

$Revision: 1.4 $ $Date: 1995/07/11 22:23:10 $

---

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.

$Revision: 1.5 $ $Date: 1995/07/11 22:23:06 $

---

asciiwrite

This procedure is used create an ascii text file of NCL numeric data type.

Synopsis

```
procedure asciiwrite(
        filepath[1]:string,
        value:numeric
)
```

Arguments

filepath
Path for new ascii file.
value
A numeric value of any dimensionality.

Description

The asciiwrite function is used to create an ascii data file from an NCL numeric variable. Each element of the *value* parameter is written in its string representation followed by a return. The data are order in their row x column internal ordering.

$Revision: 1.5 $ $Date: 1995/07/11 22:21:49 $

---

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 machines native binary format.

$Revision: 1.7 $ $Date: 1995/07/11 22:22:01 $

delete

This function deletes variables from the NCL and frees the symbol name from the symbol table.

Synopsis

```
procedure delete(
```

```
        data
)
```

Arguments

data
Any variable of any type or dimensionality.

Description

This function is used to delete variables from NCL. Although it will accept expression results and literal values, deleting these does nothing to the NCL symbol table. However, 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. 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.

$Revision: 1.6 $ $Date: 1995/07/24 22:11:57 $

---

fbinwrite

This function is used write create a binary file in raw C block I/O format from and NCL numeric data type.

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 fbinread function is used to create a binary data file from an NCL numeric variable. The variable is written to a file as a single unformatted record.

$Revision: 1.7 $ $Date: 1995/07/11 22:22:12 $

---

list_files

This procedure is used ...

Synopsis

```
procedure list_files( )
```

Description

Lists all of the current variables containing references to files.

$Revision: 1.4 $ $Date: 1995/07/11 22:22:54 $

---

list_filevars

This procedure lists all of the variables associated with a specific file.

Synopsis

```
function list_filevars(
        filevar[1]:file
)
```

Arguments

filevar

A single reference to an open file. Note that NCAR Graphics Version 4.0 does not support arrays of files.

Description

All of the variables referenced by the *filevar* parameter are listed.

$Revision: 1.4 $ $Date: 1995/07/11 22:22:55 $

---

list_hlus

This procedure is used to list all of the HLU objects currently referenced by NCL.

Synopsis

```
procedure list_hlus( )
```

Description

This procedure lists every variable reference to an HLU object currently in the NCL identifier list.

$Revision: 1.4 $ $Date: 1995/07/11 22:22:57 $

---

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.

$Revision: 1.4 $ $Date: 1995/07/11 22:22:58 $

---

list_vars

This procedure is used to list the currently defined variables.

Synopsis

```
function list_vars( )
```

Description

Lists all of the currently defined variables, their types, and their dimensionality.

$Revision: 1.4 $ $Date: 1995/07/11 22:22:59 $

---

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.

$Revision: 1.5 $ $Date: 1995/07/11 22:22:53 $

---

qsort

This procedure is used to sort single-dimension arrays.

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* and any associated coordinate variable. Missing values are sorted to either end of the array based on their actual value. The qsort algorithm does not

support ignoring missing values.

$Revision: 1.5 $ $Date: 1995/07/11 22:23:04 $

---

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.

$Revision: 1.5 $ $Date: 1995/07/11 22:23:09 $

---

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.

$Revision: 1.4 $ $Date: 1995/07/11 22:23:19 $

system

This procedure is used to pass system commands to the shell.

Synopsis

```
function 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.

$Revision: 1.4 $ $Date: 1995/07/11 22:23:20 $

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 an array dimensioned the same as the *value* parameter. Missing values are ignored.

$Revision: 1.4 $ $Date: 1995/07/11 22:21:47 $

acos

This function is used to take the inverse cosine.

Synopsis

```
function acos(
        value:float
)
```

Arguments

value
: **One or more floating point 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. Missing values are ignored. Returns a floating point array dimensioned the same as the input parameter.

$Revision: 1.5 $ $Date: 1995/07/11 22:21:45 $

---

asin

This function computes the inverse sine.

Synopsis

```
function asin(
        value:float
)
```

Arguments

value

An array of one or more floating 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*. Missing values are ignored.

$Revision: 1.4 $ $Date: 1995/07/11 22:21:58 $

---

atan

This function computes the inverse tangent.

Synopsis

