Copy No._____

Defence Research and Development Canada
Recherche et développement pour la défense Canada

DEFENCE R&D DÉFENSE

An IDL Object for Managing Gridded Bathymetry Files

W.A. Roger

Defence R&D Canada – Atlantic

Technical Memorandum
DRDC Atlantic TM 2005-265
March 2006

Canada

This page intentionally left blank.

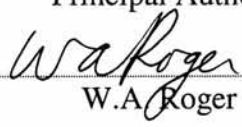
An IDL Object for Managing Gridded Bathymetry Files

W.A. Roger

Defence R&D Canada – Atlantic

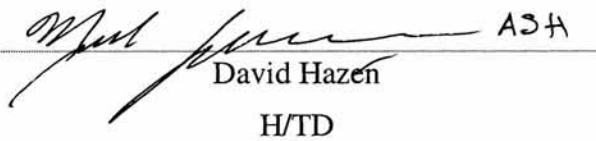
Technical Memorandum
DRDC Atlantic TM 2005-265
March 2006

Principal Author



W.A. Roger

Approved by



David Hazen

H/TD

Approved for release by

Kirk Foster

DRP

Abstract

The Networked Underwater Warfare (NUW) Technology Demonstration Project (TDP) has a need for precise ocean bottom topography of the Scotian Shelf area. The depth of the ocean at particular locations is utilized by a number of technologies within the project to either predict future ocean conditions, or as input to tactical algorithms. The ocean bathymetry over some specified area is also needed as one layer in the display of targets. This document describes a software object to extract and display underwater depths from a gridded bathymetry file. It is written in the IDL programming language to facilitate ease of use and rapid development of the controlling software. The object first reads the file into memory for quick response, then provides a number of software methods to obtain selected areas of depth, display all or portions of the bathymetry, and allow additions to the display such as contour lines and lines of bearing. The object can handle a range of bathymetry files that store depths at locations that are separated by a constant angular difference in both latitude and longitude.

Résumé

Dans le cadre du projet de démonstration de technologies (PDT) de guerre sous-marine en réseau (GSMR), on a besoin de données topographiques précises relatives au fond océanique de la région de la plate-forme Néo-Écossaise. La profondeur de l'océan en des points particuliers est utilisée par un certain nombre de technologies dans le cadre du projet soit pour la prévision des conditions futures de l'océan, soit comme paramètre d'entrée d'algorithmes tactiques. Les données bathymétriques relatives à une zone spécifiée sont aussi requises comme couche dans l'affichage des cibles. Le présent document décrit un objet logiciel permettant d'extraire et d'afficher les profondeurs sous-marines à partir d'un fichier de données bathymétriques maillées. Cet objet est écrit en langage de programmation IDL de manière à faciliter l'utilisation et à permettre l'élaboration rapide du logiciel de contrôle. L'objet charge d'abord le fichier en mémoire pour permettre une réponse rapide, puis il offre un certain nombre de méthodes logicielles permettant d'obtenir des zones de profondeur choisies, d'afficher la totalité ou une partie des données bathymétriques et d'ajouter des éléments à l'affichage, par exemple des courbes de niveau et des lignes de relèvement. L'objet peut traiter une gamme de fichiers bathymétriques dans lesquels sont stockées les profondeurs à des emplacements séparés par une différence angulaire constante tant en latitude qu'en longitude.

This page intentionally left blank.

Executive summary

An IDL Object for Managing Gridded Bathymetry Files

W.A. Roger; DRDC Atlantic TM 2005-265; Defence R&D Canada – Atlantic; March 2006.

Introduction or background

The Networked Underwater Warfare Technology Demonstration Project is building a research-level combat system to investigate issues in sharing sensor information among cooperating ASW platforms. Part of the combat system includes a tactical oceanography component called the Environment Modeling Manager (EMM), which provides predictions of ocean acoustic conditions to tactical decision aids. The core component of the EMM is an acoustic prediction engine the requires precise information on the depth of the ocean out along a portion of a great circle route. In addition, the project has need of bathymetry information that covers a rectangular area, often used for planning vessel deployment during sea trials.

This document describes a software object that provides ocean bottom depths in a variety of formats. It is written in the Interactive Data Language (IDL) for ease of use and efficient code generation.

Results

The object reads a gridded bathymetry file and provides the depth information to the controlling program. Ocean depths can be requested for a single chart location, along a line-of-bearing out to some range, and over a rectangular area. In turn, the data can be displayed or provided to signal processing and acoustic prediction algorithms.

Significance

The bathymetry object provides a convenient and flexible means to acquire ocean depth information at specific locations. It serves as part of the infrastructure to larger components that require depth either to assist signal processing algorithms or for display.

Future plans

Greater functionality will be incorporated as the object finds use a wider range of applications.

This page intentionally left blank.

Sommaire

An IDL Object for Managing Gridded Bathymetry Files

A.1 W.A. Roger; DRDC Atlantic TM 2005-265; R & D pour la défense Canada – Atlantique; March 2006.

Introduction

Dans le cadre du projet de démonstration de technologies de guerre sous-marine en réseau, on construit un système expérimental de combat pour examiner les problèmes associés au partage des données de détection entre les plates-formes de guerre anti-sous-marine (ASM) coopérantes. Le système de combat comprend une composante océanographique tactique appelée le gestionnaire de modélisation de l'environnement (EMM), qui fournit des prévisions des conditions acoustiques de l'océan à des aides à la prise de décisions tactiques. L'élément central du EMM est un moteur de prévision acoustique qui nécessite des données précises relatives à la profondeur de l'océan suivant une partie d'une route orthodromique. De plus, on a besoin, dans le cadre du projet, de données bathymétriques couvrant une zone rectangulaire, souvent utilisées pour la planification du déploiement des navires au cours d'essais en mer.

Le présent document décrit un objet logiciel qui fournit les profondeurs du fond océanique dans une variété de formats. Cet objet est écrit en langage IDL (Interactive Data Language), ce qui facilite l'utilisation et permet une génération efficace de code.

Résultats

L'objet lit un fichier de données bathymétriques maillées et fournit des données de profondeur au programme de contrôle. Les profondeurs océaniques peuvent être demandées pour un seul point de carte, suivant une ligne de relèvement jusqu'à une certaine distance et pour une zone rectangulaire. Pour leur part, les données peuvent être affichées ou fournies à des algorithmes de traitement des signaux et de prévision acoustique.

Importance

L'objet bathymétrique constitue un moyen pratique et souple pour l'acquisition de données de profondeur océanique à des points particuliers. Il est utilisé comme élément de l'infrastructure pour de plus gros ensembles qui nécessitent des données de profondeur pour appuyer les algorithmes de traitement des signaux ou à des fins d'affichage.

Perspectives

Une plus grande fonctionnalité sera intégrée à mesure que l’objet sera utilisé dans une gamme plus étendue d’applications.

Table of contents

List of figures

List of tables

This page intentionally left blank.

1. Introduction

The Networked Underwater Warfare (NUW) Technology Demonstration Project (TDP) has a need for precise ocean bottom topography of the Scotian Shelf area. The depth of the ocean at particular locations is utilized by a number of technologies to either predict future ocean conditions, or for display in research-level combat systems supporting tactical algorithms. The immediate need involves providing ocean depths to aid a sophisticated acoustic prediction system called the Environment Modeling Manager [1], [2], [3]. This software package employs a client/server system to disseminate predicted transmission loss, signal excess, sound speed profiles, and a host of other information to registered client programs. The model behind the server is called Bellhop [4], which can provide acoustic predictions that depend on range. This in turn requires an estimate of ocean depth along the range being sampled, since acoustic energy can be markedly affected by interaction with the bottom.

Bathymetry files can store the depth information in a number of formats, including *irregular*, where the depth values are provided only at the locations of the measurements, and *gridded*, where depths are interpolated onto a rectangular grid of locations. More specifically, for gridded data the sea bottom depths are provided along lines of latitude and longitude at a constant increment of angle. This is illustrated in Figure 1, where values of the water depth are provided at each of the locations where the lines intersect. In this particular file the separation of the depth values is 0.005476…º which translates into 0.005476…º x 60 x 1852 = 608.5 metres in a North/South direction (1º of latitude represents 60 nautical miles and there are 1852 metres per nautical mile).

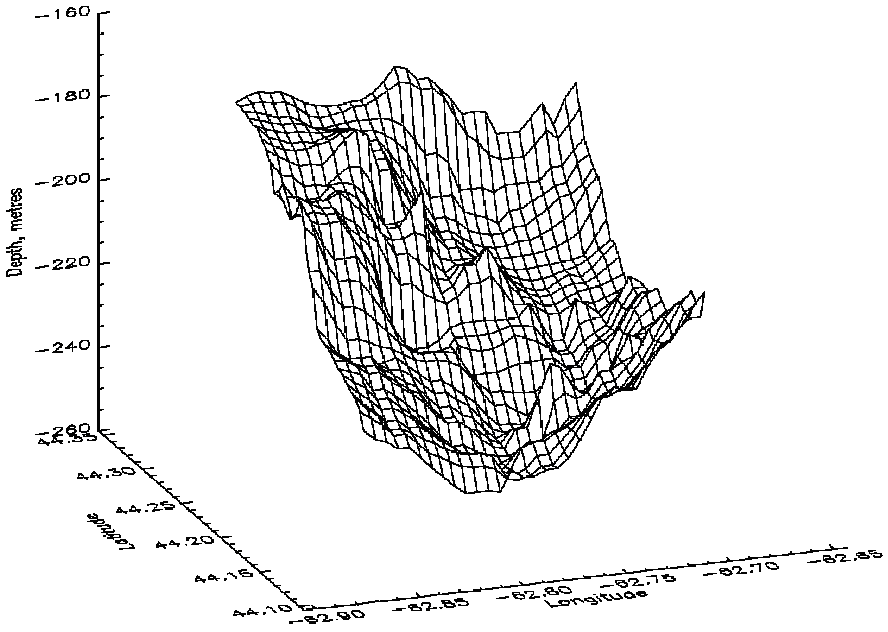


Figure 1: A small portion of a gridded bathymetry file. The distance between latitude grid points is approximately 600 metres.

This guide describes a software package that reads gridded depth files and provides the information for further processing and for display. The code is written in the Interactive Data Language (IDL) [5], a high-level programming language and software development environment that is designed to handle large data sets. It facilitates the efficient processing and display of information with minimal coding. In addition, IDL supports the concept of software objects that allows control over which aspects of the data are exposed to the "outside world". The bathymetry reader is cast as a single object that will read the data file and present the water depth information in a variety of formats through object methods.

The following section specifies the required file format, and then subsequent sections describe the available methods for obtaining the bathymetric data. It is assumed that the reader is familiar with the IDL programming language and development environment.

2. Bathymetry File Format

The IDL object reads two files when acquiring the depth information, with both files having the same name but differing extensions, namely, .bin and .hdr. The *hdr* file is a simple text file containing a description of the bathymetry parameters, while the *bin* file holds a two dimensional array of depth values. The *hdr* file consists of a set of lines with each line containing a command name, whitespace (space character(s) or tab(s)), and a value. Only the first four characters of each command are parsed, so subsequent characters can be optionally added for readability if desired. In addition, the ordering of the commands in the file is not important, and either upper or lower case can be used. The required commands are listed next and an example *hdr* file is shown in Figure 2:

1) NCOL: Specify the number of longitude values in the accompanying *bin* file.
   Command format: NCOL value
   Example: ncolums 5230

2) NROW: Specify the number of latitude values in the accompanying *bin* file.
   Command format: NROW value
   Example: nrows 3250

3) WestEdge: Specify the west-most longitude value in the accompanying *bin* file.
   Command format: WEST value
   Example: WestEdge -75.000

4) SouthEdge: Specify the south-most latitude value in the accompanying *bin* file.
   Command format: SOUT value
   Example: South 45.000

5) LatDelta: Specify the constant angular separation between latitudes in the accompanying *bin* file.
   Command format: LATD value
   Example: LatDel 0.00034567

6) LonDelta: Specify the constant angular separation between longitudes in the accompanying *bin* file.
   Command format: LOND value
   Example: LonDel 0.00012345

The following commands are optional, and can be included to provide further information about the file. Any command that is not "understood" by the parser is ignored, so further information can be added to the file.

7) NODATA: Specify the number that indicates no valid information at a particular location in the accompanying *bin* file.
   Command format: NODA value
   Example: NODATA_VALUE -9999

8) ENDIAN: Specify the endian for the values in the accompanying *bin* file. Note that the bathymetry object will automatically order the endian to the current machine.
   Command format: ENDI value
   Example: endian big

```
ncols         5845
nrows         2375
WestEdge     -72.504852983448
SouthEdge     39.49750113458
LatDelta      0.0054763357211343
LonDelta      0.0054763357211343
NODATA_value  -9999
byteorder     MSBFIRST
```

Figure 2: An example hdr file.

The accompanying *bin* file is a binary file containing an array of single precision floating point numbers. Each value in the array represents a depth, with the first value in the file representing the depth at the south-most latitude and west-most longitude. Note that this represents a minimum latitude and minimum longitude, since locations in the southern hemisphere and the western hemisphere are represented by negative latitudes and longitudes (when expressed in decimal degrees). The second value in the file is the depth for the next longitude in the grid, but the same latitude. The array can be represented by Array(LongitudeIndex,LatitudeIndex), with LongitudeIndex incrementing most rapidly. The last value in the file is the depth for the maximum (north-most) latitude and maximum (east-most) longitude. Obviously, there are NCOL longitude indices in the array, and NROW latitude indices.

The bathymetry object first reads the *hdr* file to obtain the size of the depth array, then reads the *bin* file and stores the depths in memory for quick retrieval.

3. IDL Object Methods

This section describes how to instantiate and destroy the IDL Bathymetry object, and how to obtain the bathymetric values. All actions take place within the Interactive Data Language Development Environment (IDLDE), a multi-document graphical user interface that has a command line area, an output log area, and graphical display windows. Before using the bathymetry object, it is necessary to inform the IDLDE where the code is located. Simply add the appropriate folder to the list of search paths in the *File>Preferences* menu command or through the !PATH system variable [6].

3.1 Managing the IDL Bathymetry Object

3.1.1 Creating the IDL Object

Within a program, the Bathymetry object is created through the command:

```
BathyObj = obj_new( 'BATHYMETRY', FILENAME=FileName, /APPLE_COMPUTER )
```

where the optional keyword /APPLE_COMPUTER is only needed for Macintosh computers that expect the data to be in Big Endian format. The path to the *bin* file is specified through the variable FileName or equivalently in a text string, and the *hdr* file must be in the same folder. Finally, one can choose any name for the object; here BathyObj will be used simply for convenience. During creation of the object, the two files are read into memory for quick retrieval.

Example: BO = obj_new('BATHYMETRY', FILENAME='C:\test\ScotianShelfBathy.bin')

3.1.2 Destroying the Object

Before terminating the main program, or when access to the bathymetry data is complete, destroy the object and free memory resources by issuing the command:

```
obj_destroy, BathyObj
```

where the object name should replace BathyObj.

