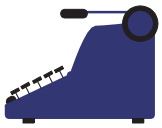
Point Raster! Poly

A Technical Friend for PC ARC/INFO® Users and Other ESRI Desktop Software

Volume **7** Number **2**

Inside:

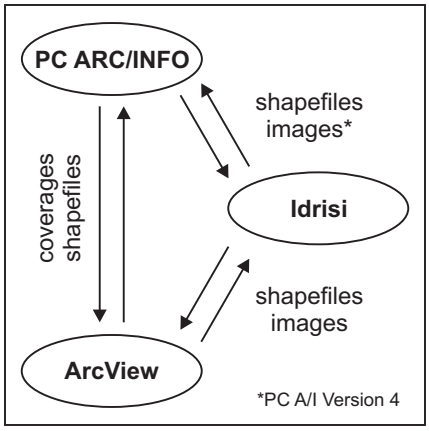


Idrisi: Third Leg in the Tripod

Editor

Many GIS professionals raised in the vector paradigm have become painfully aware of something missing in their lives.

How many of you have spent hours developing a map only to be asked: "Looks great, but could you plot that on a topo map?" For me, ArcView was useless until Version 2.1 came out packaged with Avenue, allowing me to create a tool to register scanned images. Even then, the task of scanning and stitching pieces of quad maps was a nightmare. Needless to say, as USGS DRGs (and increasingly DOQs) have become available, use of them as base layers for maps has risen dramatically.



But raster-based GIS involves more than pretty pictures. Host ARC/INFO users familiar with GRID know the power of grid algebra, cost, and surface functions. However, even GRID out-of-the-box lacks most of the functionality associated with image processing packages such as Erdas Imagine or TNTmips.

Idrisi, produced by Clark Labs (a nonprofit project within the Graduate School of Geography at Clark University), is a low-cost raster-based GIS package that combines a great deal of grid/image processing functionality (see comparison chart on p. 3). Although developed chiefly for educational purposes, it has proven an excellent toolkit for low-budget GIS shops around the world.

For me, the chief drawback of Idrisi in the past has been its basic incompatibility with ArcView and PC ARC/INFO. This has changed with two developments: 1) the release of Idrisi 2.0 for Windows, which can read and write shapefiles, and 2) the creation of the Idrisi image extension for ArcView 3.1.

Perhaps the greatest milestone will be the release of PC A/I Version 4.0, which will support images (including Idrisi) as background layers. As a result, AV's significance as an editing tool will shrink considerably. AV

(continued on p. 2)

Icon Key

Macro	Good Ideas
Buried Jewel	Positions
Lesson / Tutorial	Calendar
Editorial	Doctor's Column
Review	Letters
Off the Wire	Announcements

(continued from p. 1)

was never really intended for large-scale data creation and, for many users, should revert to a more appropriate role: data browsing and integration, simple queries, and quick, flexible map creation.

What Idrisi Can (and Can't) Do

The comparison table on p. 3 of Idrisi vs. the ArcView Spatial Analyst and Image Analysis extensions should give you a good idea of what Idrisi is and isn't capable of doing. One of Idrisi's greatest strengths is its ability to project images[1]; the only way you could do that in ArcView would be to project an array of control points and then warp the image to them. On the other hand, Idrisi currently lacks the ability to produce contours from DEMs, or to perform direct vector-raster overlays, two of the more powerful features of Spatial Analyst.

Idrisi image files (.IMG) may have 8-bit (byte), 16-bit (integer), or 32-bit (floating point) values, and may be up to 32677x32677 in size. Also, Idrisi image files support run-time compression. For attribute files, Idrisi supports ASCII (.VAL, .FXL), Access (.MDB), and xBASE (.DBF) formats.

Other (and Future) Idrisi Developments

Version 3.0 of Idrisi, due out by mid-1999, will have a number enhanced features, including full 32-bit coding, 24-bit color image support, and direct DRG import. Idrisi's API will also be restructured and made less quirky, facilitating the development of custom applications within the Idrisi framework.

Clark Labs has entered the vector-based GIS realm through its introduction of CartaLinx, a topological vector data creation and editing package which could prove competitive to DAK. Future planned releases include AnaLinx and Surface Analyst, add-ons for vector overlay analysis, TIN creation, contouring, and visualization.

---

[1]Be aware that Idrisi's projection engine is not identical to that of ARC/INFO or ArcView, leading to slightly different results (though generally less than 0.5 meters). Also, make sure that the projection parameters are the same: a difference in a scale factor of 0.00003333 could lead to a 60 foot shift or more.

PLP

POINT LINE POLY is published by Pierssen Publishing, 3125 West Wilson Drive, Flagstaff, AZ USA 86001. Domestic subscriptions: 6 issues $35. Foreign: 6 issues $50. Single Issues $7 domestic, $10 foreign. Send subscriptions and fulfillment questions to *PLP* Circulation Dept., 3125 West Wilson Drive, Flagstaff, AZ USA 86001. Or call (520) 774-7905. Submit material and technical questions to The Editor, at the address above.

ISSN: 1099-2324

POSTMASTER: Send address changes to:

Point Line Poly
Pierssen Publishing
3125 West Wilson Drive
Flagstaff, AZ USA 86001

e-mail: piersen@primenet.com

Say, what? "A flat world was ours and everything in it had a name once and all the names were ours once. With perspective, names escape from the paper and scatter into the minds of men so they can never be held down again."