```
function atan(
        value:float
)
```

Arguments

value

An array of one or more floating 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*. Missing values are ignored.

$Revision: 1.4 $ $Date: 1995/07/11 22:21:59 $

ceil

This function returns the smallest integral value larger than the input.

Synopsis

```
function ceil(
        value:float
)
```

Arguments

value

An array of one or more floating-point 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.

$Revision: 1.5 $ $Date: 1995/07/11 22:21:55 $

---

cos

This function returns the cosine of its input.

Synopsis

```
function cos(
        value:float
)
```

Arguments

value

An array of one or more floating point 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 paramter.

$Revision: 1.5 $ $Date: 1995/07/11 22:22:02 $

---

cosh

This function returns the hyperbolic cosine of its input.

Synopsis

```
function cosh(
        value:float
)
```

Arguments

value

An array of one or more floating-point 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.

$Revision: 1.5 $ $Date: 1995/07/11 22:22:04 $

fabs

This function is used to return the absolute values for floating-point numbers.

Synopsis

```
function fabs(
        value:float
)
```

Arguments

value

One or more float 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.

$Revision: 1.4 $ $Date: 1995/07/11 22:22:14 $

---

floor

This function returns the largest integral value smaller than the input.

Synopsis

```
function floor(
        value:float
)
```

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.

$Revision: 1.4 $ $Date: 1995/07/11 22:22:18 $

---

idsfft

This function is used to call the NCAR 3.2 FORTRAN function IDSFFT.

Synopsis

```
function idsfft(
        x[*]:float,
        y[*]:float,
        z[*]:float,
        dimensions[2]:integer
)
```

Arguments

x

Array of 4 or more x coordinates.

y

Array of 4 or more y coordinates dimensioned the same as *x*.

z

Array of 4 or more z coordinates dimensioned the same as *y*.

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. *x* and *y* are randomly spaced coordinates in the X-Y plane, and *y* contains the values at each X-Y coordinate pair. No two values of *x* can be equal to each other. If this condition occurs, idsfft prints an error message. NOTE: Early versions of the idsfft function exit when an error is detected; this will be modified by the 4.0 release.

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 *x* and *y*, and dividing the range by the size of the requested dimension. The *x* values define the coordinate range of the second dimension (dimension #1) of the output array, and the *y* values determine the coordinate range of the first dimension (dimension #0). The output array returned contains the derived coordinate arrays as coordinate variables (See coordinate variables description). Dimension 0 of the output is named "y" and dimension 1 of the output is named "x". For more information, see the NCAR Graphics Version 3.2 man page for IDSFFT.

$Revision: 1.5 $ $Date: 1995/07/11 22:22:16 $

log

This function is used to compute the natural log of the input.

Synopsis

```
function log(
        value:float
)
```

Arguments

value

An array of one or more floating-point 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.

$Revision: 1.4 $ $Date: 1995/07/11 22:23:01 $

log10

This function is used to compute the log base 10 of the input.

Synopsis

```
function log10(
        value:float
)
```

Arguments

value

An array of one or more floating-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.

$Revision: 1.4 $ $Date: 1995/07/11 22:23:02 $

---

sin

This function is computes the sine function.

Synopsis

```
function sin(
        value:float
)
```

Arguments

value

An array of one or more floating-point values of any dimension, in radians.

Description

The asin 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.

$Revision: 1.4 $ $Date: 1995/07/11 22:23:12 $

---

sinh

This function computes the hyperbolic sine function.

Synopsis

```
function sinh(
        value:float
)
```

Arguments

value

An array of one or more floating-point 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.

$Revision: 1.4 $ $Date: 1995/07/11 22:23:13 $

sqrt

This function computes the square root of its input.

Synopsis

```
function sqrt(
        value:float
)
```

Arguments

value
An array of one or more positive floating-point 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.

$Revision: 1.4 $ $Date: 1995/07/11 22:23:17 $

---

tan

This function is computes the tangent of the input.

Synopsis

```
function tan(
        value:float
)
```

Arguments

value
An array of one or more floating-point values of any dimension.

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.

$Revision: 1.4 $ $Date: 1995/07/11 22:23:22 $

---

tanh

This function computes the hyperbolic tangent of the input.

Synopsis