3.2 Returning Averaged Depths

Before proceeding with descriptions of the available object methods, this section will discuss the technique for obtaining a local average of depth. This averaging was implemented to provide smoothly varying bathymetry values and to eliminate sudden shifts in closely-spaced depths, particularly when the locations have a smaller separation than the gridded data. The technique chosen is called bilinear interpolation, which stands between the simplicity of a nearest-neighbour selection and the complexity of algorithms that involve first- and perhaps second-order derivatives. The bilinear averaging is accomplished through a simple weighted sum of the four closest gridded points, based on distance from the desired position. This is illustrated in Figure 3

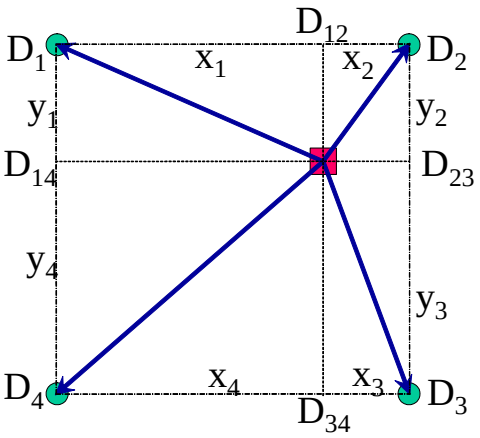


Figure 3: Using the four closest gridded depths to obtain an averaged depth at the location of interest.

where the distance between the desired location, represented by the red square, and each grid location (green circles) is used to compute the weights utilizing the *x* and *y* components. Using linearity considerations, the weighted average of the depths along each edge of the square, whose projection intersects the desired location, are computed. Next, the weighted average of these intermediate values provides the final depth value[1]:

$$
\begin{aligned}
D_{12} &= D_1 \cdot \left[\frac{x_2}{x_1 + x_2} \right] + D_2 \cdot \left[\frac{x_1}{x_1 + x_2} \right] \\
D_{34} &= D_4 \cdot \left[\frac{x_3}{x_3 + x_4} \right] + D_3 \cdot \left[\frac{x_4}{x_3 + x_4} \right] \\
D_{14} &= D_1 \cdot \left[\frac{y_4}{y_1 + y_4} \right] + D_4 \cdot \left[\frac{y_1}{y_1 + y_4} \right] \\
D_{23} &= D_2 \cdot \left[\frac{y_3}{y_2 + y_3} \right] + D_3 \cdot \left[\frac{y_2}{y_2 + y_3} \right] \\
D_x &= D_{12} \cdot \left[\frac{y_3}{y_2 + y_3} \right] + D_{34} \cdot \left[\frac{y_2}{y_2 + y_3} \right] \\
D_y &= D_{14} \cdot \left[\frac{x_3}{x_3 + x_4} \right] + D_{23} \cdot \left[\frac{x_4}{x_3 + x_4} \right] \\
D &= \frac{D_x + D_y}{2}
\end{aligned}
\tag{1}
$$

where D_i represents the depth at each of the four grid points, x_i and y_i are the horizontal and vertical distances from the desired location to each grid point, D_{ij} are the four weighted averages along each inter-grid axis, and D_x and D_y are the x and y components of the final depth, D. These

[1] Thanks to Anthony Isenor for informative discussions and advice on this problem.

equations can be simplified by working in "index" space rather than with the latitude and longitude values. Each depth in the 2D bathymetry array is identified by an x and y index value, and each index increments by one for successive grid points in the array. Hence, all the sums in the denominators (x_i+x_j and y_i+y_j) equate to 1 and $x_2=1-x_1$, $x_4=1-x_3$, $y_2=1-y_3$, and $y_4=1-y_1$. In addition, from the square in Figure 3 it is obvious that $x_1 = x_4$, $y_3 = y_4$, and so on. The equations then simplify to:

$$\begin{aligned} D_{12} &= D_1 \cdot [1 - x_4] + D_2 \cdot x_4 \\ D_{34} &= D_4 \cdot [1 - x_4] + D_3 \cdot x_4 \\ D_{14} &= D_4 \cdot [1 - y_4] + D_1 \cdot y_4 \\ D_{23} &= D_3 \cdot [1 - y_4] + D_2 \cdot y_4 \\ D_x &= D_{34} \cdot [1 - y_4] + D_{12} \cdot y_4 \\ D_y &= D_{14} \cdot [1 - x_4] + D_{23} \cdot x_4 \\ D &= \frac{D_x + D_y}{2} \end{aligned} \tag{2}$$

and substitution yields the bilinear formula [7]:

$$D = D_4 + [D_3 - D_4] \cdot x + [D_1 - D_4] \cdot y + [D_4 - D_3 - D_1 + D_2] \cdot xy \tag{3}$$

Consider the case where the location of the requested depth is identical to the location at D_4; then the distances x_4 and y_4 are zero which results in $D_{12} = D_1$, $D_{34} = D_4$, $D_{14} = D_4$, $D_{23} = D_3$, $D_x = D_{34} = D_4$, $D_y = D_{14} = D_4$, and the averaged depth $D = (D_4 + D_4)/2 = D_4$ as expected. Similarly if the location falls on D_2 then $D_{12} = D_2$, $D_{34} = D_3$, $D_{14} = D_1$, $D_{23} = D_2$, $D_x = D_{12} = D_2$, $D_y = D_{23} = D_2$, and $D = (D_2 + D_2)/2 = D_2$.

As an example, consider the request for a depth that equates to the index values x=200.5 and y=345.5. The surrounding depths are found at indices [200,345] for D_4, [201,345] for D_3, [201,346] for D_2, [200,346] for D_1, with x_4=200.5 – 200 = 0.5, and y_4 = 345.5 – 345 = 0.5. If the depths are D_1 = 120 m, D_2 = 125 m, D_3 = 135 m, and D_4 = 110 m, then solving Eqn. 2 yields:

D_{12} = 122.5 m,
D_{34} = 122.5 m,
D_{14} = 115.0 m,
D_{23} = 130.0 m,
D_x = 122.5 m,
D_y = 122.5 m, and the returned depth is
D = 122.5 m.

This method was chosen since it is relatively simple but still returns the actual gridded depth whenever the requested location overlies a grid position, and there is no jump as the desired location slips from one set of closest grid points to the next set. Inside the square of closest grid points, all the locally averaged depth values form a hyperbolic paraboloid, which can be viewed as akin to a rubber sheet stretched across a square wire frame. This is illustrated in Figure 4

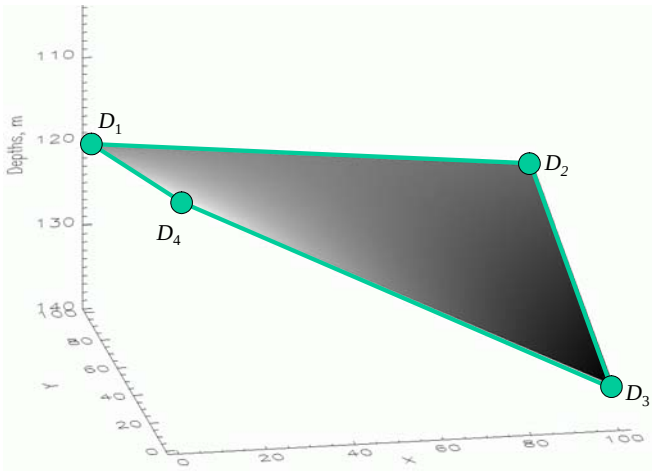


Figure 4: Surface of locally averaged depth values for each location within the square of closest gridded depths.

where the grid points are labelled as in Figure 3 and the variation in grey colouring indicates depth.

Many alternative local averaging techniques exist [7], from simple methods such as using the closest grid point or forming a plane from the closest points, up to biquintic interpolation that involves 36 terms. The more complicated interpolation schemes provide values that are more accurate when the underlying surface is a smooth curve [7], such as Gaussian peak, saddle, sine wave, or part of a sphere, etc. However, the ocean bottom tends to be irregular, with no guarantee that more complicated algorithms yield correspondingly better results.

3.3 Obtaining Depths at Individual Locations

This section describes how to obtain ocean depths at specified map positions.

3.3.1 The *Depth* Method

The *Depth* method accepts as input one or more latitude/longitude pairs and returns the corresponding averaged depth(s). A single latitude/longitude pair is represented as a 2-column vector, such as [45.0D, -63.0D], with the D indicating IDL double precision values and the operator [..] creating a vector from the interior values. The first number always represents the latitude and the second value the longitude. When obtaining the depths at several locations, the latitude/longitude pairs are arranged vertically as a 2 column by *N* row array. The depth values are returned as a 1 column by *N* row vector.

Method Syntax: Variable = BathyObj->Depth(LatLongVector)

where LatLongVector = [latitude, longitude] or [[latitude1, longitude1]
[latitude2, longitude2]
……..
[latitudeN, longitudeN]]

Example: Depths = BathyObj->Depth([45.55D, -63.22D])

3.4 Obtaining Depths along a Line of Bearing

In many cases the depths along part of a great circle are needed. For example, in EMM the Bellhop model accepts the ocean depths at equal increments along a constant bearing, and returns the transmission loss. The methods in this section provide this functionality, where the desired locations can be specified in a number of formats.

Note that lines of constant bearing on the earth's surface (assuming a spherical earth) are called *rhumb* lines, while *great circles* are the shortest distance between two points. Except for the meridians and the equator, they are not the same. Over long distances rhumb lines gradually spiral towards the closest pole. For the short distances being used here (relative to the earth's circumference), they will be treated as equal so requests for a "line of constant bearing" will generate the corresponding great circle.

3.4.1 The *SetLineOfBearing* Method

The *BathyObj->SetLineOfBearing* method is used to define a great circle along which the depths will be returned. The arguments are the start latitude/longitude, the end latitude/longitude, and the number of values to be returned:

Method Syntax:
BathyObj->SetLineOfBearing, StartOfLineLatLong, EndOfLineLatLong, NumberOfDepths

where *StartOfLineLatLong* and *EndOfLineLatLong* are 2-column vectors of latitude and longitude, and *NumberOfDepths* is a scalor.

Example: BathyObj->SetLineOfBearing, [45.0, -63.0], [46.0, -63.0], 61

Here a line of bearing heading due North has been specified, with values at each minute starting at 45º 0´N latitude, e.g. at 45º 0.0´N, 45º 1.0´N, 45º 2.0´N … 45º 59.0´N, and 46º 0.0´N.

3.4.2 The *SetLineOfBearing1* Method

This method provides an alternative way to specify the great circle line, using instead a start location, bearing, and range:

Method Syntax:
BathyObj->SetLineOfBearing1, StartOfLineLatLong, Bearing, RangeResolutionInNM, NumberOfRangeValues

where again *StartOfLineLatLong* is a 2-column vector of latitude and longitude, *Bearing* is the direction relative to North as a scalar value, *RangeResolutionInNM* is the distance between depth values expressed in nautical miles, and *NumberOfRangeValues* is a scalar indicating the total number of locations for which depth is required.

Example: BathyObj->SetLineOfBearing1, [45.0, -63.0], 0.0, 1.0, 61

This example will yield the same vector of locations as in the preceding section, namely, at 45º 0.0´N, 45º 1.0´N, … 45º 59.0´N, and 46º 0.0´N.

3.4.3 The *LineOfBearingDepths* Method

Once the line of bearing has been defined through either of the preceding *SetLineOfBearing* methods, this routine can be called to return the depths.

Method Syntax: Variable = BathyObj->LineOfBearingDepths()

where the returned depths are in a vertical column vector, one for each location along the line of bearing.

Example: D = BathyObj->LineOfBearingDepths()

Alternatively, the *SetLineOfBearing* and *LineOfBearingDepths* methods can be combined by using the optional arguments shown here:

Method Syntax: Variable = BathyObj->LineOfBearingDepths(StartOfLineLatLong, EndOfLineLatLong, NumberOfDepths)

Example: D = BathyObj->LineOfBearingDepths([45.0, -66.4], [45.5, -67.0], 50)

3.4.4 The *LineOfBearingLatLongs* Method

Once the line-of-bearing has been defined through the *SetLineOfBearing* or the *LineOfBearingDepths* methods, the individual latitude/longitude values can be obtained through the *LineOfBearingLatLongs* method:

Method Syntax: Variable = BathyObj-> LineOfBearingLatLongs ()

Example: LL = BathyObj-> LineOfBearingLatLongs ()

where the variable *LL* will be a 2-column by *N* row array containing the *NumberOfDepths* latitude/longitude pairs.

One can request just the latitudes or just the longitudes by adding the keywords /LATITUDES_ONLY or /LONGITUDES_ONLY, for example

LL = BathyObj-> LineOfBearingLatLongs (/LATITUDES_ONLY)

LL = BathyObj-> LineOfBearingLatLongs (/LONGITUDES _ONLY)

and the returned vector in *LL* will be a single column by *N* rows.

3.4.5 The *LineOfBearingLatitudes* and *LineOfBearingLongitudes* Methods

Once the line-of-bearing has been defined through the *SetLineOfBearing* or the *LineOfBearingDepths* methods, these two methods make the request for just latitudes or just longitudes explicit, rather than having to add keywords to the *LineOfBearingLatLongs* method:

Method Syntax:
```
Lats = BathyObj-> LineOfBearingLatitudes()
Lons = BathyObj-> LineOfBearingLongitudes()
```

where again the variables *Lats* and *Lons* are filled with the latitudes and longitudes as single column by *N* rows vectors.

3.4.6 An Example

Since the meridians follow great circle routes, a simple illustration of the *LineOfBearing* routines would be to obtain depths along one such line. A simple IDL program requested the depths along a line-of-bearing oriented due North between latitudes 43ºN and 44ºN and longitude 63ºW, with a total of 11 evenly spaced values between the end-point latitudes. This arrangement should generate a great circle sequence with a constant longitude and latitudes space by 0.1º. At the same time the expected set of latitudes and longitudes were generated, and the corresponding depths found through the *Depth* command. Figure 5 shows the program code, and Figure 6 presents the comparison results between the expected and received depth values.

Code	*Comments*
`BathyObj = obj_new( 'BATHYMETRY')`	Create object.
`BathyObj->SetLineOfBearing,[43.00D,-63.00D], [44.00D,-63.00D], 11`	Specify bearing line.
`Lats = BathyObj->LineOfBearingLatitudes()`	Obtain latitudes.
`Longs = BathyObj->LineOfBearingLongitudes()`	Obtain longitudes.
`Depths = BathyObj->LineOfBearingDepths()`	Obtain depths.

