Subject: I have the craziest idea...
Posted by T Bowers on Wed, 14 Feb 2001 19:32:07 GMT
View Forum Message <> Reply to Message

I've written a widget program that I've tried to design as a general data import interface. What I mean by general is that it's a wrapper widget to import various types of data and present it back to the user in some kind of standardized format for easy use. E.g., I have some 3D gridded data in ASCII text files of one format, 2D scattered data in ASCII text files of another format, etc. etc. What I did was write the widget as a function call so all I have to do is call it and I always have access to ALL data files that I know how to read. And, more importantly, the data is passed back to me in a format that's consistent and easy to process and visualize in IDL.

So, what do all of you think about the crazy idea of me putting this up on a server and all of us passing this around, adding code where we can so that we all have a centralized function that'll read many data formats? Seems to me that we all have our specialty, and when one of us wants to read in a new type we just come to this newsgroup for the answer anyway. I call it the ngDataImporter. I usually put my initials in front of the name of each function so I don't ever step on other functions. Since this would be a group effort by us all, I just fronted it with ng for newsgroup, the IDL newsgroup users.

From the intro comments:

```
;////////////////////////////////
;+
; NAME:
;  ngDataImporter
;
; PURPOSE:
;  This is a general purpose data import widget. It is meant to be
distributed
;  between users and appropriate data reading code inserted. The idea is to
present
;  to the calling routine a PREDICTABLE and INTUITIVE representation of the
data
;  for easy processing AND visualization. For example, 3 dimensional
scattered data
;  points should ALWAYS be returned as an array [4,n] of x,y,z,f(x,y,z)
vectors because
;  1) For processing, it's easy to interpolate to a 3D grid since most
interpolation
;  routines (including IDL's grid3()) expect 3D scattered data in [4,n]
format,
;  and 2) for visualization, you could vis the scatter points as an
IDLgrPolyLine
```

```
;  object. IDLgrPolyLine expects 3D data in [3,n] for x,y,z coords, and you
could
;  pass the data colored by value via the VERT_COLORS keyword. This can be
useful
;  especially if you returned the poly line descriptor in the POLYLINES
keyword (see
;  POLYLINES keyword for IDLgrPolyLine object and the d_tankleak IDL demo).
The idea
;  here is to keep in mind the routines for both processing and
visualization that
;  can work on the return data.
;  To this end, we define certain return formats for each type of data as
follows:
;    2D Scalar Scatter - [3,n] column vectors of x,y,f(x,y)
;    3D Scalar Scatter - [4,n] column vectors of x,y,z,f(x,y,z)
;    2D Scalar Gridded - [m,n] of f(x,y) scalar data values
;    3D Scalar Gridded - [m,n,o] of f(x,y,z) scalar data values
;    2D Vector Gridded - [2,u,v] where u,v are 2D arrays of u and v
magnitudes
;    3D Vector Gridded - [3,u,v,w] where u,v,w are 3D arrays of u, v and w
magnitudes
;  I haven't worked with vector data, suggested changes are welcome.
;
;  To add a new data type for import:
;  1 - Add your function/procedure call in the case statement of the
appropriate import
;    data function (e.g. in importGriddedData() function).
;  2 - Be sure to at least return the expected data, the dataClass, and
fileName.
;  3 - Place your import code in the 'support' subdirectory.
;  4 - Please try to include a small test/example data file in the 'eg_data'
subdirectory.
;  5 - Document your update in the MODIFICATION HISTORY section below. This
is VERY
;    IMPORTANT! Be sure to include the date for synchronization!!
;  6 - If you added what you believe to be necessary data return items, you
need to
;    update the code for the pointer(ptr) and return(sReturnData) structures
in this
;    module. See code below.
;
;  All feedback and additions are welcome. If you feel that some wholesale
change
;  is in order to make this a better routine, please let me know.
;
; CATEGORY:
;  Utility, Data Processing.
;
```