```
function tanh(
        value:float
)
```

Arguments

value

An array of one or more floating-point values of any dimension.

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.

$Revision: 1.4 $ $Date: 1995/07/11 22:23:23 $

NhlAddAnnotation

This function is used to add an annotation to a plot object as an external annotation. It returns a reference to an AnnoManager instance. (See NhlAddAnnotation.)

Synopsis

```
function NhlAddAnnotation(
        plot_id : 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
: **The view object to be added as an annotation.**

Description

This function adds one or more views to one or more plots 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 each element of *plot_id*. The output is dimensioned m x n, where m is the dimension of the *plot_id* parameter, and n is the dimension of *anno_view_id*.

$Revision: 1.7 $ $Date: 1995/07/20 22:15:52 $

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*.

$Revision: 1.7 $ $Date: 1995/07/20 22:15:52 $

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.

$Revision: 1.7 $ $Date: 1995/07/20 22:15:53 $

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 coordinates 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.

$Revision: 1.6 $ $Date: 1995/07/20 22:15:57 $

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[1]:graphic,
        ci[*]:integer
)
```

Arguments

work

A single HLU Workstation instance.

ci

An array of one or more color indexes. Valid range is 0-255.

Description

The Workstation instance referenced by *work* is queried to see whether each of the elements 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.

$Revision: 1.6 $ $Date: 1995/07/20 22:15:58 $

NhlName

This function is used to retrieve the name of one or more HLU object instances.

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.

$Revision: 1.5 $ $Date: 1995/07/11 22:21:35 $

NhlNewColor

The function is used to allocate new workstation color indexes.

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.

$Revision: 1.5 $ $Date: 1995/07/11 22:21:36 $

NhlChangeWorkstation

This procedure is used to change the output workstation of one or more HLU View instances. <<<<<<< NhlChangeWorkstation.html (See NhlChangeWorkstation.) ======= (See NhlChangeWorkstation.) >>>>>>> 1.2

Synopsis

```
procedure NhlChangeWorkstation(
        workstation[1]:graphic,
        hlu_objs:graphic
)
```

Arguments

workstation
HLU Workstation instance.
hlu_objs
An array of one or more references to HLU View instances.

Description

This procedure is used to change the parent workstion for one or more

HLU View instances. The parent workstation for each View instance in the array *hlu_obj* is to the workstation referenced by *workstation*.

$Revision: 1.3 $ $Date: 1995/07/25 19:58:32 $

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.

$Revision: 1.3 $ $Date: 1995/07/20 22:15:54 $

NhlDataPolyline

This procedure is used to draw a polyline using data coordinates. (See NhlDataPolyline.)

Synopsis

```
procedure NhlDataPolyline(
        objs[*]: graphic,
        x[*]: float,
        y[*]: float
)
```

Arguments

objs

An array of HLU plot objects.

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* in the viewport 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 to one or more HLU plot objects, and the NCL version does not need the length of the *x* and *y* procedures because this information is inherent in NCL data.

$Revision: 1.7 $ $Date: 1995/07/20 22:15:54 $

NhlDataToNDC (datatondc)

Converts data units into normalized device coordinates(NDC). (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 objects to map data coordinates in to 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_int*. When the data coordinates are outside of the data transformation specified in the plot, a missing value is placed in both of the coresponding indexes in the output arrays. The missing value chosen is either the *x_in* missing value, *y_in* missing value, or a system default.

$Revision: 1.4 $ $Date: 1995/07/20 22:15:55 $

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.

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.

$Revision: 1.3 $ $Date: 1995/07/11 22:23:15 $

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.

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.

$Revision: 1.4 $ $Date: 1995/07/25 16:12:01 $

NhlDraw

This procedure is used to call the HLU draw function (NhlDraw) on NCL data that are arrays of HLU View instances.

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.

$Revision: 1.4 $ $Date: 1995/07/20 22:15:56 $

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.

$Revision: 1.8 $ $Date: 1995/07/25 16:12:02 $

NhlNDCPolyline

This procedure is used to draw a polyline using NDC coordinates. (See NhlNDCPolyline.)

Synopsis