`ExpecedLatLongs = [ [43.00D, -63.00D], $`
`[43.10D, -63.00D], $`
`[43.20D, -63.00D], $`
`[43.30D, -63.00D], $`
`[43.40D, -63.00D], $`
`[43.50D, -63.00D], $`
`[43.60D, -63.00D], $`
`[43.70D, -63.00D], $`
`[43.80D, -63.00D], $`
`[43.90D, -63.00D], $`
`[44.00D, -63.00D] ]`
	Specify the correct latitude/longitude values along the 63ºW meridian.

`print, 'Lat/Longs:'`
`print, '  LineOfBearing  Expected`
`print, [ [ Lats, Longs ],  ExpecedLatLongs ]`
	Print line-of-bearing and reference lat/longs.

`print, 'Depths:'`
`print, '  LineOfBearing  Expected '`
`print,  [ Depths, BathyObj->Depth( ExpecedLatLongs ) ]`
	Print line-of-bearing and reference depths.

`obj_destroy, BathyObj`	Destroy object.
`end`	

Figure 5: Small program to illustrate use of the LineOfBearing methods.

Printout from Program	*Comments*
Lat/Longs: LineOfBearing Expected 43.000000 -63.000000 43.000000 -63.000000 43.100000 -63.000000 43.100000 -63.000000 43.200000 -63.000000 43.200000 -63.000000 43.300000 -63.000000 43.300000 -63.000000 43.400000 -63.000000 43.400000 -63.000000 43.500000 -63.000000 43.500000 -63.000000 43.600000 -63.000000 43.600000 -63.000000 43.700000 -63.000000 43.700000 -63.000000 43.800000 -63.000000 43.800000 -63.000000 43.900000 -63.000000 43.900000 -63.000000 44.000000 -63.000000 44.000000 -63.000000	Printout comparing the latitudes and longitudes from the *LineOfBearing* routines with the expected results.
Depths: LineOfBearing Expected -128.21222 -128.21220 -124.25032 -124.25032 -120.52783 -120.52784 -113.99213 -113.99242 -101.34351 -101.34352 -117.08078 -117.08071 -232.60283 -232.60269 -252.64895 -252.64889 -261.04218 -261.04220 -246.70158 -246.70153 -223.38973 -223.38973	Printout comparing the depths from the *LineOfBearingDepth* routine with the expected results.

Figure 6: Results from small program in Figure 5, showing how the received latitudes, longitudes, and depths match the expected values.

3.5 Obtaining Depths Inside a Box

Often it is desired to obtain a rectangle of bathymetry values from a window of a specific size. This is accomplished using the following set of routines.

3.5.1 The *DefineBox* Method

The window of depths is first defined by specifying the two opposite corners of the box. It does not matter whether this is the top-left/bottom-right or top-right/bottom-left corners, since the routine orders the values internally. The *DefineBox* method has two arguments that hold the latitude/longitude pairs for the opposite corners:

Method Syntax: BathyObj-> DefineBox, LatLongOfOneCorner, LatLongOfOppositeCorner

where again the variables *LatLongOfOneCorner* and *LatLongOfOppositeCorner* each contain a latitude/longitude pair.

Example: BathyObj-> DefineBox, [45.0, -66.0], [46.0, -65.0]

where the indicated box has a distance of 60 nautical miles in a north/south direction (60 nautical miles = 1º) and ~42 nm in an east/west direction.

3.5.2 The *DepthsInBox* Method

With the box defined, the depths can be returned using the *DepthsInBox* method:

Method Syntax: Variable = BathyObj-> DepthsInBox(DECIMATION_FACTOR=n)

where the parameter *Variable* contains a 2D array of depths of all the bathymetry values within the defined box. The optional keyword *DECIMATION_FACTOR* can be used if not all the gridded values are needed. Setting this keyword to a positive integer value *n* will cause only every n^{th} depth value to be returned, in both the north/south and east/west directions.

Example:
```
Depths = BathyObj-> DepthsInBox()
Depths = BathyObj-> DepthsInBox( DECIMATION_FACTOR=3 )
```

with the second form returning every 3^{rd} depth value.

If desired, one can combine the *DefineBox* and *DepthsInBox* operations by supplying the opposite corner arguments to the *DepthsInBox* method:

Method Syntax:
```
Variable = BathyObj-> DepthsInBox( LatLongOfOneCorner, LatLongOfOppositeCorner,
                                   DECIMATION_FACTOR=n )
```

Example:
```
Depths = BathyObj-> DepthsInBox( [45.0, -66.0],  [46.0, -65.0] )
```

3.5.3 The *BoxLatitude* and *BoxLongitude* Methods

To obtain the exact latitudes and longitudes for each depth value returned by the *DepthsInBox* method, invoke the methods *BoxLatitudes* and *BoxLongitudes*:

Method Syntax:
```
Variable = BathyObj-> BoxLatitudes()
Variable = BathyObj-> BoxLongitudes()
```

Example:
```
Lats = BathyObj-> BoxLatitudes()
Lons = BathyObj-> BoxLongitudes()
```

with the *Lats* variable containing a single column by N_{lat} rows of latitudes and the *Lons* variable similarly holding N_{long} rows of longitudes, corresponding to the N_{long} by N_{lat} array of depths returned by the *DepthsInBox* method.

3.6 Displaying the Depths

A number of methods are available to display the bathymetry provided by the box routines of the last section. The display uses the Mercator projection to map points on the earth's surface, expressed in latitude and longitude, onto a plane surface. If desired, lines of contour can be added and generic IDL plot commands can be used to overlay extra information. For example, results from the various lines of bearing methods can be overlaid on the display.

Table 1: Default colour table showing index assignment.

Colour Index	*Colour Range*	*Comments*
0 – 210	Navy to blue to white	Underwater colours
211 – 235	Dark green to light green	Above water colours
236 - 245	Dark yellow to light yellow	Contour line colours
246	Red	Labels, symbols and lines
247	Green	Labels, symbols and lines
248	Blue	Labels, symbols and lines
249	Yellow	Labels, symbols and lines
250	Purple	Labels, symbols and lines
251	Light Red	Labels, symbols and lines
252	Light Green	Labels, symbols and lines
253	Light Blue	Labels, symbols and lines
254	Light Yellow	Labels, symbols and lines
255	White	Labels, symbols and lines

3.6.1 Managing the Colour Table

Before proceeding with a description of the display options, a short discussion of the default colour table will be useful. The colour table is designed to accommodate both below and above water colours, a set of colours for contour plotting, and colours for over-plotting labels, symbols and lines. From an assumed range of 256 index values, the colour table has the layout shown in Table 1. Assigning an index value from this table to a method keyword will result in the corresponding colour being used.

Some methods allow specification of a colour table to replace various components of the default colour table. For the underwater component, the bathymetry object extracts 211 colours from the new colour table (spanning its full range) and these replace the default values between indices 0 and 210. In like manner the contour lines component and label component can be replaced with new colour tables. The replacement colours are obtained internally from the IDL command *loadct*, which has a selection of 41 colour tables from which to choose. The IDL reference [6] lists the available colour tables and their corresponding indices.

3.6.1.1 The *LoadColourTable* Method

The *LoadColourTable* method can be used to preload the colour table before plotting the bathymetry, or to serve as the colour table for other IDL windows. Its syntax is:

Method Syntax: BathyObj->LoadColourTable, WATERINDEX=n, CONTOURINDEX=n, LABELINDEX=n

Examples:
```
BathyObj->LoadColourTable
BathyObj->LoadColourTable, CONTOURINDEX=13
BathyObj->LoadColourTable, WATERINDEX=3, LABELINDEX=30
```

where all keywords are optional:

- *WATERINDEX=n*: Set this keyword to the desired index to replace the underwater component. If not specified the colour table with index *n*=1 is used (navy through blue to white).
- *CONTOURINDEX=n*: Set this keyword to the desired index to replace the range of indices used for colouring the contour lines.
- *LABELINDEX=n*: Set this keyword to the desired index to replace the set of indices that provide a selection of colours for subsequent plotting.

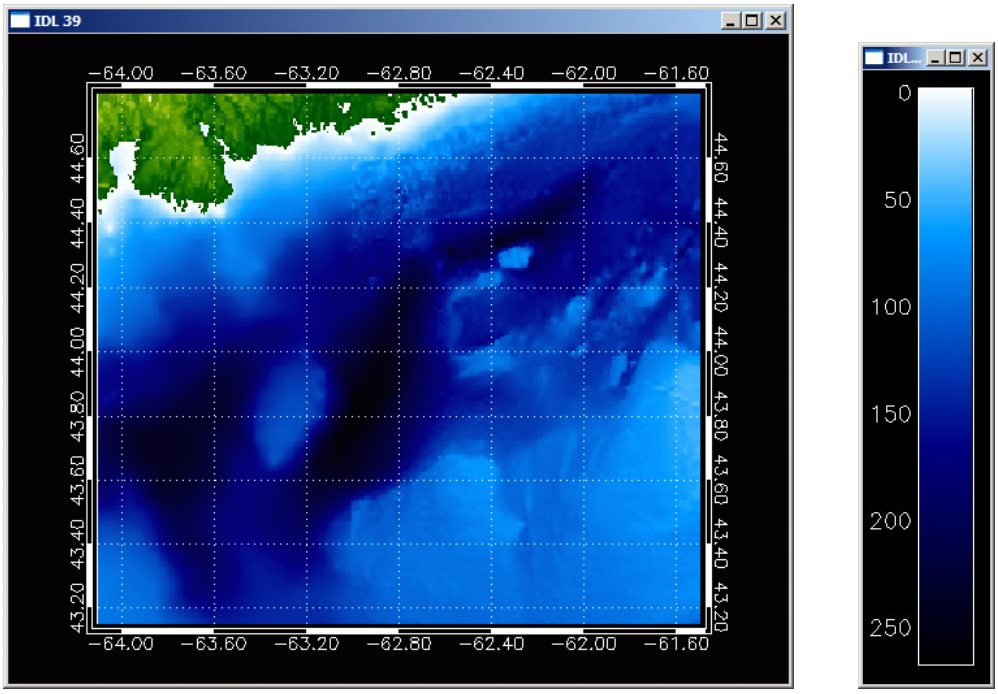
3.6.1.2 The *ShowColourTable* Method

The underwater component of the current colour table, and the associated depths, can be plotted to a separate window with the *ShowColourTable* method. It has the following format:

Method Syntax: BathyObj->ShowColourTable, WinWidth, WinHeight, DepthIncrementInM, CHARSIZE=CharSize

Examples:
```
BathyObj->ShowColourTable
BathyObj->ShowColourTable,  100, 500
BathyObj->ShowColourTable,  150, 450, 25
BathyObj->ShowColourTable,  200, 500, CHARSIZE=2
```

The following optional arguments are available:



(a) (b)

Figure 7: (a) An example Bathymetry display from the DisplayBoxDepths method. The display shows the Halifax peninsula in the top left corner and Emerald Basin near the centre.
(b) A colour table indicating the depths represented by each underwater colour.

- *WinWidth*: This argument sets the horizontal size of the window displaying the colour table. By default, the window will be 100 pixels wide.
- *WinHeight*: This argument sets the vertical height of the colour table window. By default, the window will be 500 pixels high.
- *DepthIncrementInM:* This argument sets the difference between successive depths in the colour table window. The default value is 50 metres, and this was used to obtain the colour table shown in Figure 7b.

The ShowColourTable method supports the following keyword:

- *CHARSIZE=n:* The default character size is 1.6. This keyword can be used to increase or decrease the font size of the depth labels.

The method first determines the width of the largest depth value, then adjusts the colour table image to fill the window remaining after the labels have been written. Consequently, changing the *WinWidth* value and/or the character size will alter the width of the colour table image.

3.6.2 The *DisplayBoxDepths* Method

This routine uses the currently defined box to generate a bathymetry plot. An example is shown in Figure 7a with Halifax harbour just visible in the top left corner, and Emerald Basin indicated by the darker blue colour in the centre of the display. The method has the following format:

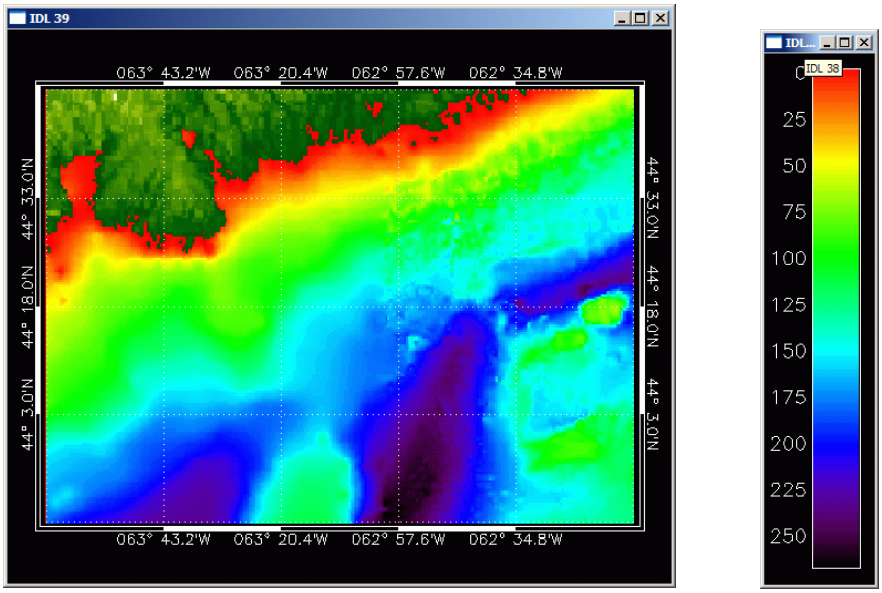
Method Syntax: BathyObj->DisplayBoxDepths, LatLongForOneCorner, LatLongForOppositeCorner, DecimationFactor, DEEPLIMIT=n, SHALLOWLIMIT=n, COLOURTABLEINDEX=n, /NOCOLOURTABLE, /RAINBOWCOLOR, /NOGRID, /EXACTGRIDAXES, /DEGMINUTES, CHARSIZE=CharSize

Examples:
```
BathyObj->DisplayBoxDepths()
BathyObj->DisplayBoxDepths([45.0, -66.0], [46.0, -65.0], 3 )
BathyObj->DisplayBoxDepths(COLORTABLEINDEX=3)
```

where all arguments and keywords are optional.