William S. Burroughs, *The Western Lands*

	Idrisi	Spatial Analyst	Image Analysis
Raster Attribute Table	•	•	
Number of Raster Import Formats	15	5	14
Vector to Raster Conversion	•	•	
Raster to Point,Poly Conversion	•	•	
Raster to Line Conversion		•	
Point Surface Interpolation: Methods	1	4	
Surface Interpolation from Contours	•		
Density Surface from Points		•	
Euclidean Distance	•	•	
Cost Distance/Path Analysis	•	•	
Allocation	•	•	
Contour Generation		•	
Shaded Relief / Ilumination	•	•	
Orthographic Display / Image Draping	•		
Histograms	•	•	•
Profiles	•	•	?
Grid Algebra	•	•	
Cross-tabulation	•	•	
Vector-Raster Overlay		•	
Aspect	•	•	
Surface Curvature		•	
Slope Method: Neighbors	4	8	
Viewshed Analysis	•	•	
Local/Zone Statistics	•	•	
Autocorrelation	•		?
Aggregation	•	•	•
Pattern Analysis	•	•	
Filter Analysis: Number of Types	10	2	3
Boundary Functions	•	•	•
Regression Analysis	•		
Change Detection	•		•
Watershed Determination	•	•	
Stream Flow Modelling		•	
Flow Accumulation		•	
Projection / Datum Transformation	•		
Rectification / Resampling	•	•	•
Store/Display Geometry and Combine			•
Bands w/o Creating New Image Files			
Brightness/Contrast/Stretch	•		•
Reclassification	•	•	•
Unsupervised Classification	•		•
Hyperspectral Classification	•		?
NDVI Analysis	•		•
Fuzzy Classifiers and Bayesian Logic	•		
Multi-Criteria Evaluation Tools	•		
Merge	•	•	?
Extract by Rectangle	•	•	?
Extract by Shape or Value		•	
Application Programming Interface	•		
Object-Oriented Access via Avenue		•	•

Functionality of the Image Analysis extension is not known in detail by the reviewer—known functional categories were determined from the white paper published by ESRI and ERDAS. See p. 15 for a link to Idrisi's detailed online command description.

Idrisi and Digital Raster Graphics

Importing DRG Images into Idrisi

Clark Labs has announced the eventual release of a module to import DRG images directly. Until then, the workaround is to use a graphics utility package to convert the image to a compatible format. *(Image Alchemy is highly recommended because it's fairly inexpensive, has a command line interface, can be used for batch processing, supports a multitude of formats, and is VERY FAST.)*

Windows bitmap (BMP) format is generally preferable to uncompressed TIFF because the BMPIDRIS module runs more quickly than TIFIDRIS.

Macro Example:

```
BMPIDRIS x 2 i01 i01 drg
```

Note that Idrisi's online documentation for this macro command is incorrect. The argument specifying the name of an output palette file (in this case "drg") is required, and the output palette file will be written to the IDRISIW directory.

Once the image is imported, its georeferencing information must be entered into the documentation file. Two programs are available for download[1], REDOC.EXE and REDOCW.EXE. The program REDOC.EXE, executable from the MS-DOS prompt, reads the .FGD metadata file of a DRG image and enters the appropriate information into the .DOC file, optionally applying a y-shift to the coordinates. For example:

```
redoc i01 i01
```

REDOC only affects the title, min. X, max. X, min. Y, max. Y, and resolution entries of the documentation file; other data (e.g. ref. system) will need to be entered manually. REDOCW works similarly save that it reads georeferencing information from the .TFW world file.

WARNING: *Although the projection and datum may be read from the .FGD file, the ellipsoid may not be correctly reported. For example, O35113G7.FGD reports "GRS 1980" but is actually "Clarke 1866". Thus the appropriate parameter file is US27TM12 rather than US83TM12.*

Concatenating DRG Images in Idrisi

Because DRG images contain information outside the actual map area (titles, legends, etc.) it is necessary to set those cells to 0. This may be done by means of POLYRAS and OVERLAY.

WARNING: *Avoid packbit compression during this process, as it may lead to image shifting.*

1) Create a lat/long vector file representing the map area.

Idrisi Basic Concepts:

The Idrisi-oriented articles in this issue assume some working knowledge of Idrisi and its file types.

For those of you who just want to get the general idea of what's going on, this table lists some of the more basic file types (there are many others!).

File Type	Extension	Description
Image File	IMG	Stores image data
Documentation File	DOC	Stores georeferencing info and metadata
Vector File	VEC	Stores vector feature geometry, may be ASCII or binary
Vector Documentation File	DVC	Stores metadata
Binary Palette File	SMP	Stores colormap for pseudocolor rendering
ASCII Palette File	PAL	Older version of colormap
Reference System Parameter File	REF	Stores coordinate system parameters for the projection engine.
Idrisi Macro Language	IML	Rudimentary batch processing language for automating most of Idrisi's commands.

The program FRAME.EXE[1], executable from the MS-DOS prompt, reads the .FGD metadata file of a DRG image and writes out a vector file representing the map area in lat/long. For example:

```
frame i01 i01
```

i01.vec:

```
1 5
-113.875000 35.750000
-113.875000 35.875000
-113.750000 35.875000
-113.750000 35.750000
-113.875000 35.750000
0 0
```

i01.dvc:

```
file title  : Frame file
id type     : integer
file type   : ascii
object type : line
ref. system : latlon27
ref. units  : deg
unit dist.  : 1.0000000
min. X      : -113.875000
max. X      : -113.750000
min. Y      : 35.750000
max. Y      : 35.875000
pos'n error : unknown
resolution  : unknown
```

Note that the default ref. system is LATLONG. In the above example, the DRG uses the Clarke 1866 spheroid; thus the .DVC file has been edited to refer to a parameter file called LATLON27:

latlon27.ref:

```
ref. system : Geodetic Coordinates
(Latitide/Longitude)
projection  : none
datum       : NAD27
delta WGS84 : -8 160 176
ellipsoid   : Clarke 1866
major s-ax  : 6378206.40
minor s-ax  : 6356583.80
origin long : 0
origin lat  : 0
origin X    : 0
origin Y    : 0
scale fac   : 1.0
units       : deg
parameters  : 0
```

2) Project the vector file to the DRG's reference system.