```
procedure NhlNDCPolyline(
        plots[*]: graphic,
        x[*]: float,
        y[*]: float
)
```

Arguments

plots
: **An array of instances of HLU plot objects.**

x
: **The X coordinates of the polyline in NDC coordinates.**

y
: **The Y coordinates of the polyline in NDC coordinates.**

Description

NhlNDCPolyline draws the polyline defined by the pair of vectors *x* and *y* in the viewport of each HLU plot in the *plots* array. The difference between the NCL version and the HLU version is that the NCL version draws the polyline to one or more HLU plots, and the NCL version does not need the length of the *x* and *y* procedures because this information is inherent in NCL data.

$Revision: 1.7 $ $Date: 1995/07/20 22:15:58 $

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 coordinates.

y_in

Single-dimensioned floating-point vector of y NDC 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 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 objects to map NCL 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.

$Revision: 1.4 $ $Date: 1995/07/20 22:15:59 $

NhlRemoveAnnotation

This procedure is used to remove annomanagers from the plots they are registered in. (See NhlRemoveAnnotation.)

Synopsis

```
procedure NhlRemoveAnnotation(
        plot_id : graphic,
        anno_manager_id : graphic
)
```

Arguments

plot_id
: **One or more HLU plots.**

anno_manager_id
: **References to AnnoManager instances to be removed from each of the *plot_id* elements.**

Description

This function is used to remove an external annotations from the plots they have previously been added to.

$Revision: 1.7 $ $Date: 1995/07/20 22:16:00 $

NhlRemoveData

This procedure is used to remove data items from one or more plots. (See NhlRemoveData)

Synopsis

```
procedure NhlRemoveData(
        plot_objs[*]:graphic,
        data_objs[*]:graphic
)
```

Arguments

plot_objs
: **An array of one or more HLU plots from which data is to 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.

$Revision: 1.6 $ $Date: 1995/07/20 22:16:00 $

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 overlay description for more detail.

$Revision: 1.1 $ $Date: 1995/07/24 22:11:51 $

NhlSetColor

The function is used to allocate new workstation color indexes.

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.

$Revision: 1.5 $ $Date: 1995/07/11 22:21:42 $

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 the above 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.

$Revision: 1.6 $ $Date: 1995/07/20 22:16:01 $

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 NhlUpdataWorkstation procedure is used to flush the internal buffers of HLU Workstation instances. It calls the the HLU function NhlUpdateWorkstation for each element of the *work* array.

$Revision: 1.4 $ $Date: 1995/07/20 22:16:02 $

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 the HLU function NhlUpdateWorkstation for each element of the *work* array.

overlay

Used to overlay one HLU plot object over a specific style of transformation.

Synopsis

function 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 draw one plot 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. The return value of the overlay function is the reference to the annotation object created.

$Revision: 1.5 $ $Date: 1995/07/11 22:23:05 $

---

ncargpath

This procedure is used to change the output workstation of one or more HLU View instances.

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 path name for the requested directory. Below is a list of the valid strings. An invalid string returns the 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.
```

$Revision: 1.3 $ $Date: 1995/07/26 00:58:24 $

---

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.

$Revision: 1.2 $ $Date: 1995/07/25 19:58:29 $

NCL data type conversion functions

These functions are used to coerce values from one basic data type to another.

Synopsis

```
function inttobyte(
        int_val : integer
)


function inttochar(
        int_val : integer
)

function inttoshort(
        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 longtoint(
        long_val : long
)

function floattobyte(
        float_val : float
)

function floattochar(
        float_val : float
)

function floattoshort(
        float_val : float
)

function floattoint(
```

```
        float_val : float
)

function floattolong(
        float_val : float
)

function doubletobyte(
        double_val : double
)

function doubletochar(
        double_val : double
)

function doubletoshort(
        double_val : double
)

function doubletoint(
        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 chartoint(
        char_val : character
)
```

Description

All of the functions listed here function 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.