The arguments *LatLongForOneCorner*, *LatLongForOppositeCorner*, and *DecimationFactor* allow the box and decimation factor to be specified within the *DisplayBoxDepth* method without first defining the desired box with the *DefineBox* method. The keywords for this routine have the following actions:

- *DEEPLIMIT=n* and *SHALLOWLIMIT=n*: Normally the colour table representing the underwater topography spans the range from the deepest to shallowest depths within the box. This can be adjusted through the DEEPLIMIT and SHALLOWLIMIT keywords. By default, DEEPLIMIT is set internally to the deepest bathymetric value and SHALLOWLIMIT to the shallowest value (or 0.0 in the case when the box includes land). Setting either or both keywords to intermediate depths (specified as negative decimal numbers) causes all depths below the DEEPLIMIT to be assigned the first colour in the colour table, and all depths above the SHALLOWLIMIT to be assigned the last colour. This provides a mechanism to enhance the colour discrimination over a restricted range of depths.
- *COLOURTABLEINDEX=n*: The default colour table was described in Section 3.6.1, but the underwater component can be altered by specifying the desired index from the IDL command *loadct* [6]. An example is provided in Figure 8b where COLORTABLEINDEX=13 was specified. This index produces a “heat map” of colours ranging from navy to red. The IDL reference [6] lists the available colour tables and their corresponding indices.
- */NOCOLOURTABLE*: The default colour table (possibly with the underwater component replaced through the *ColourTableIndex* keyword) is automatically loaded within this method before plotting the bathymetry. Alternatively, the colour table can be preloaded with the *LoadColourTable* method or the IDL routines *loadct* or *tvlct*. Invoking this keyword then ensures no colour table is installed before plotting commences.
- */NOGRID:* By default, the bathymetry display includes lines of latitude and longitude along with the appropriate labels. If the latitude/longitude grid is not desired, add this keyword to the *DisplayBoxDepths* method.



(a) (b)

Figure 8: (a) An example bathymetry plot using the colour table identified by index 13.
(b) The depths represented by each underwater colour.

- *EXACTGRIDAXESLAT=n* and *EXACTGRIDAXESLONG=n*: As shown in Figure 7a, the default action is the presentation of latitude/longitude labels with "nice" values. Use either or both keywords to force the number of grid lines to an exact value. For example, *EXACTGRIDAXESLAT=5* and *EXACTGRIDAXESLONG=6* were added to the *DisplayBoxDepths* method shown in Figure 8a. This forced the program to generate 3 latitude and 4 longitude interior grid lines, plus the two along the respective edges of the plot.
- *DEGMINUTES=n*: Setting this keyword will force the labels to indicate the latitudes and longitudes in degrees and minutes, rather than in decimal degrees. The integer *n* specifies the number of digits in the minutes' value. As an example, the plot in Figure 8a was generated by adding *DEGMINUTES=1* to the *DisplayBoxDepths* method.
- *CHARSIZE=n*: The size of the labels can be controlled with the *CHARSIZE* keyword. The default size is set to 1.6, but increasing the value may produce a more pleasing label, especially when plotting to smaller windows.

3.6.3 The *DisplayBoxContours* Method

Contours can be added to a plot of bottom topography through the *DisplayBoxContours* method. It has the format:

Method Syntax: BathyObj->DisplayBoxContours, ContourDepths, C_CHARSIZE=n, C_COLORS=n, C_LABELS=n

Examples: BathyObj->DisplayBoxContours
BathyObj->DisplayBoxContours, [-50, -100, -150, -500], CHARSIZE=2

```
BathyObj->DisplayBoxContours   C_COLOURS=[230, 231, 232, 233]
BathyObj->DisplayBoxContours  C_LABELS=[1, 0, 1, 0, 1]
```

where all arguments and keywords are optional:

- *ContourDepths*: This optional argument sets the precise depths at which to draw the contours, expressed as negative values. However, for convenience positive numbers can be specified as the routine automatically changes them to the corresponding negative value. If the argument is not provided, the program calculates "nice" values to provide roughly five different contour values.
- *C_LABELS=n*: This keyword corresponds to the equivalent one in the IDL *contour* [6] command. It should be a vector of the same length as the depths specified in the *ContourDepths* argument. Insert the value "1" at those locations in the vector where it is desired to have that contour line labeled with the depth value. Insert the value "0" at those locations where the contour should be unlabeled. For example, if ContourDepths = [-50, -100, -150, -200] and C_LABELS=[0, 1, 0, 1] then only the 100 and 200 depth contours will be labeled. If this keyword is not set, all contours are labeled.
- *C_COLOURS=n*: This keyword corresponds to C_COLORS in the IDL command *contour*. Use this keyword to specify the colour table indices to be used in colouring the corresponding contour line and possible label. If not used, the yellow to orange indices at the top of the colour table are used, as described in Section 3.6.1.
- *C_CHARSIZE=n*: This is another keyword from the IDL *contour* command. Set the value *n* to a number greater than 1 to enlarge the labels in the contour lines.

3.6.4 Displaying Elements over the Bathymetry Plot

Once a bathymetry map has been displayed using the *DisplayBoxDepths* method, other information may be placed over the topography. The IDL routines *oplot* and *plots* can be used to add symbols or lines using latitudes and longitudes to specify location.

3.6.4.1 The *OPlotBox* Method

There is often a need to place a rectangle on the display to indicate some area of particular interest. This may indicate an exercise area for a sea trial, or for water management purposes during an experiment. The *OPlotBox* method displays a rectangle of arbitrary size on the bathymetry plot.

Method Syntax:
```
BathyObj->OPlotBox,  LatLongForOneCorner,  LatLongForOppositeCorner,
COLOUR=n, LINETHICKNESS=n
```

Examples:
```
BathyObj->OPlotBox, [45.33, -66.33], [44.12, -65.00]
BathyObj->OPlotBox, [45.33, -66.33], [44.12, -65.00], COLOUR=250
BathyObj->OPlotBox, [45.33, -66.33], [44.12, -65.00] , LINE=3, COL=254
```

where:

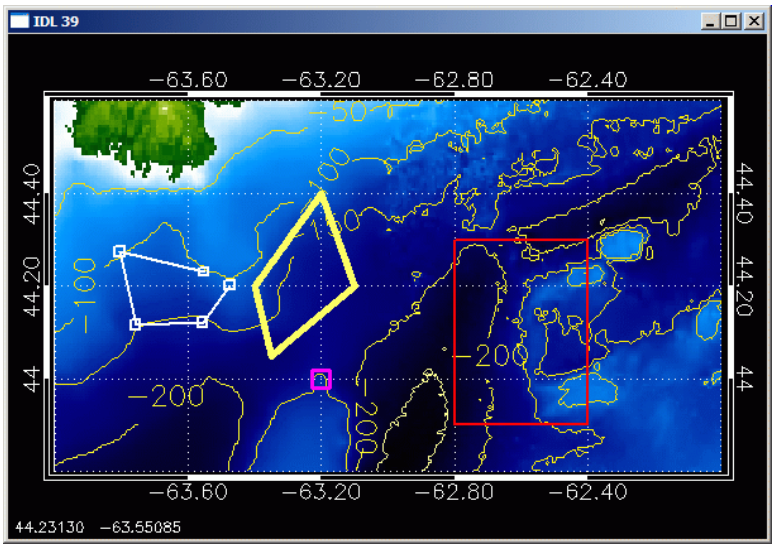


Figure 9: Screen capture of bathymetry plot showing use of the OplotBox, OPlotPolygon, and GetCursorLatLong methods, plus the IDL command 'plots'.

- *LatLongForOneCorner* and *LatLongForOppositeCorner*: These arguments define the opposite corners of the box, expressed as 2-element vectors of latitude and longitude.
- *COLOUR=n*: This optional keyword specifies the colour to be used when drawing the box. Table 1 provides a listing of the available colours for the default colour table. The default colour is white, with *n* = 255
- *LINETHICKNESS=n*: This optional keyword defines the width of the line used to draw the box. A value of one for *n* results in a thin, 1 pixel wide line, and values greater than about 3 produces very thick lines. The default line thickness is 1 pixel wide.

The red rectangle in Figure 9 was generated with *OPlotBox* command. Note that each side represents a portion of a great circle route.

3.6.4.2 The *OPlotPolygon* Method

The *OPlotPolygon* method will plot a polygon of arbitrary shape on the display. The command format is:

Method Syntax: BathyObj->OPlotPolygon, LatLongVertexVector, COLOUR=n, LINETHICKNESS=n, /CLOSEPOLYGON

Examples: BathyObj->OPlotPolygon, [[45.33, -66.33], [44.12, -65.00], [44.34, -64.80]]
BathyObj->OPlotBox, LatLongVertices, COLOUR=250, /CLOSE

where:

- *LatLongVertexVector*: This argument defines the vertices of the polygon, expressed as a 2-column by *N* row array. The first column contains the latitude values and the second column the longitudes.

- */CLOSEPOLYGON*: This optional keyword ensures the polygon is closed by copying the first vertex to the end of the array of vertices.
- *COLOUR=n*: This optional keyword specifies the colour index to be used when drawing the polygon. Select the index number by referring to Table 1; the default value is *n* = 255.
- *LINETHICKNESS=n*: This optional keyword defines the width of the line used to draw the box. The default line thickness is 1 pixel wide.

The yellow polygon in Figure 9 was generated with *OPlotPolygon* command. Note that each side represents a portion of a great circle route.

3.6.4.3 The *GetCursorLatLong* Method

Once a rectangle of depths has been plotted via the *PlotBoxDepths* method, the mouse can be used to extract points of interest. Calling the *GetCursorLatLong* method allows one to select specific latitude/longitude locations by depressing the left mouse button. Each time this occurs the latitude/longitude pair under the cursor is appended to a list. To terminate the process, click the right mouse button and the vector of positions is returned to the calling program. Optionally, the designated locations can be visually indicated on the chart and the current mouse location can be displayed at the bottom of the plot window. The method syntax is:

Method Syntax: Variable = BathyObj->GetCursorLatLong(/SINGLE, PAINTSYMBOL=n, SYMSIZE=n, COLOUR=n, LINETHICKNESS=n, /SHOWLOCATION)

Examples: LatLongs = BathyObj->GetCursorLatLong(/SINGLE)
LL = BathyObj->GetCursorLatLong(PAINTSYMBOL=6, SYMSIZE=2, /SHOWLOCATION)

where all keywords are optional:

- */SINGLE*: A single latitude/longitude pair is required. If this keyword is set, a left-mouse click will cause an immediate exit from the method with the indicated location returned in the variable.
- *PAINTSYMBOL=n*: Set this keyword to have an IDL plotting symbol indicate the location of each left mouse click. The desired symbol is selected by the value of *n*, with the choices shown in Table 2. The default is not to show any symbol.
- *SYMSIZE=n*: The size of each symbol can be specified by this keyword, with the default being *n*=1 representing a symbol roughly the size of a text character. This keyword is ignored if *PaintSymbol* is not present.
- *COLOUR=n*: This keyword specifies the colour to be used for each symbol, through the appropriate colour index as listed in Table 1. The default is the colour white (*n*=255). This keyword is ignored if *PaintSymbol* is not present.

Table 2: IDL plotting symbols and their corresponding indices. The user-defined symbol is specified with the USERSYM procedure.

Index	*IDL Plot Symbol*
1	+
2	*
3	•
4	Diamond
5	Triangle
6	Square
7	X
8	User-defined

- *LINETHICKNESS=n*: Invoking this keyword causes the location of each mouse click to be connected via a straight line, with a thickness specified by *n*. This keyword is ignored if *PaintSymbol* is not present.
- /*SHOWLOCATION*: When this keyword is present, the current location of the cursor is continuously updated in the lower left corner of the plot window. The latitude and longitude values are normally shown in decimal degrees unless the *PlotBoxDepths* method includes the *DegMinutes* keyword. In this case the position is indicated in degrees and minutes. Note the latitude/longitude values in the extreme lower left corner of the window shown in Figure 9.

The white polygon in Figure 9 was generated with this command.

3.6.4.4 Examples of Plot Overlays

The four overlay items shown in Figure 9 were produced with the code fragment that follows:

```
BathyObj->OPlotBox, [44.3D, -62.8D], [43.9D, -62.4D], LINE=2, COLOUR=246
PolyVertices =   [  [44.40D, -63.20D],  $
                    [44.20D, -63.40D],  $
                    [44.05D, -63.35D],  $
                    [44.20D, -63.10D]  ]
BathyObj->OPlotPolygon, PolyVertices, COLOUR=254,  LINE=5, /CLOSE
plots, -63.20D, 44.0D, PSYM=6, SYMSIZE=2, THICK=3, color=250
LL = BathyObj->GetCursorLatLong(PAINT=6, SYMS=1, /SHOWLOCATION, LINE=2 )
```

The *OPlotBox* created the red rectangle using the specified opposite corners and a colour index of 246. After the vertices are defined via the variable *PolyVertices*, the *OPlotPolygon* method generated the yellow polygon using colour index 254 and the */CLOSE* keyword to ensure a closed figure. The IDL *plots* routine used symbol index 6 (see Table 2) and colour index 250 (Table 1) to form the small purple square. Finally, the white polygon resulted from calling *GetCursorLatLong* while specifying that connected small squares (index 6) should indicate each mouse click. Following the right mouse click the variable LL was returned with the following latitude/longitude pairs: `[-63.474872, 44.201352], [-63.554648, 44.122321], [-63.755989, 44.116867], [-63.805375, 44.274838], [-63.550849, 44.231302].` In

turn this variable could be used with the *OplotPolygon* or other methods that accept a vector of latitude/longitude values.

3.7 Determining the File Boundaries and Grid Resolution

The routines in this section allow determination of the coverage of the bathymetry in the current file, and the distance between the gridded latitude/longitude depths.

3.7.1 The *LatitudeBounds* and *LongitudeBounds* Methods

The *LatitudeBounds* and *LongitudeBounds* methods return the edge locations of the bathymetry file, with the variable attached to *LatitudeBounds* receiving the minimum and maximum latitudes as a 2-element vector, and *LongitudeBounds* providing the minimum and maximum longitudes.

The methods have the following format:

Method Syntax: LatVariable = BathyObj->LatitudeBounds()
LonVariable = BathyObj->LongitudeBounds()
where LatVariable = [min. latitude, max. latitude]
and LonVariable = [min. longitude, max. longitude]

Examples: MinMaxLats = BathyObj->LatitudeBounds()
MinMaxLons = BathyObj->LongitudeBounds()

with, for example, MinMaxLats = [45.55, 55.0] and MinMaxLons = [-70.0, -50.0].

3.7.2 The *LatitudeResolution* and *LongitudeResolution* Methods

The difference between successive latitudes and longitudes can be determined by calling the methods *LatitudeResolution* and *LongitudeResolution*. The associated variable will hold a single double precision value representing the delta between grid points, expressed in decimal degrees.

The methods have the following format:

Method Syntax: LatVariable = BathyObj->LatitudeResolution(/MINUTES)
LonVariable = BathyObj->LongitudeResolution(/MINUTES)

Examples: LatsResolution = BathyObj->LatitudeResolution()
LonsResolution = BathyObj->LongitudeResolution()
LatsRes = BathyObj->LatitudeResolution(/Minutes)
LonsRes = BathyObj->LongitudeResolution(/Minutes)

with, for example, LatsResolutions = 0.0054763357 and LonsResolution = 0.005487. These values can be returned in degree minutes by using the optional */MINUTES* keyword.

3.8 Utility Methods

A few utility routines are included to allow extraction of depth values for further processing, and for plotting of bathymetry outside the included methods.