Macro Example:

```
PROJECT x 2 i01 latlon27 i01p us27tm12
```

3) Convert the vector file to a raster mask and overlay the DRG image.

Use INITIAL to create an image from the DRG with a value of 0. Then use POLYRAS to add the vector file data to the image. Because the polygon's value is set to 1 and the INITIAL image is set to 0, the result is a mask that may be multiplied against the DRG image to zero out the margin area.

Macro Example:

```
INITIAL x i01v 3 1 0 1 i01 unspecified
POLYRAS x i01p i01v
OVERLAY x 3 i01 i01v i01ov
```

4) CONCAT the resulting images using the transparent option.

Macro Example:

```
CONCAT x # 4 drg 2 i01ov i02ov i03ov
i04ov
```

WARNING: *Make sure you have downloaded the latest Idrisi update, as it fixes a CONCAT bug* *(see p. 15 for more info).*

[The remaining steps convert the margin areas back to white; they are optional.]

5) CONCAT the raster masks using the transparent option.

Macro Example:

```
CONCAT x # 4 drgv 2 i01v i02v i03v i04v
```

(See WARNING above.)

6) Use RECLASS to swap the 0 and 1 values.

Macro Example:

```
RECLASS x i drgv drgr 2 0 1 2 1 0 1
-9999
```

(Continued on p. 8)

Importing Contours from PC ARC/INFO

Users who maintain topographic datasets in PC ARC/INFO may bring them into Idrisi to create a surface. Idrisi's surface functions (see p. 3) include slope/aspect analysis, hillshading, and watershed delineation. The basic process is as follows:

1) In PC ARC/INFO, use ARCSHAPE to export the contour data.

```
ARCSHAPE topo topo line
```

2) In Idrisi, use SHAPEIDR to convert the shapefile to an Idrisi vector file.

WARNING: *Do NOT use the macro version of this command, as it has a serious bug and will wipe out the .DBF file!* The command may be executed through the API (see p. 8) but the .MDB values file will not automatically be created.

3) Use INITIAL to create a base image for the area (the vector's document file contains min and max coordinate information).

Macro Example:

```
INITIAL x topo 1 1 0 2 815 481 plane
ft 11540 19690 9970 14780 1.0 ft
```

4) Use LINERAS to add line values to the image.

Macro Example:

```
LINERAS x topo topo
```

5) In DATABASE WORKSHOP, open the vector attribute table and assign the elevation field values to the image *(see illustration on p. 7)*. Use IDR_ID as the feature definition field. Because DW requires a (2-byte) integer field to assign values to an integer image, you may need to add an integer field and populate it from the elevation field.

SQL Example:

```
UPDATE topo
SET [elev2] = [elev]
```

6) Use INTERCON to interpolate the surface elevations from the contours. To estimate corner values, you can use the cursor query tool on the display created by the ASSIGN operation in step 5.

Macro Example:

```
INTERCON x topo2 dem 0 1628 1466 1402
1417
```

7) Apply FILTER as desired to smooth out any angular effects.

Macro Example:

```
FILTER x dem dem2 1
```

8) [Optional] Crop the DEM to the project boundary. If the project area is irregularly shaped, create a boundary polygon in PC ARC/INFO. The polygon should be completely inside the contours so that edge effects may be clipped out. Convert the coverage to a shapefile and import it into Idrisi.

Use INITIAL to create an image from the DEM with a value of 0. Then use POLYRAS to add the vector file data to the image. Because the polygon's value (IDR_ID) is set to 1 and the INITIAL image is set to 0, the result is a mask that may be multiplied against the DEM image to zero out the outlying areas.

Macro Example:

```
INITIAL x demv 3 1 0 1 dem2
unspecified
POLYRAS x clip demv
OVERLAY x 3 dem2 demv demclip
```

Finally, before performing any surface analysis, you will want to do the following in DOCUMENT: a) make sure that the value units are set appropriately, and b) set the flag value to "0" and the flag definition to "background".

PLP

Using Idrisi Images in ArcView

Displaying Idrisi Images in ArcView

ArcView 3.1 users may display and print Idrisi images directly using the AVIdris extension[1]. Supported data types are byte, integer, and real, and supported file types are binary and packed binary (not ASCII).

By default, images are displayed in gray scale. If a palette (.SMP or .PAL) file is present with the same name as an image, the image will be displayed using that color scheme. As with the Display Launcher in Idrisi, integer and real images are scaled to 0-255; if an SMP palette file is present, the lower and upper autoscale limit values will be used for scaling.