$Revision: 1.4 $ $Date: 1995/07/11 22:23:16 $

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
```

$Revision: 1.5 $ $Date: 1995/07/24 22:13:14 $

begin keyword

The *begin* keyword is used in the following NCL statement types:

```
Blocks
Function and Procedure Definitions
```

$Revision: 1.4 $ $Date: 1995/07/11 22:23:55 $

break keyword

The *break* keyword can be used in the following NCL statment types:

```
Do
While
If
```

$Revision: 1.4 $ $Date: 1995/07/11 22:23:56 $

continue keyword

The *continue* keyword can be used in the following NCL statment types:

```
Do
While
If
```

$Revision: 1.4 $ $Date: 1995/07/11 22:23:49 $

The *end* keyword is used in the following NCL statement types:

```
Do
While
```

```
If
create
getvalues
setvalues
Blocks
Function and procedure definitions
```

$Revision: 1.5 $ $Date: 1995/07/24 22:13:09 $

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. Assigment 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 graphic type is a reference to an instance of an HLU object. The file type is a reference to a disk file. The logical type is generated from relational expressions. Logical values are either True or False. Graphic values are returned by the create 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

There are many times that types must be converted from one basic data type to another. In many cases, this coercion occurs automatically. For example, when a float value is multiplied by an integer value, the integer value must be converted to a float value before the multiplication can occur. The following table lists the conversion possibilities for all of the basic types. If a coercion to a common type doesn't exist for both sides of an expression, then a type mismatch error is generated.

Type	Coercible to
character	string
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 doubletoint 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

All numeric data types, strings, characters, and logical and graphic types can be created using the *new* statements. There are two types of new calls, one that creates the array of data with a specific missing value assigned to each element, and one that creates the array of data using the default missing value. The *new* statement takes, as parameters, an array of dimension sizes, the type keyword to create, and optionally a missing value to assign to each element of the new data array. The following creates a three-dimensional float array:

```
a = new((/5,6,7/),float)
```

The following is an example of how to assign a specific missing value.

```
a = new((/5,6,7/),float,-1e12)
```

File data types are created differently. Files are only created by the intrinsic function *addfile*. Arrays of the graphic type can be created with new, but none of the elements of the array will actually reference an existing HLU object. Each element will contain a missing value.

In addition to new, there are ways of entering data manually. Currently, there are four types of literal values. Floating point numbers must be entered with a decimal point. The following show creation of data using literal floating point values:

```
a = 2.
a = 1.2
a = (/ 1., 2., 3. /)
```

Integer values are specified without the decimal point:

```
a = 2
a = (/ 1, 2, 3 /)
```

String values are specified as characters enclosed in double quotes (''):

```
a = "NCAR Graphics"
```

NOTE: There is currently no way to specify characters literally.

All other types must be forcibly coerced. See above.

$Revision: 1.8 $ $Date: 1995/07/11 22:23:52 $

NCL variables overview

Variable names must begin with an alphabetic character, but they can contain any mix of numbers 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 can be created by the assignment statement, or by the new statement.

Creating variables

Variables of all numeric data types, strings, characters, and logical and graphical types can be created using the *new* intrinsic statement. There are two types of new calls, one that creates the array of data with a specific missing value assigned to each element, and one that creates the array of data using the default missing value. The *new* statement takes, as parameters, an array of dimension sizes, the keyword representing the type to create, and optionally a missing value to assign to each element of the new data array. The following creates a three-dimensional float array:

```
a = new((/5,6,7/),float)
```

The following is an example of how to assign a specific missing value:

```
a = new((/5,6,7/),float,-1e12)
```

File data types are created differently. Files are only created by the intrinsic *addfile*. Arrays of type graphic can be created with new, but none of the elements of the array will actually reference an existing HLU object. Each element will contain a missing value.

Attributes

Attributes are descriptive pieces of information that can be associated with an already existing variable, file, or file variable. Attributes can be assigned single-dimensioned arrays or strings, but not files or graphical objects. 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 is an example 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
temperature@scale_factors = (/ .233, .452, .434, .621 /)
```

Attributes can be treated just like variables. They can be used in expressions and subscripted in

the same fashion as variables. The attribute *_FillValue* is a special reserved attribute name that denotes what values in the variable are missing or filled values. Missing values are significant when evaluating expressions. In the above examples, if the attributes were already defined, they are reassigned the new value. In the case of the *_FillValue* attribute, every occurrence of the previous value in the variable temperature is replaced with the new *_FillValue*. The *_FillValue* attribute must be the same type as the data type referenced by the variable.