3.8.1 The *ReturnFullBathymetry* Method

The entire bathymetry map can be placed in a variable by using the *ReturnFullBathymetry* method. It has the following format:

Method Syntax: Variable = BathyObj->ReturnFullBathymetry()

Examples: Bathy = BathyObj->ReturnFullBathymetry()

3.8.2 The *MappedBathyImage* Method

This method returns the 2D image after the IDL *map_image* [6] routine has warped it from a spherical surface, expressed by latitude/longitude values, to a plane surface. For example, consider a window into the bathymetry table, such as shown in Figure 8a, with 300 longitude values and 200 latitude values. This image is represented by an array of size [300,200]. The *map_image* routine transforms this into another array of size [1250,935] (say) that will fill a plot on the screen of size 1250 pixels horizontally by 935 pixels vertically. Each value in the new array still represents depth in metres but the location of each depth has been converted from latitude/longitude units to pixel units.

This method has the following format:

Method Syntax: Variable = BathyObj->MappedBathyImage()

Example: x = BathyObj->MappedBathyImage()

This routine is provided primarily to allow alternate plotting of the bathymetry.

3.8.3 The *IndexedBathyImage* Method

This method returns the 2D image after the warped image (returned by *MappedBathyImage*) has been converted from depths in metres to colour table indices. It can be used with the IDL routine *tv* [6] to directly plot the bathymetry image to a window.

The method has the following format:

Method Syntax: Variable = BathyObj->IndexedBathyImage()

Examples: x = BathyObj->IndexedBathyImage()

3.8.4 The *ReturnLatIndex and ReturnLongIndex* Methods

The companion routines, *ReturnLatIndex* and *ReturnLongIndex*, convert specific latitude/longitude locations into the corresponding indices in the full bathymetry map. The methods have the following format:

Method Syntax: LatVariable = BathyObj->ReturnLatIndex(LatitudeVector, VALID=Variable)
LonVariable = BathyObj->ReturnLongIndex(LongitudeVector, VALID=Variable)

Examples:
```
LatsIndex = BathyObj-> ReturnLatIndex( 45.234 )
LonsIndics = BathyObj-> ReturnLongIndex([ [-66.00D], [-65.3D], [-55.33D] ])
i = BathyObj-> ReturnLatIndex([ [40.0D], [45.0D] ], VALID=IsValid)
```

where the VALID keyword is optional.

The arguments and keywords for these methods are:

- *Latitude:* The input latitude for which the index is desired. This input can be either a single value or a column vector of latitudes.
- *Longitude:* The input longitude for which the index is desired. This input can be either a single value or a column vector of longitudes.
- *VALID=Variable*: If this keyword is supplied, the variable will be returned with a value or vector that indicates whether the corresponding locations are within the bounds of the bathymetry map. A "1" represents a valid location and a "0" indicates a location outside the map area.

As an example, the depth at location 45ºN and 66ºW could be obtained through a code fragment like:

```
BathyMap = BathyObj->ReturnFullBathymetry()
LatIndex = BathyObj-> ReturnLatIndex(45.0)
LonIndex = BathyObj-> ReturnLongIndex(-66.0)
Depth = BathyMap[LonIndex, LatIndex]
```

4. Example Code for Exercising the Bathymetry Object

The IDL source code shown in Table 3 illustrates how to create and use the bathymetry object. The comments to the right describe the purpose of each line, and running the program will produce the displays shown in the following figures.

Table 3: Example source listing that illustrates the use of the IDL bathymetry object, and an accompanying explanation.

Line #	*Source Listing*	*Explanation*
	PRO TESTING_BATHY	
1	BathyObj = obj_new('BATHYMETRY')	Create the bathymetry object
2	print, 'Resolution: ', BathyObj->LongitudeResolution(/MINUTES)	Print the resolution of the grid
3	print, 'MAP EDGES: Latitude: ', BathyObj->LatitudeBounds()	Show the edges of the
4	print, 'MAP EDGES: Longitude: ', BathyObj->LongitudeBounds()	bathymetry in latitude and longitude
5	Corner1 = [43.15D, -64.1D]	Specify one corner of the box
6	Corner2 = [44.6D, -61.45D]	Specify the opposite corner
7	BathyObj->DefineBox, Corner1, Corner2	Define the box to be used
8	Lats = BathyObj->BoxLatitudes()	Obtain the gridded latitudes
9	Longs = BathyObj->BoxLongitudes()	Obtain the longitudes in the box
10	Map = BathyObj->DepthsInBox()	Obtain all depths within the box
11	MapIndices = bytscl(Map)	Set up colouring for the next plot
12	loadct, 8	Load the colour table to be used
12	window, /FREE, XSIZE=1000, YSIZE=800	Create a new window for plotting
14	shade_surf, Map, Longs, Lats, TITLE='Bathymetry', XTITLE='Longitude', YTITLE='Latitude', ZTITLE='Depth', AZ=10, CHARSIZE=2, SHADES=MapIndices, XSTYLE=1, YSTYLE=1	Show a surface plot of the box depths, with each depth coloured according to the depth value.
15	window, /FREE, XSIZE=1000, YSIZE=800	Create a new window
16	BathyObj->DisplayBoxDepths, EXACTGRIDAXESLAT=5, EXACTGRIDAXESLONG=6, DEGMINUTES=0, COLOURTABLEINDEX=8	Display the box depths on a latitude/longitude chart, with specific grid values.
17	BathyObj->DisplayBoxContours	Display contours over the plot
18	BathyObj->SetLineOfBearing, [43.88D, -63.04D], [43.51D, -62.51D], 100	Define a line of bearing with 100 latitude/longitude values
19	LatLongs = BathyObj->LineOfBearingLatLongs()	Obtain the 100 lat/long values
20	oplot, LatLongs[1,*], LatLongs[0,*], color=240, THICK=3	Plot the line of bearing on the bathymetry display
21	BathyObj->SetLineOfBearing, [43.51D, -62.51D], [43.51D, -61.98D], 100	Define another line of bearing
22	LatLongs = BathyObj->LineOfBearingLatLongs()	Retrieve the lat/long values
23	oplot, LatLongs[1,*], LatLongs[0,*], color=240, THICK=3	Plot line of bearing on bathymetry plot
24	BathyObj->ShowColourTable, 100, 500, 25, CHARSIZE=2.0	Display the corresponding color table
25	SinglePointLatLong = [44.0D, -64.0D]	Define a single location
26	print, 'Single Depth at Latitude: ', SinglePointLatLong[0], ' and Longitude: ', SinglePointLatLong[1], ' is ', BathyObj->Depth(SinglePointLatLong), ' metres'	Print the location and the corresponding depth at this location
27	obj_destroy, BathyObj	Destroy the bathymetry object when it is no longer needed.
28	print, 'DONE!'	
29	end	End procedure

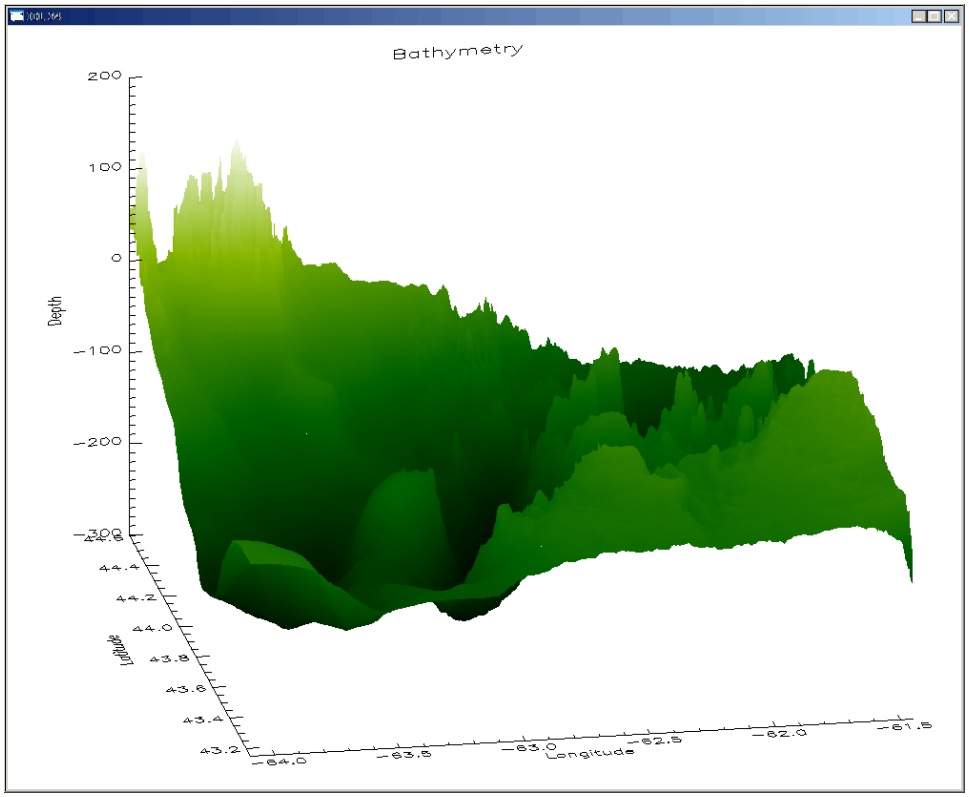


Figure 10: Surface plot using the IDL command Shade_Surf. Each bathymetry value is coloured according to its depth.

As this program is executed, line 14 containing the *shade_surf* command produces the plot shown in Figure 10, with the 2D bathymetry image represented by the *Map* variable being coloured to represent each depth. This colouring corresponds to that seen in the subsequent figures. Execution of line 16 results in a new plot showing the bathymetry chart form, as illustrated in Figure 11. Line 17 then generates the set of contour lines, and lines 18 to 23 define and plot the two lines of bearing that appear as straight lines in the figure. Figure 12 shows the window that results from execution of line 24. Finally, the output appearing in the logging window of the IDL DE is presented in Figure 13.

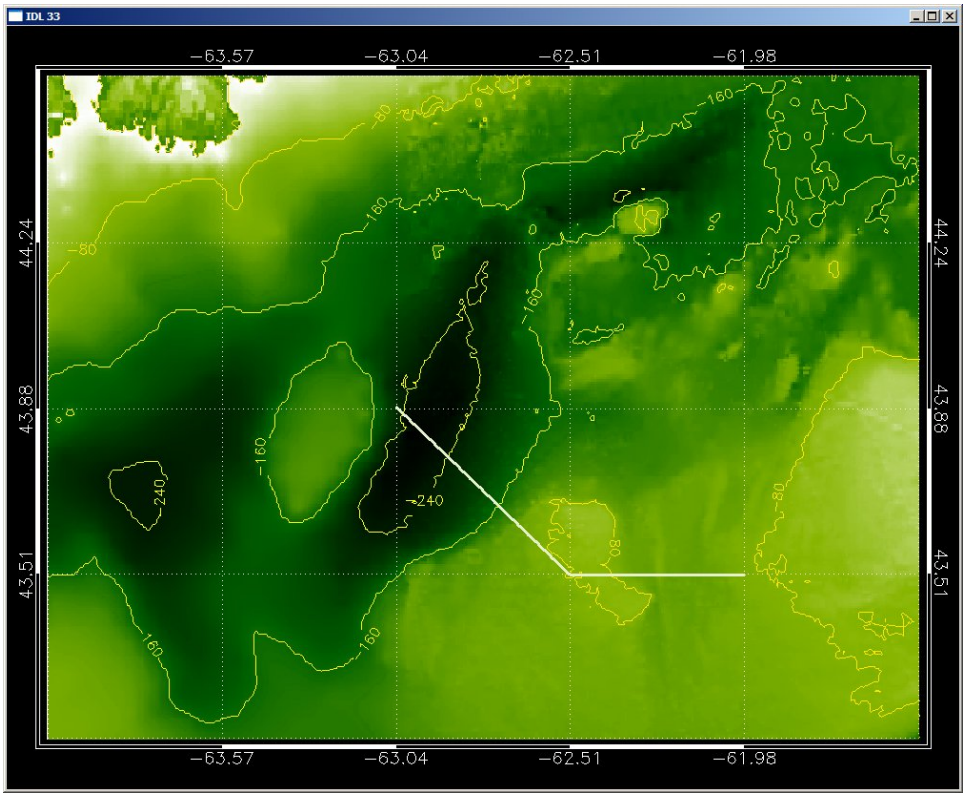


Figure 11: Bathymetry plot of the box area specified by the Corner1 and Corner2 variables. The lines of bearing defined in the program appear overlaid on the plot.

Figure 12: Screen capture of the colour table window and corresponding depth values.

```
IDL> testing_bathy
Resolution:    0.32858014
MAP EDGES:  Latitude:        39.497501       52.498322
MAP EDGES: Longitude:       -72.504853      -40.501147
Single Depth at Latitude:   44.000000 and Longitude:   -64.000000 is    -141.15723 metres
DONE!
```

Figure 13: Output in the log window of the IDL DE following execution of the procedure testing_bathy.

5. Summary

This document describes a small software object to extract and display underwater depths from a gridded bathymetry file. It is written in the IDL programming language to facilitate ease of use and rapid development of the controlling software. The object first reads the file into memory for quick response, then provides a number of software methods to obtain selected areas of depth, display all or portions of the bathymetry, and allow additions to the display such as contour lines and lines of bearing. In addition, the cursor can be used to obtain locations by clicking the mouse button over the bathymetry plot.

This object can be used in a range of oceanographic projects. During the planning of sea trials areas can be designated for water space management or for platform tracks based on issues with ocean depths. It can be used to feed acoustic models that provide range-dependent predictions, or to provide plots of ocean topography within documents.

References

[1] Scott MacDonald, *User's Guide for the Environment Modeling Manager*, DRDC Atlantic CR 2002-099, June 2002

[2] Scott MacDonald, *Enhancement of the Environment Modeling Manager*, DRDC Atlantic CR 2003-074, May 2003

[3] Sean Webb, *Enhanced Functionality to the Environment Modeling Manager*, DRDC Atlantic CR 2004-288, May 2005

[4] D. McCammon, *An Investigation of the Bellhop Acoustic Prediction Model*, DRDC Atlantic CR 2004-285, November 2005.

[5] Website for Research Systems Inc: http://www.rsinc.com/

[6] *Using IDL*, Research Systems Inc, IDL Version 6

[7] David Kidner, Mark Dorey and Derek Smith, "What's the point? Interpolation and extrapolation with a regular grid DEM", *Proceedings of the IV International Conference on GeoComputation*, 25-28 July 1999, Fredericksburg VA USA

Annex A. IDL Source Listings

The following sections list the source code for the bathymetry object. Note that the "$" symbol at the end of a line is the continuation character within IDL.