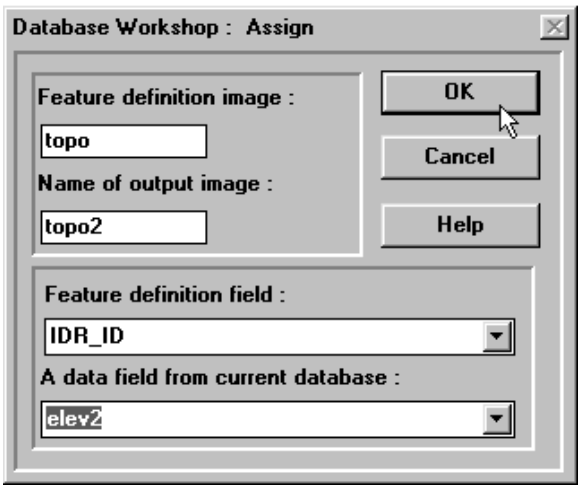
AVIdris takes advantage of AV 3.1's new ImgDLL class, which is discussed in greater depth on p. 12.

Exporting Idrisi Images to ArcView

There are two basic export options: ERDAS and TIFF. ERDAS format may be exported in one step, but cannot be compressed and requires setting up a colormap in ArcView. TIFF format may be compressed and requires no colormap, but exporting and georeferencing is a two-step process. Of the two formats, ArcView seems to display TIFF much more efficiently.

Exporting to ERDAS

1) Use ERDIDRIS to export the image to a .GIS file.



Macro Example:

```
ERDIDRIS x 3 drg drg.gis
```

2) Use PALETTE WORKSHOP to export the palette (.SMP) to a .PAL file (no Macro command available).

3) In ArcView, run the script View.LoadGIS[2] to load the image as a theme and assign color values from the .PAL file.

USGS Digital Raster Graphic (DRG) images use a standard palette. The script View.LoadDRG[2] will load the image as a theme in ArcView and assign the standard DRG palette without need to create a .PAL file.

Exporting to TIFF

1) Use TIFIDRIS to export the image to a .TIF file, specifying the name of the palette to be used.

Macro Example:

```
TIFIDRIS x 2 drg drg 90 drg
```

2) Create a world file (.TFW) containing the georeferencing information. The program WORLD.EXE[2], executable from the MS-DOS prompt, reads the document file of the image and generates the appropriate world file. For example:

```
world drg drg
```

3) [Optional, highly recommended] If compression is desired, use a third-party program (e.g. Corel Photo-Paint or Image Alchemy) to convert the image to TIFF with packbit compression. Depending on the compression ratio, ArcView can handle packbit compressed TIFF images much more efficiently than uncompressed.

[1]The extension may be downloaded at:
www.primenet.com/~piersen/arcview/view.htm
[2]May be downloaded at:
www.primenet.com/~piersen/idrisi/arcview.htm

PLP

Idrisi's API

Although Idrisi for Windows will accept a list of macro commands in a text file (default .IML extension), the ability to write even relatively simple programs is virtually nonexistent. Fortunately, Clark Labs has made an API available for free download[1], which programmers in Visual C++/Basic, Delphi, or other Windows-compatible packages may use.

To install the API, all that is necessary is to copy MERCURY.DLL and MERCUR32.DLL to the Windows system directory; the existing DLLs may be renamed to a .BAK extension.

The file "api documentation.rtf" documents the functions that are available. Also included are declaration and/or library files for Visual Basic, Borland Pascal, Delphi, and C++ Builder.

Using the API in Visual C++

Visual C++ compatible .LIB and .H files are not included with the API; therefore, users must link to the DLL explicitly. Because explicit links are somewhat clunky to code, it may be desirable to create a class that handles them. The following header file is an example of a class called Idrapp that handles the minimum functions necessary to check if Idrisi is active, launch Idrisi, set the working directory, register a client, launch a module (macro statement), monitor the progress of a launched module, unregister a client, and close Idrisi[2]:

idrapp.h:

```
#define PROCSTATUS_ACTIVE 1
#define PROCSTATUS_NORMAL_TERMINATE 2
#define PROCSTATUS_ERROR_TERMINATE 3
#define REPORT_TYPE_WORKING 1
#define REPORT_TYPE_PERCENTDONE 2
#define REPORT_TYPE_PASS_X_OF_N 3
#define REPORT_TYPE_PASS_TOTALUNKNOWN
4
#define REPORT_TYPE_COMPLEXPASS 5
#define MONITOR_FROM_CLIENT_ONLY 0
#define MONITOR_FROM_IDRISI 1
#define SHOW_IDRISI_HIDDEN 0
#define SHOW_IDRISI_NORMAL 1
#define SHOW_IDRISI_MINIMIZED 2
#define SHOW_IDRISI_MAXIMIZED 3

struct ProgStruct
{
  short Status;
  short ReportType;
  short PassNum;
  short TotalPasses;
  float PercentDone;
  short ErrorCode;
  char ErrorFile[256];
  char ErrorMessage[256];
  char SubstString1[256];
  char SubstString2[256];
};

class Idrapp
{
protected:
   HINSTANCE hinstLib;
public:
   Idrapp();     // constructor
   short IsIdrisiPresent();
   short LaunchIdrisi(short);
   short SetDataDirectory(char *);
   short RegisterClient();
   short LaunchModule(short, short,
char *, char *, char *, char *, short
*);
   short GetProgress(short, short,
ProgStruct *);
   short UnRegisterClient(short);
```

(Continued from p. 5)

7) Use OVERLAY with the Cover option to create the final image.

Macro Example:

```
OVERLAY x 7 drgr drg drgw
```

As you can see, this process lends itself fairly well to automation.