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. 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 their dimension. Additionally, the values in these arrays must be monotonically increasing or decreasing. Non-monotonic values can be assigned, but coordinate subscripting will not work. When a non-monotonic array is assigned, a warning message is printed. These arrays represent the data coordinates for each index in the named dimension. Coordinate variables support the concept of coordinate subscripting. 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
```

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.

Standard subscripts

Standard subscripting includes the capability of selection ranges and strides as subscript in addition to the ability to select data using a vector of integer indices. 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 indices begin at 0. Standard subscript indices 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. For example,

```
temperature(1:3,5,6)
temperature(1:3,4:5,5:6)
```

The first selection above 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.

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.

Finally, a vector of integer indices 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. This also selecting a single index more than one. For example, take the following array and its use

as a vector subscript 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 indices 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 make the subselections. When specifying a coordinate subscript, braces (’{’ ’}’) are used to state that the subscript should use the indices that the start and end values are located in from the coordinate variable. Essentially, the start and end values are "looked" up in the coordinate variable, and the indices 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.

```
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.

Files and file variables

It is very important to understand the distinction between a variable that references a file and a file variable. When a file is opened with the addfile command, a reference to the file is assigned to a variable. The type of 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.

Variables that reference files have a few special exceptions. First, dimension names and numbers do not represent the shape of the file variable; they are the dimensions used by the variables stored in the file. Coordinate variables are handled similarly. Attributes, coordinate variables, and dimension names, when assigned to a variable referencing a file, are written into the file rather than saved in memory as with normal variables.

Furthermore, when a variable references a file, the '->' operator is used to select variables stored in the file.

```
file->variable1
```

The above demonstrates referencing the file variable name variable1 in the file reference by the variable file. This type of referencing can occur either on the left or the right side of an assignment. However, if on the left side, the file must be opened with write permissions.

File variables can be operated on and subscripted in the exact same fashion that standard variables can be with the '!', '&' and '@' operators. Dimensions and coordinate variables are not unique to individual file variables; they are shared by all variables in the file with the same dimensions. This means that adding or modifying dimension names and coordinate variables for one file variable changes them for all variables with the same dimension.

Variables that reference files are created by the addfile intrinsic function.

$Revision: 1.9 $ $Date: 1995/07/11 22:23:42 $

NCL expressions overview

Expressions are anything that results in a value. Variable references are expressions as well as functions and literal values. Multiple expressions can occur on a single line joined together by operators and the are group by terms and operator precedence.

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 expression that should be computed first.

```
-       Negative
^       Exponent
*       Multiply
/       Divide
%       Modulus
+       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. 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 integral types, the result is an integer type. Therefore, the decimal remainder is truncated.

Modulus %

The result is the integral remainder of integral division. Both operands must be integral types.

Plus +

Adds left operand to right operand. No special conditions or restrictions.

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 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.

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 type results. And, or, exclusive-or, and not require logical operands, and the rest accept any type.

All of the logical expressions function similarly with respect to comparisons 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 the size 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 and 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 slashes '(/' and '/)'. 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 /)
```

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 literal 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. If a parameter is coerced before the function is called, changes within the function *will not* be reflected in the parameter. 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 changes made to the parameter can be passed out from the function, unless it is returned by the function.

Intrinsic functions

An identifier with a list of parameters, the parameters being separated by commas enclosed and in parentheses. Intrinsic functions return values when called. An intrinsic function is not defined by NCL source; it is a C or Fortran routine that has been added to the NCL function set. Intrinsic functions often perform operations that NCL source does not support.

$Revision: 1.8 $ $Date: 1995/07/25 18:47:13 $

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. Since logical operators can operate on entire arrays of values, produce array results, and each element in the array result could be True, False, or a missing value, *if* statements require the results of the logical expression to be a single scalar logical value. When *if* statements encounter a missing value instead of True or False, a FATAL error message is generated and execution is terminated if the if statement is in a block statement list. There are three intrinsic functions that help with this problem. The intrinsic 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 intrinsic function *ismissing* returns True if the input to it is a missing 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 demonstrates this:

```
if((i .lt. dimsizes(a)) .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 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.

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"
file->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,.../)
file->a&Dimension1 = (/.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
file->a@units = "Degrees C"
```

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 a parameter must be coerced to the correct type, the variable is not affected by changes to the parameter. Even parameters that are subsections of variables are passed by reference. When the procedure finishes, the values of the subsection are mapped back into their original locations in the referenced variable.