; CALLING SEQUENCE:
; sImported = ngDataImporter()
;
; INPUTS:
; None.
;
; KEYWORDS:
; VARDATA: Dependent data, e.g. temperature. - OUTPUT
; XDATA: Independent data in X, e.g. longitude. - OUTPUT
; YDATA: Independent data in Y, e.g. latitude. - OUTPUT
; ZDATA: Independent data in Z, e.g. depth. - OUTPUT
; TDATA: Independent data in T (time), e.g. hour. - OUTPUT
; ANCILLARYDATA: Ancillary data, anything else you want to pass back. -
OUTPUT
;   E.g. I have some oceanographic data of 'profiles' (vs. depth) of
salinity.
;   In the data files, there's a header associated with each profile that
also
;   lists that location's bathymetry. I can return this as ancillary data.
; DATANAME: Set this to the data set's name. - OUTPUT
;   Many files contain the data set's name within the header.
; FILENAME: Set this to the name(s) of the data file(s) read. - OUTPUT
;   The calling routine then may use this to determine exactly what was
read.
; DATACLASS: A data classification specifier so we can distinguish general
data types. - OUTPUT
;   Use this in conjunction w/ the DATATYPE kewyord and/or file extensions
;   (e.g. 'jpg') which can be extracted from FILENAME keyword to help you
decide
;   how to handle the returned data. This classification is to let the user
know
;   what form the data is in regardless of source.
;   Supported data classes are:
;   SCAL_GRID_NSMT : SCALar_GRIDded_NSpatialMTemporal dimensions
;   (N=1-n spatial dims, M=0-1 temporal dims)
;   SCAL_SCAT_NSMT : SCALar_SCATtered_NSpatialMTemporal dimensions
;   (N=1-n spatial dims, M=0-1 temporal dims)
;   VECT_GRID_NSMT : VECTor_GRIDded_NSpatialMTemporal dimensions
;   (N=1-n spatial dims, M=0-1 temporal dims)
;   VEVT_SCAT_NSMT : VECTor_SCATtered_NSpatialMTemporal dimensions
;   (N=1-n spatial dims, M=0-1 temporal dims)
;   SCAL_IMAG_NSMT : SCALar_Image_NSpatialMTemporal dimensions
;   (N=1-n spatial dims, M=0-1 temporal dims)
;   ANEW_TYPE_NSMT : If you add a new type, document it here
; DATATYPE: You can use this to be more specific about the data type that
was read in. - OUTPUT
;   There are no restrictions on it's usage. E.g., if you contribute a
routine

```
;   that reads in Level-3 processed SeaWiFs satellite imagery which comes in
an
;   HDF file, you could set this as HDF_SEAWIFS_LEVEL3, or something
similar.
;  POLYLINES:  Poly line descriptor for IDLgrPolyLine object. - OUTPUT
;   May need this to visualize imported scatter point data as individual
lines
;   in an IDLgrPolyLine object. See POLYLINES keyword for IDLgrPolyLine
object
;   and run the d_tankleak.pro demo (in IDL's demo subdirectory) to see the
usefulness
;   of individual polylines. It's necessary to include on import because it
may be
;   difficult for the user to distinguish individual data 'sets' when
scattered data
;   returned as [3,n] (for 2D) or [4,n] (for 3D) v=column vectors.
;  CANCEL: This will be 1 if the user canceled this widget without selecting
data to import. - OUTPUT
;  ERROR: This will be 1 if an error occurred that caused IDL to call the
catch block. - OUTPUT
;  GROUPLEADER: Specifies this widget's group leader. - INPUT
;
; OUTPUTS:
;  Returns an anonymous structure whose fields contain all relavent data
;  and information. This information is also returned via optional keywords.
;
; SIDE EFFECTS:
;  Displays an error dialog in catch block.
;
; RESTRICTIONS:
;  Coded in IDL v5.31.
;
; EXTERNAL MODULES:
;  If you're import code needs helper routines, put those in the 'support'
directory
;  and include the name of the routine(s) here. In general, you shouldn't
have to
;  use external (non-IDL) function/procedure calls in this routine unless
you wish
;  to change the look and feel of the interface. If you add code to one of