[1]These programs may be downloaded at: www.primenet.com/~piersen/idrisi/drg.htm

PLP

```
   short CloseIdrisi();
   ~Idrapp();    // destructor
};
```

Although **enum** would be more elegant, **#define** is used to declare enumerations because the API calls use **short** rather than **int** data types. Also note the class constructor and destructor; these are used to load and unload MERCUR32:

idrapp.cpp:

```
#include <windows.h>
#include "idrapp.h"

typedef short (CALLBACK*
IsIdrisiPresentFuncType)();
typedef short (CALLBACK*
LaunchIdrisiFuncType)(short);
typedef short (CALLBACK*
SetDataDirectoryFuncType)(char *);
typedef short (CALLBACK*
RegisterClientFuncType)();
typedef short (CALLBACK*
LaunchModuleFuncType)(short, short,
char *, char *, char *, char *, short
*);
typedef short (CALLBACK*
GetProgressFuncType)(short, short,
ProgStruct *);
typedef short (CALLBACK*
UnRegisterClientFuncType)(short);
typedef short (CALLBACK*
CloseIdrisiFuncType)();

Idrapp::Idrapp() {
   hinstLib = LoadLibrary("mercur32");
   return;
}

Idrapp::~Idrapp() {
   if (hinstLib == NULL) return;
   BOOL fFreeResult =
FreeLibrary(hinstLib);
   return ;
}
```

The remaining class functions merely provide the front end to the DLL's exported functions, for example:

```
short Idrapp::GetProgress(short
ClientId, short ProcID, ProgStruct *
Prog) {
   GetProgressFuncType f;
   if (hinstLib == NULL) return 0;
   f = (GetProgressFuncType)
GetProcAddress(hinstLib,
"GetProgress");
   if (!f) return 0;
   return f(ClientId, ProcID, Prog);
}
```

At this point we're ready to develop our first Idrisi application! Let's write a console app that will:

1) Launch Idrisi if necessary
2) Set the working directory to "c:\0home\test\idr_vc"
3) Execute a module command, equivalent to the following macro statement:

```
project x 1 westboro spc27ma1 testwb
us27tm19 279078 290565.3 4678095
4686651 614 486 0 1
```

test1.cpp:

```
#include <windows.h>
#include <iostream.h>
#include "idrapp.h"

Idrapp i;

void main() {

   ProgStruct Prog;
   short result, ClientId, ProcId;
   short PtHinst = 0;
   short show = SHOW_IDRISI_MINIMIZED;
   short mon = MONITOR_FROM_IDRISI;

   char *dir =
"c:\\0home\\test\\idr_vc\\";
   char *module = "project";
   char *opt = "1 westboro spc27ma1
testwb us27tm19 279078 290565.3
4678095 4686651 614 486 0 1";
   char *title = "";
   char *units = "";

   if (! i.IsIdrisiPresent())
      result = i.LaunchIdrisi(show);
   i.SetDataDirectory(dir);

   ClientId = i.RegisterClient();
   ProcId = i.LaunchModule(ClientId,
mon, module, opt, title, units,
&PtHinst);
   do
   {
      result = i.GetProgress(ClientId,
```

```
ProcId, &Prog);
   }
   while (Prog.Status ==
PROCSTATUS_ACTIVE);
   result =
i.UnRegisterClient(ClientId);
   cout << "Done.\n";
   return;
}
```

Note that even though the SHOW_IDRISI_MINIMIZED option is specified, Idrisi won't run minimized. This is a known bug that hopefully Clark Labs will fix when Version 3 comes out in mid-1999. Right now, I recommend just keeping a copy of Idrisi active rather than launching it. (Don't forget that you can also add custom commands to Idrisi's menu!)

As you build more advanced applications, you're going to want to take advantage of error reporting, progress reporting (to Idrisi), image/vector documentation, and other functions available in the API. All of these functions are described in the documentation.

Progress Reporting

One thing to watch out for in monitoring progress is that ProgStruct values to be supplied to SetProgress are not necessarily the same as those returned by GetProgress[3]. For example, for REPORT_TYPE_PERCENTDONE GetProgress populates PercentDone with 0.0-100.0 while SetProgress requires 0.0-1.0. (For REPORT_TYPE_COMPLEXPASS, returned values of PassNum and PercentDone are even more bizarre.) The following code is an example of reading the progress of a module and then supplying it to Idrisi.

```
result = i.GetProgress(ClientId,
ProcId, &Prog);
Prog2.Status = Prog.Status;
Prog2.ReportType = Prog.ReportType;
switch (Prog.ReportType)
{
case REPORT_TYPE_WORKING:
   Prog2.PassNum = 1;
   Prog2.TotalPasses = 1;
   Prog2.PercentDone = 0.0;
   break;
case REPORT_TYPE_PERCENTDONE:
   Prog2.PassNum = 1;
   Prog2.TotalPasses = 1;
   Prog2.PercentDone =
Prog.PercentDone / 100;
   if (Prog2.PercentDone < 0.0)
Prog2.PercentDone = 0.0;
   if (Prog2.PercentDone > 1.0)
Prog2.PercentDone = 1.0;
   break;
case REPORT_TYPE_PASS_X_OF_N:
   Prog2.PassNum = Prog.PassNum;
   if (Prog2.PassNum < 0)
Prog2.PassNum = 0;
   Prog2.TotalPasses =
Prog.TotalPasses;
   Prog2.PercentDone = 0.0;
   break;
case REPORT_TYPE_PASS_TOTALUNKNOWN:
   Prog2.PassNum = Prog.PassNum;
   if (Prog2.PassNum < 0)
Prog2.PassNum = 0;
   Prog2.TotalPasses = 0;
   Prog2.PercentDone = 0.0;
   break;
case REPORT_TYPE_COMPLEXPASS:
   calc = (float)(Prog.PassNum - 2080)
/ 100;
   Prog2.PassNum = (short) calc + 1;
   Prog2.TotalPasses =
Prog.TotalPasses;
   if (Prog2.PassNum >
Prog2.TotalPasses)
      Prog2.PassNum =
Prog2.TotalPasses;
   calc = -207900 - Prog.PercentDone;
   calc = calc - ((Prog2.PassNum - 1)
* 9900);
   Prog2.PercentDone = calc / 99.0f;
   if (Prog2.PercentDone < 0.0)
Prog2.PercentDone = 0.0;
   if (Prog2.PercentDone > 100.0)
Prog2.PercentDone = 100.0;
   break;
}
result2 = i.SetProgress(ClientId2,
StatId, &Prog2);
```