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 name to assign the function to, followed by a list of declarations that can optionally be followed by a list of local variable names. Finally, a block of statements follows.

function ***function_name*** **(** ***declaration_list*** **)**
local ***local_identifier_list***
begin
    statement list
    return(*return_value*)
end

It is very important to note that if a variable is intended to be local to the function or procedure does not occur in the local list that the function may find that variable name outside of the scope of the function or procedure. It is very 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.

As mentioned 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.

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

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.

setvalues ***obj_reference***
resource_string* : *value_expression
end setvalues

getvalues

getvalues is used to retrieve resource values from HLU objects. It is different than the setvalues and create NCL resource list in that it requires an identifier rather than an expression after the colon. getvalues functions identically to the assignment expression with respect to how it assigns values to the identifier.

getvalues ***obj_reference***
resource_string* : *identifier
end getvalues

load script from 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 literal 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*** **literal string value. The "stop record" command terminates this process.**

record "*file_path***"**
...
statements
...
stop record

$Revision: 1.11 $ $Date: 1995/07/24 22:13:15 $

Extending the NCL function and procedure set

NCL has been designed to allow users the ability to extend the built-in default function set. Although this functionality is available a far 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.

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 polynomials 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 function. First and most important the C function, FORTRAN function or FORTRAN subroutine name must not conflict with an 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 mentioned on this page should be used.

The following is the C source 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, 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 and interface for passing back data. It currently does not support passing back coordinates 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 d
                                       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

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 an 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 which 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 parameters 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 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 r
);

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 NewAr
                                                     by successive calls to SetA
        char* fname,                              /* String name of function to
        int n_args                                /* Total number of arguments t
);
```

The following is the source for the function NclAddUserFuncs. More that one user function can be registered from with-in 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

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 need in the order they should linked in to compile line:

```
libhlu.a                HLU library

libncarg_gksC.a         NCAR Graphics gks C bindings 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
```

Usage help page

This page is intended to provide helpful hints for using NCL efficiently and effectively. A general working knowledge of the NCL syntax is required for this page.

Supported file format differences

Each supported data format may have its own limitations with respect to operations provided by NCL. The user guide explains the difference (if any) between supported data formats. See the Importing and exporting data section of the user guide.

Use of Blocks

When writing scripts it can be very useful to enclose the script in a block. This forces NCL to scan the entire source of the script in 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
```

Memory usage vs. Disk I/O

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 id's are store. 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)
```

Defining functions and procedures in scripts

Because functions and procedures can only be defined once and can not be deleted like variables, it is important not to put them in scripts that are intended to be run multiple times from the same invocation of NCL.

Array assignment

NCL's array assigment capability can greatly enhance efficiency in scripts whenever possible. Consider the following script to remove duplicate entries in a sorted array "x".

```
len = dimsizes(x)
do i = 0 , len - 2
        if((x(i) .eq. x(i+1)))
                do j = i , len - 2
                        x(j) = x(j+1)
                end do
                len = len - 1
                i = i - 1       ;Must decriment i in case more than one duplicate
        end if
        if(i .ge. len - 2)
                break
        end if
end do
```

The above NCL source can be written as follows

```
len = dimsizes(x)
do i = 0 , len - 2
        if((x(i) .eq. x(i+1)))
                x(i:len-2) = x(i+1:len-1)
                len = len - 1
                i = i - 1       ;Must decriment i in case more than one duplicate
        end if
        if(i .ge. len - 2)
                break
        end if
```

```
end do
```

If "x" is a very large array a tremendous about of memory allocations and frees can be saved, thus improving performance.

Do loop limitations

In NCL do loops currently have the limitation that once the do loop is initiated. The upper bound (the stopping point) can not be changed. The the following loop demonstrates this property. Since l is set to 10 when the loop starts, i will loop from 0 to 10 regardless of what l ends up being set to.

```
l = 10
do i = 0,l
        l = l - i
end do
```

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.

$Revision: 1.6 $ $Date: 1995/07/24 22:13:08 $

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 is 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.