A.1 Constructor and Destructor Methods

A.1.1 Definition of the object structure

```
PRO  BATHYMETRY__DEFINE
  dummy2 = {BATHYMETRY_BOX_INDICES, South:0, North:0, West:0, East:0 }
  dummy1 = {BATHYMETRY_BOX, SouthLat:0.0D, NorthLat:0.0D, WestLong:0.0D, EastLong:0.0D, DecimationFactor:0,
            $
                        Index:{BATHYMETRY_BOX_INDICES}, Valid:0 }
  dummy3 = {BATHYMETRY_COLOURTABLE, Bottom:0, Top:0, Number:0}

  dummy = {BATHYMETRY,  LatGrid:ptr_new(), LongGrid:ptr_new(), NumLats:0, NumLongs:0, Bathymetry:ptr_new(),
            FileName:'', $
                  SouthLat:0.0D, NorthLat:0.0D, WestLong:0.0D, EastLong:0.0D, ShallowLimit:0.0, DeepLimit:0.0, $
                  LatDelta:0.0D, LatIndexScale:0.0D, LongDelta:0.0D, LongIndexScale:0.0D, ColourTable:0,  $
                  CurrentBox:{BATHYMETRY_BOX}, CurrentLineOfBearingLatLongs:ptr_new(),
            CurrentLOBLatLongsValid:0, $
                  MappedBathyImage:ptr_new(), IndexedBathyImage:ptr_new(), NoData:0.0, VectorLatLongs:ptr_new(),$
                  PlotInDegMin:0, PlotWinNumber:0, $
                  WaterTable:{BATHYMETRY_COLOURTABLE}, LandTable:{BATHYMETRY_COLOURTABLE}, $
                  ContourTable:{BATHYMETRY_COLOURTABLE}, LineTable:{BATHYMETRY_COLOURTABLE} }
end
```

A.1.2 The *Init* Constructor

```
FUNCTION  BATHYMETRY::INIT,   FILENAME=FileName,   APPLE_COMPUTER=AppleComputer

  if defined(FileName) then self.FileName = FileName else self.FileName = '..\..\..\Bathymetry Data
            File\AtlanticBathymetryPC.bin'
  hdrFileName = make_filename(self.FileName, '', '', '!.hdr')

  if file_exists( self.FileName ) eq !false then begin
    ok = dialog_message( self.FileName + string(13B) + string(10B) +  'Bathymetry file  not found.  Object creation failed.',
            /ERROR, TITLE= 'BATHYMETRY> FATAL ERROR')
    return, 0
  endif
  if file_exists( hdrFileName ) eq !false then begin
    ok = dialog_message( hdrFileName + string(13B) + string(10B) +  'Bathymetry Header file  not found.  Object creation
            failed.', /ERROR, TITLE= 'BATHYMETRY> FATAL ERROR')
    return, 0
  endif

  if keyword_set(AppleComputer) then begin
    openr, lunit1, hdrFileName, /GET_LUN, /SWAP_IF_LITTLE_ENDIAN
    openr, lunit2, self.FileName, /GET_LUN, /SWAP_IF_LITTLE_ENDIAN
    endif else begin
    openr, lunit1, hdrFileName, /GET_LUN, /SWAP_IF_BIG_ENDIAN
    openr, lunit2, self.FileName, /GET_LUN, /SWAP_IF_BIG_ENDIAN
  endelse
  Line = ''
  repeat begin
    readf, lunit1, Line
    Command = strUPcase( strsplit(Line, '  ', /EXTRACT))
    case strmid(Command[0],0,4) of
```

```
      'NCOL': self.NumLongs = fix(Command[1])
      'NROW': self.NumLats = fix(Command[1])
      'WEST': self.WestLong = double(Command[1])
      'SOUT': self.SouthLat = double(Command[1])
      'LATD': self.LatDelta = double(Command[1])
      'LOND': self.LongDelta = double(Command[1])
      'NODA': self.NoData = float(Command[1])
     else:
  endcase
 endrep until EOF(lunit1)

 tmpbathy = fltarr(self.NumLongs, self.NumLats)
 readu, lunit2, tmpbathy
 free_lun, lunit1, lunit2

 self.Bathymetry = ptr_new( rotate(tmpbathy, 7) )

 ;NOTE:  This has the array with tmpbathy[0,0] = [WestLong,SouthLat] (that is, [minvalue, minvalue] and opposite corner
 ;       is tmpbathy[max,max] = [EastLong, NorthLat]( i.e.: [maxvalue,maxvalue]).  So for displaying the bathymetry, the
           bathy array must
 ;       be inverted North <-> South, using rotate(bathy,7) that moves the top row to the bottom row, 2nd top to 2nd bottom,
           etc.  So,
 ;       longitudes are left intact, latitudes are inverted to get the smallest latitude (South) to the bottom for displaying)

 self.LongGrid = ptr_new( self.WestLong + dindgen(self.NumLongs) * self.LongDelta )
 self.LatGrid  = ptr_new( self.SouthLat + dindgen(self.NumLats)  * self.LatDelta )

 self.NorthLat = (*self.LatGrid)[self.NumLats-1]
 self.EastLong = (*self.LongGrid)[self.NumLongs-1]

 self.LatIndexScale  = 1.0D / self.LatDelta
 self.LongIndexScale = 1.0D / self.LongDelta
 self.CurrentBox.Valid = !false
 self.CurrentBox.DecimationFactor = 1
 self.CurrentLOBLatLongsValid = !false
 self.ColourTable = -1
 self.PlotWinNumber = -1

;--------------  Setting up colour table ------------------
 self.WaterTable.Bottom = 0
 self.WaterTable.Number = 211
 self.WaterTable.Top =   self.WaterTable.Bottom + self.WaterTable.Number - 1

 self.LandTable.Bottom = self.WaterTable.Top + 1
 self.LandTable.Number = 25
 self.LandTable.Top =   self.LandTable.Bottom + self.LandTable.Number - 1

 self.ContourTable.Bottom = self.LandTable.Top + 1
 self.ContourTable.Number = 10
 self.ContourTable.Top =   self.ContourTable.Bottom + self.ContourTable.Number - 1

 self.LineTable.Bottom = self.ContourTable.Top + 1
 self.LineTable.Number = 10
 self.LineTable.Top =   self.LineTable.Bottom + self.LineTable.Number - 1
 return, 1
end
```

A.1.3 The *CleanUp* Destructor

```
PRO  BATHYMETRY::CLEANUP
 ptr_free, self.Bathymetry, self.LongGrid, self.LatGrid, self.CurrentLineOfBearingLatLongs
 ptr_free, self.MappedBathyImage, self.IndexedBathyImage, self.VectorLatLongs
end
```

A.2 Methods Involving Depths for Specific Locations

A.2.1 The *Depth* Method

```
FUNCTION  BATHYMETRY::Depth,  LatLongVector
  LatIndex =  ( (self.LatIndexScale * (LatLongVector[0,*] - self.SouthLat) ) > 0.0 ) < (self.NumLats - 2)          ;with fractional
              part
  LongIndex = ( (self.LongIndexScale * (LatLongVector[1,*] - self.WestLong) ) > 0.0 ) < (self.NumLongs - 2)        ;with
              fractional part

  x = LongIndex - fix(LongIndex)
  y = LatIndex - fix(LatIndex)
  D12 = (*self.Bathymetry)[fix(LongIndex), fix(LatIndex+1.0)]*(1. - x) + (*self.Bathymetry)[fix(LongIndex+1.0),
              fix(LatIndex+1.0)]*x
  D34 = (*self.Bathymetry)[fix(LongIndex), fix(LatIndex)]*(1. - x) + (*self.Bathymetry)[fix(LongIndex+1.0), fix(LatIndex)]*x
  D14 = (*self.Bathymetry)[fix(LongIndex), fix(LatIndex)]*(1. - y) + (*self.Bathymetry)[fix(LongIndex), fix(LatIndex+1.0)]*y
  D23 = (*self.Bathymetry)[fix(LongIndex+1.0), fix(LatIndex)]*(1. - y) + (*self.Bathymetry)[fix(LongIndex+1.0),
              fix(LatIndex+1.0)]*y
  Dx = D34*(1. - y) + D12*y
  Dy = D14*(1. - x) + D23*x
  D = ( Dx + Dy ) / 2
  return, D
end
```

A.3 Methods Involving Depths along a Line Of Bearing

A.3.1 The *SetLineOfBearing* Method

```
PRO  BATHYMETRY::SetLineOfBearing,  StartOfLineLatLong, EndOfLineLatLong,  NumberOfDepths
  RB = LatLong_to_RB( EndOfLineLatLong, StartOfLineLatLong )
  DeltaDistances = ( (RB[0] / 1.852 ) / float( NumberOfDepths - 1) )            ;need distance in nm, and at about distance
              increment in file
  Distances = dindgen(1,NumberOfDepths) * DeltaDistances
  Bearing = RB[1]
  ptr_free, self.CurrentLineOfBearingLatLongs
  self.CurrentLineOfBearingLatLongs = ptr_new(  LatLongAlongBearing( StartOfLineLatLong, Distances,  Bearing ) )
  self.CurrentLOBLatLongsValid = !true
end
```

A.3.2 The *SetLineOfBearing1* Method

```
PRO  BATHYMETRY::SetLineOfBearing1,  StartOfLineLatLong,  Bearing,  RangeResolutionInNM,
              NumberOfRangeValues
  EndOfLineLatLong = LatLongAlongBearing( StartOfLineLatLong, RangeResolutionInNM * (NumberOfRangeValues - 1),
              Bearing )
  self->SetLineOfBearing,  StartOfLineLatLong, EndOfLineLatLong,  NumberOfRangeValues
end
```

A.3.3 The *LineOfBearingDepths* Method

```
FUNCTION  BATHYMETRY::LineOfBearingDepths,  StartOfLineLatLong, EndOfLineLatLong,  NumberOfDepths
  if defined(StartOfLineLatLong) then   self->SetLineOfBearing,  StartOfLineLatLong, EndOfLineLatLong,
              NumberOfDepths
  return,  self->Depth( *self.CurrentLineOfBearingLatLongs )
end
```

A.3.4 The *LineOfBearingLatLongs* Method

```
FUNCTION  BATHYMETRY::LineOfBearingLatLongs,  LATITUDES_ONLY=LatitudesOnly,
          LONGITUDES_ONLY=LongitudesOnly
  if keyword_set(LatitudesOnly) then return, (*self.CurrentLineOfBearingLatLongs)[0,*]
  if keyword_set(LongitudesOnly) then return, (*self.CurrentLineOfBearingLatLongs)[1,*]
  return, *self.CurrentLineOfBearingLatLongs
end
```

A.3.5 The *LineOfBearingLatitudes* and *LineOfBearingLongitudes* Methods

```
FUNCTION  BATHYMETRY::LineOfBearingLatitudes
  return, (*self.CurrentLineOfBearingLatLongs)[0,*]
end

FUNCTION  BATHYMETRY::LineOfBearingLongitudes
  return, (*self.CurrentLineOfBearingLatLongs)[1,*]
end
```

A.4 Obtaining Depths inside a Box

A.4.1 The *DefineBox* Method

```
PRO  BATHYMETRY::DefineBox,  LatLongForOneCorner,  LatLongForOppositeCorner
  self.CurrentBox.SouthLat = min( [ LatLongForOneCorner[0], LatLongForOppositeCorner[0] ]) > self.SouthLat
  self.CurrentBox.NorthLat = max( [ LatLongForOneCorner[0], LatLongForOppositeCorner[0] ]) < self.NorthLat
  self.CurrentBox.WestLong = min( [ LatLongForOneCorner[1], LatLongForOppositeCorner[1] ]) > self.WestLong
  self.CurrentBox.EastLong = max( [ LatLongForOneCorner[1], LatLongForOppositeCorner[1] ]) < self.EastLong

  self.CurrentBox.Index.South = self->ReturnLatIndex( self.CurrentBox.SouthLat )
  self.CurrentBox.Index.North = self->ReturnLatIndex( self.CurrentBox.NorthLat )
  self.CurrentBox.Index.West = self->ReturnLongIndex( self.CurrentBox.WestLong )
  self.CurrentBox.Index.East = self->ReturnLongIndex( self.CurrentBox.EastLong )
  self.CurrentBox.Valid = !true
end
```

A.4.2 The *BoxLatitude* and *BoxLongitude* Methods

```
FUNCTION  BATHYMETRY::BoxLatitudes
  if self.CurrentBox.Valid eq !false then begin
    ok = dialog_message('Must specify Box first, using x->DefineBox, C1, C2   method.  Returning to caller.', /ERROR,
          TITLE='BOX NOT DEFINED')
    return, [ 0.0D ]
  endif
  return, (*self.LatGrid)[self.CurrentBox.Index.South:self.CurrentBox.Index.North:self.CurrentBox.DecimationFactor]
end

FUNCTION  BATHYMETRY::BoxLongitudes
  if self.CurrentBox.Valid eq !false then begin
    ok = dialog_message('Must specify Box first, using x->DefineBox, C1, C2   method.  Returning to caller.', /ERROR,
          TITLE='BOX NOT DEFINED')
    return, [ 0.0D ]
  endif
  return, (*self.LongGrid)[self.CurrentBox.Index.West:self.CurrentBox.Index.East:self.CurrentBox.DecimationFactor]
end
```

A.4.3 The *DepthsInBox* Method

```
FUNCTION  BATHYMETRY::DepthsInBox,  LatLongForOneCorner,  LatLongForOppositeCorner,
          DECIMATION_FACTOR=DecimationFactor
```

```
 if defined(DecimationFactor) then self.CurrentBox.DecimationFactor = DecimationFactor
 if defined(LatLongForOneCorner) then self->DefineBox, LatLongForOneCorner,  LatLongForOppositeCorner

 return,  (*self.Bathymetry)[self.CurrentBox.Index.West:self.CurrentBox.Index.East:self.CurrentBox.DecimationFactor, $
                    self.CurrentBox.Index.South:self.CurrentBox.Index.North:self.CurrentBox.DecimationFactor]
end
```