Other Caveats

Warning Messages: Warning messages, such as those issued by BMPIDRIS and SURFACE, currently cannot be disabled in the API. Attempts to circumvent this by monitoring from the client will lead to "exception eedfadeH" in MERCUR32.DLL if GetProgress is called after LaunchModule.

CloseIdrisi: Idrisi may think that Clients are still active, even after unregistering them.

Accessing Idrisi's API from ArcView

Although the MERCUR32 DLLProcs are accessible from ArcView, Avenue cannot manipulate data structures such as ProgStruct. For example, it is not possible to use the SetProgress DLLProc. However, in situations where retrieving extended data is not essential (such as GetProgress), a phony buffer string may be used instead:

Idrisi.Go:

```
ModName = SELF.Get(0)
Cmdln = SELF.Get(1)
OutTitle = SELF.Get(2)
OutUnits = SELF.Get(3)
theDLL =
DLL.Make("c:\windows\system\mercur32.d
ll".AsFilename)

'register client
RegisterClient = DLLProc.Make(theDLL,
"RegisterClient",
#DLLPROC_TYPE_INT16,{})
theCID = RegisterClient.Call({})
if (theCID = 0) then
   IsIdrisiPresent = nil
   RegisterClient = nil
   theDLL = nil
   av.PurgeObjects
   return "Could not register client"
end

'launch the process
LaunchModule =
DLLProc.Make(theDLL,"LaunchModule",#DL
LPROC_TYPE_INT16,
{#DLLPROC_TYPE_INT16,
#DLLPROC_TYPE_INT16,
#DLLPROC_TYPE_STR, #DLLPROC_TYPE_STR,
#DLLPROC_TYPE_STR, #DLLPROC_TYPE_STR,
#DLLPROC_TYPE_PINT16})
MonOp = 1
thePint = 0
parm_list = {theCID, MonOp, ModName,
Cmdln, OutTitle, OutUnits, thePint}
thePID = LaunchModule.Call(parm_list)

'wait until done
Msg = "OK"
dummy_P = String.MakeBuffer(1088)
GetProgress =
DLLProc.Make(theDLL,"GetProgress",
#DLLPROC_TYPE_INT16,
{#DLLPROC_TYPE_INT16,
#DLLPROC_TYPE_INT16,
#DLLPROC_TYPE_STR})
status = GetProgress.Call({theCID,
thePID, dummy_P})
while (status = 1)
   status = GetProgress.Call({theCID,
thePID, dummy_P})
end
if (status = 3) then
   Msg = "Process terminated with
error"
end

'unregister client and clean up
UnRegisterClient = LLProc.Make(theDLL,
"UnRegisterClient",
#DLLPROC_TYPE_INT16,
{#DLLPROC_TYPE_INT16})
result =
UnRegisterClient.Call({theCID})
if (result = 0) then
   if (Msg = "OK") then
      Msg = "Could not unregister
client"
   else
      Msg = Msg + NL + "Could not
unregister client"
   end
end
RegisterClient = nil
UnRegisterClient = nil
LaunchModule = nil
GetProgress = nil
theDLL = nil
av.PurgeObjects
return Msg
```

Conclusion

Idrisi's API, while it does have its quirks and bugs, is well worth learning. The ability to develop custom procedures and algorithms is an important feature of any GIS, and Clark Labs has demonstrated their commitment to this. Hopefully, with the release of Version 3.0, the problems currently encountered will be eliminated.

---

[1] See p. 15.

[2] A complete version of the Idrapp class is available for download at: www.primenet.com/~piersen/idrisi/idrisi.htm

[3] Thanks to Stewart Dibbs of VYSOR Integration, Inc. (www.vysor.com) for his investigations of the bugs and quirks in Idrisi's API.

PLP

Supporting Custom Image Formats in ArcView 3.1

With the introduction of the ImgDLL class in ArcView 3.1, it is now possible for developers to add support for any image format. Two components are necessary:

· A DLL that ArcView can call to retrieve image info and data.

Three functions are required:

1) int IsValidFile(path) Returns TRUE if the input path name points to a valid image file. FALSE is returned otherwise.

2) int QueryImageInfo(path, nbands, width, height, MBR) Queries the input image defined by path for the number of bands, image width in pixels, image height in pixels and the geo-referenced Minimum Bounding Rectangle (MBR). The MBR is expressed as left, top, right, and bottom. TRUE is returned on success, FALSE is returned otherwise.

3) int GetImage(path, out_path, clip_MBR, out_width, out_height) Extracts an image from path into the temporary image out_path in the output format defined when the format is registered with ImgDLL with the RegisterFormat class request. The output format must be either a BIL, BIP, or BSQ image. The clip_MBR is the extent needed from the image in map coordinates. The out_width and out_height is the needed size for the temporary output image in pixels. The MBR is expressed as left, top, right, and bottom. TRUE is returned on success, FALSE on error.