the above
;  import routines, add the documentation for the external module call
there.
;
;
; EXAMPLE:
;  oGraphicsModel = obj_new('IDLgrModel')
;  sImported = ngDataImporter()
```

```
;  if ((sImported.cancel) or (sImported.error)) then begin
;   r = dialog_message("No data imported, returning NaN", /information)
;   return, !values.f_NaN
;  endif
;  ;//Add new graphic object based on what type of data was imported
;  case (sImported.dataClass) of
;   "SCAL_GRID_3S0T": begin
;    ;//Volume
;    oGraphic = obj_new('IDLgrVolume', DATA0=bytscl(sImported.data))
;   end
;   "SCAL_GRID_2S0T": begin
;    ;//Surface
;    oGraphic = obj_new('IDLgrSurface', $
;     DATAX=sImported.xData, DATAY=sImported.yData, DATAZ=sImported.data)
;   end
;   "SCAL_SCAT_3S0T": begin
;    ;//Don't interpolate, view as a 3Dpolyline object (or objects if
polyline
;    ;   descriptor present) colored by data value
;    if(sImported.polylines ne 0b) then $
;     oGraphic = obj_new('IDLgrPolyline', $
;      X_COORDS=sImported.xData,  Y_COORDS=sImported.yData,
Z_COORDS=sImported.zData, $
;      VERT_COLORS=(bytscl(sImported.data, /NaN)),
POLYLINES=sImported.polylines)
;    else $
;     oGraphic = obj_new('IDLgrPolyline', $
;      X_COORDS=sImported.xData,  Y_COORDS=sImported.yData,
Z_COORDS=sImported.zData, $
;      VERT_COLORS=(bytscl(sImported.data, /NaN)))
;   end
;   "SCAL_SCAT_2S0T":  begin
;    ;//Interpolate to a surface
;    triangulate, reform(sImported.xData), reform(sImported.yData), angles,
b
;    limits = [min(sImported.xData,/NaN), min(sImported.yData,/NaN), $
;      max(sImported.xData,/NaN), max(sImported.yData,/NaN)]
;    zGrid = trigrid(sImported.xData, sImported.yData, sImported.zData, $
;      angles, [0,0], limits, $
;      NX=100, NY=100,$
;      XGRID=xGrid, YGRID=yGrid, MISSING=!Values.F_NaN)
;    oGraphic = obj_new('IDLgrSurface', DATAX=xGrid, DATAY=yGrid,
DATAZ=zGrid)
;   end
;   "SCAL_IMAG_2S0T": begin
;    ;//Map to a polygon for display
;    imgType = strlowcase(strmid(sImported.fileName[0], $
;      (rstrpos(sImported.fileName[0],".") + 1),
```

```
strlen(sImported.fileName[0])))
;    if(imgType eq "jpg") then $
;     oImage = obj_new('IDLgrImage', DATA=sImported.data, /INTERPOLATE) $
;    else if(imgType eq "gif") then $
;     oImage = obj_new('IDLgrImage', DATA=sImported.data, /INTERPOLATE)
;    oGraphic = obj_new('IDLgrPolygon', TEXTURE_MAP=oImage,
TEXTURE_INTERP=1)
;   end
;  endcase
;  oGraphicsModel->add, oGraphic
;
;  ;//Let's say the data imported was a US ARMY GMS 3D grid file
;  IDL> help, sImported, /STRUCT
;  ** Structure <1354308>, 13 tags, length=2022472, refs=1:
;     DATA            DOUBLE    Array[101, 61, 41]
;     XDATA           DOUBLE    Array[101]
;     YDATA           DOUBLE    Array[61]
;     ZDATA           DOUBLE    Array[41]
;     TDATA           BYTE        0
;     ANCILLARYDATA   BYTE        0
;     DATANAME        STRING    'c532(1/m)'
;     FILENAME        STRING    Array[1]
;     DATACLASS       STRING    'SCAL_GRID_3S0T'
;     DATATYPE        STRING    'USA_GMS3DG'
;     POLYLINES       BYTE        0
;     ERROR           BYTE        0
;     CANCEL          BYTE        0
;
; MODIFICATION HISTORY:
;  Written by: Todd Bowers, 13Feb2001. (tbowers@nrlssc.navy.mil)
;
;  Modified:
;   Added US Army Corps of Eng. Groundwater Modeling System 3D Grid
format(.3dg) - 13Feb2001 Todd Bowers
;   Added US Army Corps of Eng. Groundwater Modeling System 2D Grid
format(.3dg) - 13Feb2001 Todd Bowers
;   Added US Navy Modaps Automated Processing System(MAPS) ocean profile
data format - 14Feb2001 Todd Bowers
;
; This software is provided as is without any express or implied warranties.
;-
;//////////////////////////
```

Feedback would be appreciated.
Todd Bowers