A.5 Displaying the Bathymetry in a Box

A.5.1 The *DisplayBoxDepths* Method

```
PRO  BATHYMETRY::DisplayBoxDepths, LatLongForOneCorner,  LatLongForOppositeCorner,  DecimationFactor, $
                              DEEPLIMIT=DeepLimit, SHALLOWLIMIT=ShallowLimit,
            COLOURTABLEINDEX=ColorTableIndex, NOCOLOURTABLE=NoColourTable,  $
                              NOGRID=NoGrid, EXACTGRIDAXESLAT=ExactGridAxesLat,
            EXACTGRIDAXESLONG=ExactGridAxesLong, DEGMINUTES=DegMinutes, CHARSIZE=CharSize

 if defined(DegMinutes) then self.PlotInDegMin = DegMinutes else self.PlotInDegMin = -1
 tmpbathy = self->DepthsInBox( LatLongForOneCorner,  LatLongForOppositeCorner,
            DECIMATION_FACTOR=DecimationFactor  )

 if undefined(CharSize) then CharSize = 1.6
 map_set,/MERCATOR, LIMIT=[self.CurrentBox.SouthLat, Self.CurrentBox.WestLong, self.CurrentBox.NorthLat,
            self.CurrentBox.EastLong], /ISOTROPIC, XMARGIN=3+Charsize, YMARGIN=3+CharSize

 ptr_free, self.MappedBathyImage
 self.MappedBathyImage = ptr_new( map_image( tmpbathy, StartX, StartY, Xsize, Ysize, /COMPRESS, scale=.04, $     ;
            prepare image for overplotting with the map projection
           LATMIN=self.CurrentBox.SouthLat, LATMAX=self.CurrentBox.NorthLat,  $                                  ; note that
            min/max boundaries start/end at the edge of a bathymetric tile
           LONMIN=self.CurrentBox.WestLong, LONMAX=self.CurrentBox.EastLong ) )

 if undefined(DeepLimit)    then self.DeepLimit    = min(tmpbathy) > (-5000.0) else self.DeepLimit = DeepLimit
 if undefined(ShallowLimit) then self.ShallowLimit = max(tmpbathy) < 0.0       else self.ShallowLimit = ShallowLimit
 if undefined(ColorTableIndex)  then ColorTableIndex  =1
 if NOT keyword_set(NoColourTable) then  self->LoadColourTable, WATERINDEX=ColorTableIndex

 tmpImage1 = bytscl( *self.MappedBathyImage, MIN=self.DeepLimit, MAX=self.ShallowLimit, TOP=self.WaterTable.Top )
 tmpImage2 = bytscl( *self.MappedBathyImage, MIN=0.0, TOP=self.LandTable.Number)
 indices = where( *self.MappedBathyImage gt 0.0, Count)
 if Count gt 0 then  tmpImage1[indices] = tmpImage2[indices] + self.LandTable.Bottom
 ptr_free, self.IndexedBathyImage
 self.IndexedBathyImage = ptr_new(tmpImage1)
 tv, *self.IndexedBathyImage, StartX, StartY, XSIZE=Xsize, YSIZE=Ysize       ;Xsize/Ysize only used for printing [variable
            pixel size]
 self.PlotWinNumber = !d.window

 if keyword_set(NoGrid) eq !true then return     ;no grid lines

 if undefined(ExactGridAxesLat) then ExactGridAxesLat1 = 0 else ExactGridAxesLat1 = ExactGridAxesLat
 if undefined(ExactGridAxesLong) then ExactGridAxesLong1 = 0 else ExactGridAxesLong1 = ExactGridAxesLong
 if ExactGridAxesLat1 eq 1 then ExactGridAxesLat1 = 7
 if ExactGridAxesLong1 eq 1 then ExactGridAxesLong1 = 7
 if ExactGridAxesLat1 gt 0 then begin
   Lats = frange(ExactGridAxesLat1, self.CurrentBox.SouthLat, self.CurrentBox.NorthLat)
   if defined(DegMinutes) then  LatDegMin =  DEG_DEGMIN( Lats, /LATITUDE, NUMDECIMALS=DegMinutes )
 endif
 if ExactGridAxesLong1 gt 0 then begin
   Lons = frange(ExactGridAxesLong1, self.CurrentBox.WestLong, self.CurrentBox.EastLong)
   if defined(DegMinutes) then  LongDegMin =  DEG_DEGMIN( Lons, /LONGITUDE, NUMDECIMALS=DegMinutes )
 endif
```

```
 if ExactGridAxesLat1 eq 0 and ExactGridAxesLong1 eq 0  then begin
  map_grid, /BOX_AXES, CHARSIZE=CharSize
  return
 endif

 if ExactGridAxesLat1 eq 0 and ExactGridAxesLong1 gt 0 and defined(DegMinutes) eq !false  then begin
  map_grid, /BOX_AXES, CHARSIZE=CharSize, LONS=Lons
  return
 endif

 if ExactGridAxesLat1 eq 0 and ExactGridAxesLong1 gt 0 and defined(DegMinutes) eq !true  then begin
  map_grid, /BOX_AXES, CHARSIZE=CharSize, LONS=Lons, LONNAMES=LongDegMin
  return
 endif


 if ExactGridAxesLat1 gt 0 and ExactGridAxesLong1 eq 0 and defined(DegMinutes) eq !false  then begin
  map_grid, /BOX_AXES, CHARSIZE=CharSize, LATS=Lats
  return
 endif

 if ExactGridAxesLat1 gt 0 and ExactGridAxesLong1 eq 0 and defined(DegMinutes) eq !true  then begin
  map_grid, /BOX_AXES, CHARSIZE=CharSize, LATS=Lats,  LATNAMES=LatDegMin
  return
 endif


 if ExactGridAxesLat1 gt 0 and ExactGridAxesLong1 gt 0 and defined(DegMinutes) eq !false  then begin
  map_grid, /BOX_AXES, CHARSIZE=CharSize, LATS=Lats, LONS=Lons
  return
 endif

 if ExactGridAxesLat1 gt 0 and ExactGridAxesLong1 gt 0 and defined(DegMinutes) eq !true  then begin
  map_grid, /BOX_AXES, CHARSIZE=CharSize, LATS=Lats, LONS=Lons, LATNAMES=LatDegMin,
          LONNAMES=LongDegMin
  return
 endif
end
```

A.5.2 The *DisplayBoxContours* Method

```
PRO  BATHYMETRY::DisplayBoxContours,  ContourDepths,    C_CHARSIZE=CharSize,
          C_COLORS=ContourColorIndices,  C_LABELS=LabelContourLevels

 device, WINDOW_STATE = State
 if self.PlotWinNumber lt 0   ||  State[self.PLotWinNumber] eq 0 then begin       ;short-curcuit OR
  self.PlotWinNumber = -1
  ok = dialog_message('Must call method PlotBoxDepths first. Disregarding.', TITLE='BATHYMETRY-
          >DisplayBoxContours: No Plot')
  return
 endif
 if undefined(ContourDepths) then begin
  del = abs( ( max(self->DepthsInBox()) - min(self->DepthsInBox()) ) / 5.0 )
  del = 10.0 * fix(del / 10.0)
  StartDepth = 0.0
  while StartDepth ge max(self->DepthsInBox()) do begin
   StartDepth = StartDepth - del
  endwhile
  ContourDepths1 = StartDepth
  NextDepth = StartDepth
  while min(ContourDepths1) - del ge min(self->DepthsInBox()) do begin
   NextDepth = NextDepth - del
   ContourDepths1 = [ContourDepths1, NextDepth]
  endwhile
 endif else ContourDepths1 = ContourDepths
```

```
 if min(ContourDepths1) gt 0.0 then ContourDepths1 = -ContourDepths1     ;ensure depths are negative
 ContourDepths1 = ContourDepths1[ sort(ContourDepths1) ]                 ;needs to be in ascending order

 if undefined(CharSize) then CharSize1 = 1.3 else CharSize1 = CharSize
 if undefined(ContourColorIndices) then ContourColorIndices1 = indgen(self.ContourTable.Number) +
          self.ContourTable.Bottom else ContourColorIndices1 = ContourColorIndices   ;default to upper 10 colors in
          color table
 if undefined(LabelContourLevels) then LabelContourLevels1 = Replicate(1,n_elements(ContourDepths1))  else
          LabelContourLevels1 = LabelContourLevels   ;default to upper 10 colors in color table

 Contour, self->DepthsInBox(),  self->BoxLongitudes(), self->BoxLatitudes(),  LEVELS=ContourDepths1, /FOLLOW,
          /OVERPLOT,  $
               C_CHARSIZE=CharSize1,  C_COLORS=ContourColorIndices1,  C_LABELS=LabelContourLevels1
end
```

A.6 Managing the Colour Table

A.6.1 The *LoadColourTable* Method

```
PRO BATHYMETRY::LoadColourTable,  WATERINDEX=WaterIndex, CONTOURINDEX=ContourIndex,
          LABELINDEX=LabelIndex
 LastWindow = !d.window

 device, WINDOW_STATE = State
 if self.PlotWinNumber ge 0   &&  State[self.PLotWinNumber] eq 1  then  wset, self.PlotWinNumber

 if undefined(WaterIndex) then WaterIndex1 = 1 else WaterIndex1 = ((WaterIndex > 0) < 40)

 loadct, 8, NCOLORS=self.LandTable.Number + 25        ;Set green for land
 tvlct, LandR, LandG, LandB, /GET
 LandR = LandR[15:self.LandTable.Number+15-1]
 LandG = LandG[15:self.LandTable.Number+15-1]
 LandB = LandB[15:self.LandTable.Number+15-1]

 loadct, WaterIndex1, BOTTOM=self.WaterTable.Bottom,  NCOLORS=self.WaterTable.Number   ;Water
 tvlct, LandR, LandG, LandB, self.LandTable.Bottom                    ;Load Land green colours

 if defined(ContourIndex) then begin
  loadct, ((CountourIndex > 0) < 40), BOTTOM=self.ContourTable.Bottom, NCOLORS=self.ContourTable.Number
  endif else begin
  Red   = [ 255, 255, 255, 253, 250,   248, 245, 240, 236, 230 ]          ;Contour colours
  Green = [ 255, 255, 250, 240, 230,   220, 210, 190, 166, 150 ]
  Blue  = [ 200, 150,  40,   0,   0,     0,   0,   0,   0,   0 ]
  tvlct, Red, Green, Blue, self.ContourTable.Bottom
 endelse

 if defined(LabelIndex) then begin
  loadct, ((LabelIndex > 0) < 40), BOTTOM=self.LineTable.Bottom, NCOLORS=self.LineTable.Number
  endif else begin
  Red   = [ 255,   0,   0, 255, 255,   255, 165, 165, 255, 255 ]          ;Line colours
  Green = [   0, 255,   0, 150,   0,   165, 255, 165, 255, 255 ]
  Blue  = [   0,   0, 255,   0, 255,   165, 165, 255, 100, 255 ]
 ;          Red  Gn   Bl   Or  Pur   LRd  LtG  LtB  Yel   WT
  tvlct, Red, Green, Blue, self.LineTable.Bottom
 endelse
 wset, LastWindow
end
```

A.6.2 The *ShowColourTable* Method

```
PRO BATHYMETRY::ShowColourTable, WinWidth, WinHeight, DepthIncrementInM, CHARSIZE=CharSize
```

```
 device, WINDOW_STATE = State
 if self.PlotWinNumber lt 0   ||  State[self.PlotWinNumber] eq 0  then begin
   self.PlotWinNumber = -1
   ok = dialog_message('Must call method PlotBoxDepths first. Disregarding.',  TITLE='BATHYMETRY-
            >ShowColourTable: No Plot Window')
   return
 endif
 if undefined(WinWidth) then WinWidth = 100
 if undefined(WinHeight) then WinHeight = 500
 if undefined(CharSize) then CharSize1 = 1.6 else CharSize1 = CharSize
 if undefined(DepthIncrementInM) then DepthIncrementinM = 50.
 WinWidth = WinWidth > 80.
 WinHeight = WinHeight > 100.
 OldWindow = !d.window
 win, WinWidth, WinHeight
 xyouts, 0, 5, strtrim(string(fix(abs(self.DeepLimit))),2), CHARSIZE=CharSize1, WIDTH=CharWidthNormalized
 CharWidth = CharWidthNormalized * WinWidth + 10
 Width = (WinWidth - CharWidth - 10) > 10
 Height = WinHeight - 30.
 Left = CharWidth
 Bottom = 15.
 Yzero = self.WaterTable.Top
 image = bindgen(1, Yzero)
 tv, congrid( image, Width, Height ), Left, Bottom
 plots, [Left, Left, Left+Width, Left+Width, Left], [Bottom, Bottom+Height, Bottom+Height, Bottom, Bottom], /DEVICE
 Xpos = Left - 5
 Increment = 0.0
 repeat begin
   Ypos = ( ( ( (Yzero - 0.0) / ( 0.0 - self.DeepLimit) ) * ( Increment -  self.DeepLimit ) + 0.0 ) / Yzero) * Height + Bottom -
            10    ;-10 to move character down slightly
   xyouts, Xpos, Ypos, strtrim(string(fix(abs(Increment))),2),   ALIGNMENT=1.0, /DEVICE, CHARSIZE=CharSize1
   Increment = Increment - abs(DepthIncrementinM)
 endrep until  Increment lt  self.DeepLimit
 wset, OldWindow
end
```

A.7 Displaying Elements over the Bathymetry Plot

A.7.1 The *OPlotBox* Method

```
PRO  BATHYMETRY::OPlotBox,   LatLongForOneCorner,  LatLongForOppositeCorner, COLOUR=PlotColour,
            LINETHICKNESS=Thickness

 device, WINDOW_STATE = State
 if self.PlotWinNumber lt 0   ||  State[self.PLotWinNumber] eq 0  then begin
   self.PlotWinNumber = -1
   ok = dialog_message('Must call method PlotBoxDepths first. Disregarding.', TITLE='BATHYMETRY->OPlotBox: No
            Plot')
   return
 endif
 LastWindow = !d.window
 wset, self.PlotWinNumber

 if undefined(PlotColour) then PlotColour = 255
 if undefined(Thickness) then Thickness = 1

 SouthLat = min( [ LatLongForOneCorner[0], LatLongForOppositeCorner[0] ]) > self.SouthLat
 NorthLat = max( [ LatLongForOneCorner[0], LatLongForOppositeCorner[0] ]) < self.NorthLat
 WestLong = min( [ LatLongForOneCorner[1], LatLongForOppositeCorner[1] ]) > self.WestLong
 EastLong = max( [ LatLongForOneCorner[1], LatLongForOppositeCorner[1] ]) < self.EastLong

 self->SetLineOfBearing, [NorthLat, WestLong], [NorthLat, EastLong], 100
 LatLongs = self->LineOfBearingLatLongs()
```

```
  oplot, LatLongs[1,*], LatLongs[0,*], color=PlotColour, THICK=Thickness

  self->SetLineOfBearing, [NorthLat, EastLong], [SouthLat, EastLong], 100
  LatLongs = self->LineOfBearingLatLongs()
  oplot, LatLongs[1,*], LatLongs[0,*], color=PlotColour, THICK=Thickness

  self->SetLineOfBearing, [SouthLat, EastLong], [SouthLat, WestLong], 100
  LatLongs = self->LineOfBearingLatLongs()
  oplot, LatLongs[1,*], LatLongs[0,*], color=PlotColour, THICK=Thickness

  self->SetLineOfBearing, [SouthLat, WestLong], [NorthLat, WestLong], 100
  LatLongs = self->LineOfBearingLatLongs()
  oplot, LatLongs[1,*], LatLongs[0,*], color=PlotColour, THICK=Thickness
  wset, LastWindow
end
```