· An AVX file that registers the image format with ArcView and points to the DLL

To load an image theme, the following events take place:

1) As the theme browser scans the current directory, ArcView passes the path of each file with a registered extension to the DLL to check if it's valid. If the returned value is true, the file is added to the image theme list; otherwise, it's skipped.

2) When the theme is loaded, ArcView queries the DLL for the image bounds and other attributes.

3) When the theme is first made visible, ArcView initializes the necessary file handles by sending a request for a zero row/column image. Then, a second request is made for the desired image, which is subsequently displayed in the view. Temporary image files are stored in the $TEMP directory.

Because the basic Idrisi image format is simple, it makes a good example of developing a custom image extension. In coding the DLL routines, I chose to take an object-oriented approach:

avidris.cpp:

```
#include <windows.h>
#include "idrimg.h"

int IsValidFile(char * path);
int QueryImageInfo(char * path, long *
nbands, long * width, long * height,
double * mbr);
int GetImage(char * path, char *
out_path, double * clip_mbr, long
out_width, long out_height);

int IsValidFile(char * path)
{
   Idrimg theImg(path);
   return theImg.IsValid();
}

int QueryImageInfo(char * path, long *
nbands, long * width, long * height,
double * mbr)
{
   Idrimg theImg(path);
   return theImg.Query(nbands, width,
height, mbr);
}

int GetImage(char * path, char *
out_path, double * clip_mbr, long
out_width, long out_height)
{
   Idrimg theImg(path);
   return theImg.WriteBIP(out_path,
clip_mbr, out_width, out_height);
}
```

```
/* This function is optional
int QueryBandStats(char * path, long
band, double * stats)
{
   Idrimg theImg(path);
   return theImg.BandStat(band,stats);
}
*/
```

This code can easily be adopted to any class name that you care to make. The advantages are obvious:

1) You can revise and improve your class and its member functions without having to rewrite the front end.

2) If a particular image class already exists, you can derive a new class from it that incorporates the necessary functions.

The following file is used to define the exports:

avidris.def:

```
LIBRARY    AVIDRIS
DESCRIPTION "Adds Idrisi image support
to ArcView"
EXPORTS
   IsValidFile
   QueryImageInfo
   GetImage
```

The Idrimg class declaration is as follows:

idrimg.h:

```
typedef enum {
   IMG_NODTYPE  = 0,
   IMG_BYTE     = 1,
   IMG_INTEGER  = 2,
   IMG_REAL     = 3,
} IMGDataType;
typedef enum {
   IMG_NOFTYPE  = 0,
   IMG_ASCII    = 1,
   IMG_BINARY   = 2,
   IMG_PACKED   = 3,
} IMGFileType;
typedef enum {
   IMG_NOPALETTE = 0,
   IMG_SMP       = 1,
   IMG_PAL       = 2,
} IMGPaletteType;
class Idrimg
{
protected:
   char Path[_MAX_PATH];
   int Valid;
   IMGDataType DataType;
   IMGFileType FileType;
   long Rows;
   long Cols;
   double MinX;
   double MinY;
   double MaxX;
   double MaxY;
   double MinVal;
   double MaxVal;
   IMGPaletteType Palette;
public:
   Idrimg(char * img_path);   //
constructor
   int IsValid();
   int Query(long * nbands, long *
width, long * height, double * mbr);
   int WriteBIP(char * out_path,
double * clip_mbr, long out_width,
      long out_height);
/* This function is optional
   int BandStat(long band, double *
stats);
*/
};
```

Rather than list out the (somewhat lengthy) source code for the class member functions, I will discuss the basic approaches that were taken, since approaches may vary with different image formats[1]. The output image is always 1-band, 8-bit BIP format[2], which by default is displayed in ArcView as gray scale. If a color palette (.SMP or .PAL) file is present with the same name as the image, a corresponding colormap file is written for the output image. Integer and real image values are scaled to 0-255 or to the autoscale values in the .SMP file (this is consistent with Idrisi's Display Launcher).
A different approach may be preferable depending on the image type; for example, 24-bit images could be written as a 3-band, 8-bit BIP. This is the case with the Mr. SID extension.

Constructor:

1) See if the .IMG and .DOC files exist.

2) Read the pertinent data from the .DOC file into class attributes.

3) See if a corresponding .SMP or .PAL file exists.

IsValid:

Return the Valid attribute.

Query:

Assign 1 to nbands, the Cols attribute to width; the Rows attribute to height; and the MinX, MaxY, MaxX, and MinY attributes to mbr[].

WriteBIP:

1) Determine image parameters. If ArcView is initializing the image or the clip_mbr is completely outside, set degenerate image parameters (equal to pixel 0,0). If ArcView is requesting a smaller pixel size than the input image, adjust the parameters to use actual size pixels; this reduces write operations[3].

2) Allocate memory for I/O buffers and open the input/output files.

3) Write the header file.

4) If a palette exists, read it, write the corresponding colormap file, and set the scaling parameters.

5) For each cell of the output BIP, find the nearest neighboring cell value of the input image. If the image data format is integer or real, scale the value appropriately using the MinVal and MaxVal attributes.

Uncompressed Idrisi images are basically lists of **unsigned char**, **short**, or **float** values. In this case, the simplest approach is to calculate where the file pointer should go for each row and read in the number of values needed to resample to the output row. With compressed images, a different approach is necessary: the image must be read from the beginning until the start value is reached, and must continue to be read until the end value is reached. Fortunately, a well-compressed image can be read in more quickly, and will often display faster in ArcView than the uncompressed counterpart.