A.7.2 The *OPlotPolygon* Method

```
PRO  BATHYMETRY::OPlotPolygon,  LatLongVector, COLOUR=PlotColour, LINETHICKNESS=Thickness,
            CLOSEPOLYGON=ClosePolygon

  device, WINDOW_STATE = State
  if self.PlotWinNumber lt 0   ||  State[self.PLotWinNumber] eq 0  then begin
    self.PlotWinNumber = -1
    ok = dialog_message('Must call method PlotBoxDepths first. Disregarding.',  TITLE='BATHYMETRY->OPlotPolygon:
            No Plot')
    return
  endif
  LastWindow = !d.window
  wset, self.PlotWinNumber
  if undefined(PlotColour) then PlotColour = 255
  if undefined(Thickness) then Thickness = 1
  if n_elements(LatLongVector[0,*]) lt 3 then begin
    ok = dialog_message('Need to define a polygon of at least 3 vertices. Disregarding.',  TITLE='BathyObj-
            >OPlotPolygon> ERROR: Too few points')
    return
  endif
  LatLongVector1 = LatLongVector
  if keyword_set(ClosePolygon) then LatLongVector1 = [ [LatLongVector1], [LatLongVector1[*,0]] ]   ;close the poly with
            initial value at end
  for i = 0, n_elements(LatLongVector1[0,*]) - 2 do begin
    self->SetLineOfBearing, LatLongVector1[*,i], LatLongVector1[*,i+1],  100
    LatLongs = self->LineOfBearingLatLongs()
    oplot, LatLongs[1,*], LatLongs[0,*], color=PlotColour, THICK=Thickness
  endfor
  wset, LastWindow
end
```

A.7.3 The *GetCursorLatLong* Method

```
FUNCTION BATHYMETRY::GetCursorLatLong,   SINGLE=SinglePoint, PAINTSYMBOL=PaintSymbol,
            SYMSIZE=SymSize, COLOUR=Colour, SHOWLOCATION=ShowLocation, LINETHICKNESS=LineThickness

  device, WINDOW_STATE = State
  if self.PlotWinNumber lt 0   ||  State[self.PLotWinNumber] eq 0  then begin
    self.PlotWinNumber = -1
    ok = dialog_message('Must call method PlotBoxDepths first. Disregarding.',  TITLE='BATHYMETRY-
            >GetCursorLatLong: No Plot')
    return, [-9999., -9999.]
  endif
  LastWindow = !d.window
  wset, self.PlotWinNumber
  if keyword_set(SinglePoint) then begin
    cursor, X, Y, /WAIT
```

```
  wset, LastWindow
  return, [X, Y]
 endif
 if undefined(SymSize) then SymSize1 = 1 else SymSize1 = SymSize
 if undefined(Colour) then Colour1 = 255 else Colour1 = Colour
 !mouse.button = 0
 LastButton = 0
 TextWidth = 4
 FirstClick = 1
 while !mouse.button ne 4  do begin
  cursor, X, Y, /NOWAIT
  Valid = X gt self.CurrentBox.WestLong and X lt self.CurrentBox.EastLong and Y gt self.CurrentBox.SouthLat and Y lt
          self.CurrentBox.NorthLat
  if Valid gt 0 then begin
   if keyword_set(ShowLocation) then begin
    XY = strtrim(string(Y, FORMAT='(F15.5)'),2) + '  ' + strtrim(string(X, FORMAT='(F15.5)'),2)
    if self.PlotInDegMin gt -1 then begin
     XY = strtrim(deg_degmin(Y, /LATITUDE, NUMDECIMALS=self.PlotInDegMin),2) + '  ' +  strtrim(deg_degmin(X,
          /LONGITUDE, NUMDECIMALS=self.PlotInDegMin),2)
    endif
    plots, [3, TextWidth], [4+!D.Y_CH_SIZE/2, 4+!D.Y_CH_SIZE/2], /DEVICE,  THICK=!D.Y_CH_SIZE+4,
          COLOR=!p.background
    xyouts, 5, 5, XY,  /DEVICE
    TextWidth = (strlen(XY)*!D.X_CH_SIZE+2)
   endif
  endif
  if !mouse.button eq 0 and LastButton eq 1 and Valid then begin
   if undefined(LatLongVector) then begin
    LatLongVector = [X, Y]
    endif else begin
    LatLongVector = [ [LatLongVector], [X, Y] ]
   endelse
   if defined(PaintSymbol) then begin
    if FirstClick eq 1 then begin
     plots, X, Y, PSYM=((PaintSymbol>1) < 8), SYMSIZE=SymSize1, COLOR=Colour1
     FirstClick = 0
     endif else begin
     plots, [Xlast, X], [YLast, Y],  PSYM=-((PaintSymbol>1) < 8), SYMSIZE=SymSize1, COLOR=Colour1,
          THICK=LineThickness
    endelse
   endif
   XLast = X
   YLast = Y
  endif
  LastButton = !mouse.button
  wait, 0.02
 endwhile
 if keyword_set(ShowLocation) then plots, [3, TextWidth], [4+!D.Y_CH_SIZE/2, 4+!D.Y_CH_SIZE/2], /DEVICE,
          THICK=!D.Y_CH_SIZE+4, COLOR=!p.background

 wset, LastWindow
 if defined(LatLongVector) then return, LatLongVector else return, [-1D, -1D]
end
```

A.8 Determining the File Boundaries

A.8.1 The *LatitudeBounds* and *LongitudeBounds* Methods

```
FUNCTION  BATHYMETRY::LatitudeBounds
 return, [ self.SouthLat, self.NorthLat ]
end

FUNCTION  BATHYMETRY::LongitudeBounds
```

```
  return, [ self.WestLong, self.EastLong ]
end
```

A.8.2 The *LatitudeResolution* and *LongitudeResolution* Methods

```
FUNCTION  BATHYMETRY::LatitudeResolution, MINUTES=Minutes
  if keyword_set(Minutes) then return, 60.0D * self.LatDelta
  return, self.LatDelta
end

FUNCTION  BATHYMETRY::LongitudeResolution, MINUTES=Minutes
  if keyword_set(Minutes) then return, 60.0D * self.LongDelta
  return, self.LongDelta
end
```

A.9 Utility Methods

A.9.1 The *MappedBathyImage* Method

```
FUNCTION  BATHYMETRY::MappedBathyImage, VALID=Valid
  Valid = ptr_valid(self.MappedBathyImage)
  if Valid eq !true then  return, *self.MappedBathyImage  else  return, 0
end
```

A.9.2 The *IndexedBathyImage* Method

```
FUNCTION  BATHYMETRY::IndexedBathyImage, VALID=Valid
  Valid = ptr_valid(self.IndexedBathyImage)
  if Valid eq !true then  return, *self.IndexedBathyImage  else  return, 0
end
```

A.9.3 The *ReturnLatIndex* and *ReturnLongIndex* Methods

```
FUNCTION  BATHYMETRY::ReturnLatIndex,  Latitude, VALID=Valid
  Valid = ( Latitude ge self.SouthLat )   and   ( Latitude le self.NorthLat )
  return, ( ( fix(self.LatIndexScale * (Latitude - self.SouthLat) + 0.5) ) > 0 ) < (self.NumLats - 1)
end

FUNCTION  BATHYMETRY::ReturnLongIndex,  Longitude,  VALID=Valid
  Valid = ( Longitude ge self.WestLong )   and   ( Longitude le self.EastLong )
  return, ( ( fix(self.LongIndexScale * (Longitude - self.WestLong) + 0.5) ) > 0 ) < (self.NumLongs - 1)
end
```

A.9.4 The *ReturnFullBathymetry* Method

```
FUNCTION  BATHYMETRY::ReturnFullBathymetry
  return, *self.Bathymetry
end
```

Bibliography

IDL Online Help, supplied with the Interactive Data Language software package.

List of symbols/abbreviations/acronyms/initialisms

DND	Department of National Defence
EMM	Environment Modeling Manager
IDL	Interactive Data Language
RSI	Research Systems Inc.
DE	Development Environment

This page intentionally left blank.

Distribution list

Document No.: DRDC Atlantic TM 2005-265

LIST PART 1: Internal Distribution by Centre:

2 DRDC Atlantic LIBRARY FILE COPIES
3 DRDC Atlantic LIBRARY (SPARES)
1 AUTHOR
1 G.R. MELLEMA
1 M. LEFRANCOIS
1 B. TOPP
1 J. OSLER
1 P. HINES
1 J. THERIAULT
1 D. ELLIS

13 TOTAL LIST PART 1

LIST PART 2: External Distribution by DRDKIM

1 NDHQ/ ADM S&T/ DRDKIM 3
1 CONTRACTOR - F. CAMPAIGNE, 42 Autumn Drive, Eastern Passage, NS B3G 1L5
1 CONTRACTOR - D. MCCAMMON, 475 Baseline Road, Waterville, NS B0P 1V0
1 CONTRACTOR - G. INGLIS, General Dynamics Canada Ltd., 11 Thornhill Drive, Suite 102, Dartmouth, NS B3B 1R9
1 CONTRACTOR - C. BROBECK, General Dynamics Canada Ltd., 11 Thornhill Drive, Suite 102, Dartmouth, NS B3B 1R9

5 TOTAL LIST PART 2

18 TOTAL COPIES REQUIRED

This page intentionally left blank.

DOCUMENT CONTROL DATA

(Security classification of title, body of abstract and indexing annotation must be entered when the overall document is classified)

1. ORIGINATOR (The name and address of the organization preparing the document. Organizations for whom the document was prepared, e.g. Centre sponsoring a contractor's report, or tasking agency, are entered in section 8.)

DRDC Atlantic

2. SECURITY CLASSIFICATION (Overall security classification of the document including special warning terms if applicable.)

Unclassified

3. TITLE (The complete document title as indicated on the title page. Its classification should be indicated by the appropriate abbreviation (S, C, R or U) in parentheses after the title.)

An IDL Object for Managing Gridded Bathymetry Files

4. AUTHORS (First name, middle initial and last name. If military, show rank, e.g. Maj. John E. Doe.)

W.A. Roger

5. DATE OF PUBLICATION (Month and year of publication of document.)

March 2006

6a. NO. OF PAGES (Total containing information, including Annexes, Appendices, etc.)

34

6b. NO. OF REFS (Total cited in document.)

6

7. DESCRIPTIVE NOTES (The category of the document, e.g. technical report, technical note or memorandum. If appropriate, enter the type of report, e.g. interim, progress, summary, annual or final. Give the inclusive dates when a specific reporting period is covered.)

Technical Memorandum

8. SPONSORING ACTIVITY (The name of the department project office or laboratory sponsoring the research and development – include address.)

DRDC Atlantic
9 Grove St
Dartmouth, NS B3A 3C5

9a. PROJECT OR GRANT NO. (If appropriate, the applicable research and development project or grant number under which the document was written. Please specify whether project or grant.)

11cg

9b. CONTRACT NO. (If appropriate, the applicable number under which the document was written.)

10a. ORIGINATOR'S DOCUMENT NUMBER (The official document number by which the document is identified by the originating activity. This number must be unique to this document.)

DRDC Atlantic TM 2005-265

10b. OTHER DOCUMENT NO(s). (Any other numbers which may be assigned this document either by the originator or by the sponsor.)

11. DOCUMENT AVAILABILITY (Any limitations on further dissemination of the document, other than those imposed by security classification.)

(X) Unlimited distribution
( ) Defence departments and defence contractors; further distribution only as approved
( ) Defence departments and Canadian defence contractors; further distribution only as approved
( ) Government departments and agencies; further distribution only as approved
( ) Defence departments; further distribution only as approved
( ) Other (please specify):

12. DOCUMENT ANNOUNCEMENT (Any limitation to the bibliographic announcement of this document. This will normally correspond to the Document Availability (11). However, where further distribution (beyond the audience specified in (11) is possible, a wider announcement audience may be selected.))

13. ABSTRACT (A brief and factual summary of the document. It may also appear elsewhere in the body of the document itself. It is highly desirable that the abstract of classified documents be unclassified. Each paragraph of the abstract shall begin with an indication of the security classification of the information in the paragraph (unless the document itself is unclassified) represented as (S), (C), (R), or (U). It is not necessary to include here abstracts in both official languages unless the text is bilingual.)

The Networked Underwater Warfare (NUW) Technology Demonstration Project (TDP) has a need for precise ocean bottom topography of the Scotian Shelf area. The depth of the ocean at particular locations is utilized by a number of technologies within the project to either predict future ocean conditions, or as input to tactical algorithms. The ocean bathymetry over some specified area is also needed as one layer in the display of targets. This document describes a software object to extract and display underwater depths from a gridded bathymetry file. It is written in the IDL programming language to facilitate ease of use and rapid development of the controlling software. The object first reads the file into memory for quick response, then provides a number of software methods to obtain selected areas of depth, display all or portions of the bathymetry, and allow additions to the display such as contour lines and lines of bearing. The object can handle a range of bathymetry files that store depths at locations that are separated by a constant angular difference in both latitude and longitude.

Résumé

Dans le cadre du projet de démonstration de technologies (PDT) de guerre sous-marine en réseau (GSMR), on a besoin de données topographiques précises relatives au fond océanique de la région de la plate-forme Néo-Écossaise. La profondeur de l'océan en des points particuliers est utilisée par un certain nombre de technologies dans le cadre du projet soit pour la prévision des conditions futures de l'océan, soit comme paramètre d'entrée d'algorithmes tactiques. Les données bathymétriques relatives à une zone spécifiée sont aussi requises comme couche dans l'affichage des cibles. Le présent document décrit un objet logiciel permettant d'extraire et d'afficher les profondeurs sous-marines à partir d'un fichier de données bathymétriques maillées. Cet objet est écrit en langage de programmation IDL de manière à faciliter l'utilisation et à permettre l'élaboration rapide du logiciel de contrôle. L'objet charge d'abord le fichier en mémoire pour permettre une réponse rapide, puis il offre un certain nombre de méthodes logicielles permettant d'obtenir des zones de profondeur choisies, d'afficher la totalité ou une partie des données bathymétriques et d'ajouter des éléments à l'affichage, par exemple des courbes de niveau et des lignes de relèvement. L'objet peut traiter une gamme de fichiers bathymétriques dans lesquels sont stockées les profondeurs à des emplacements séparés par une différence angulaire constante tant en latitude qu'en longitude.

14. KEYWORDS, DESCRIPTORS or IDENTIFIERS (Technically meaningful terms or short phrases that characterize a document and could be helpful in cataloguing the document. They should be selected so that no security classification is required. Identifiers, such as equipment model designation, trade name, military project code name, geographic location may also be included. If possible keywords should be selected from a published thesaurus, e.g. Thesaurus of Engineering and Scientific Terms (TEST) and that thesaurus identified. If it is not possible to select indexing terms which are Unclassified, the classification of each should be indicated as with the title.)

Bathymetry, GIS, Interactive Data Language

This page intentionally left blank.

Defence R&D Canada

Canada's leader in defence
and National Security
Science and Technology

R & D pour la défense Canada

Chef de file au Canada en matière
de science et de technologie pour
la défense et la sécurité nationale

www.drdc-rddc.gc.ca