The BIL/BIP/BSQ format is very easy to write. ArcView's online help gives the specification (see "extended image formats" in the index).

Debugging

One problem in debugging the DLL is finding out how it's being used. For that reason, it can be useful to trap parameters that are being passed to it by ArcView. The following code in WriteBIP creates a log file for the temporary image:

```
   // Debugging variables

   bool debug = true;
   char log_path[_MAX_PATH];
   FILE *out_log;

   if (debug)
   {
      strcpy(log_path,out_path);
      epos = strrchr(log_path, '.') -
log_path + 1;
      strncpy(log_path + epos, "log",
3);
      success = (! ((out_log =
fopen(log_path,"w")) == NULL));
      if (! success) return false;
      fprintf(out_log,"Path =
'%s'\n",Path);
      fprintf(out_log,"out_path =
'%s'\n",out_path);
      fprintf(out_log,"clip_mbr[0] =
'%f'\n",clip_mbr[0]);
      fprintf(out_log,"clip_mbr[1] =
'%f'\n",clip_mbr[1]);
      fprintf(out_log,"clip_mbr[2] =
'%f'\n",clip_mbr[2]);
      fprintf(out_log,"clip_mbr[3] =
'%f'\n",clip_mbr[3]);
      fprintf(out_log,"out_width =
'%i'\n",out_width);
      fprintf(out_log,"out_height =
'%i'\n",out_height);
   }
```

Other variables may be reported as well.

The Extension

Once the DLL is ready to use or test with ArcView, the extension's AVX file must be created. The online help gives the sample code for the necessary scripts, which were only modified slightly for the AVIdris extension:

Clark Labs

The Clark Labs web site is located at:

http://www.clarklabs.org/

Those interested in finding out more about Idrisi and other Clark Labs products may go directly to:

http://www.clarklabs.org/03prod/03prod.htm

Under the Idrisi product description is a complete, detailed list of commands. Especially check out the wide variety of image processing commands that are available (the table on p. 3 doesn't really do it justice!).

Existing Idrisi users may download the latest service packs and the API at:

http://www.clarklabs.org/13dnlds/13dnlds.htm

Also at that site is a double-to-single precision database conversion utility, suitable for use with shapefiles.

Evaluation copies of Idrisi and CartaLinx are also available at that site.

PLP

PLP Online

http://www.primenet.com/~piersen/PLP

User Name: plpv7n2

Password: m5ox7ru5

IdrImg.MakeExt:

```
theExt =
Extension.Make("$HOME\avidris.avx".AsF
ileName, "Idrisi Image Support", NIL,
NIL, {})
theExt.SetAbout("Extends ArcView to
support Idrisi images (v1.0).")
theExt.SetExtVersion(1.0)
theExt.SetCanUnloadScript(av.FindScrip
t("IdrImg.CanUnload"))
theExt.SetLoadScript(av.FindScript("Id
rImg.Load"))
theExt.SetUnloadScript(av.FindScript("
IdrImg.Unload"))
theExt.Commit
```

IdrImg.Load:

```
ImgDll.RegisterFormat("$AVBIN\avidris.
dll".AsFileName, "img", "bip")
```

IdrImg.CanUnload:

```
av.PurgeObjects
if (ImgDll.GetFormatUseCount("img") =
0) then
  return(TRUE)
end
return(FALSE)
```

IdrImg.Unload:

```
ImgDll.UnregisterFormat("img")
```

Once the scripts are created and compiled, running "IdrImg.MakeExt" will generate the AVX file.

Have fun!

[1]The source code, included in the AVIdris extension, may be downloaded at the following address: www.primenet.com/~piersen/arcview/upload/view/avidris.htm

[2]Indeed, binary byte or integer .IMG files are identical to single-band BIP or BIL files, and could be displayed directly in ArcView if they were renamed and appropriate header/colormap files created.

[3]Thanks to Kenneth McVay for that tip.

PLP

Pulling Items

This routine, to be executed at the ARC/W prompt, is convenient for users who are tired of typing in the internal items for coverage attribute tables in PULLITEM. Not only are the internal items automatically added, but the remaining items in the table are listed out as a reminder!

pullit.r:

```
&routine pullit

&define file -11 &var
&define wksp -12 &var
&if &eq "x%-1" "x" &do
   &delim < >
   &type "Usage:  &r pullit [in_file]"
   &delim [ ]
   &return
&end
&if &ninfo %-1 &do
   &type "%-1 is not a table."
   &return
&end
&value -11 -1
&value -12 WKSP
&openw [wksp]t$it.lis
ITEMS [file] NONE LIST
&closew
&open [wksp]t$it.lis error
&openw [wksp]t$pull.sml
&sv -2 .TRUE.
&while &do
   &read -1 [break]
   &if &eq %-2 .TRUE. &do
      &write "%-1"
   &else
      &write "&type ""%-1"" "
   &end
   &value -3 -1 %<len %-1 - 2 > %<len
%-1>
   &value -4 -1 %<len %-1 - 1 > %<len
%-1>
   &if &eq "%-3" "_ID" &or &eq "%-4"
"_I" &do
      &if &eq %-2 .TRUE. &do
         &sv -2 .FALSE.
         &write "&type ""Internal
items added."" "
         &write "&type ""Remaining
items:"" "
      &end
   &end
&end
&closew
&close
& DEL [wksp]t$it.lis
PULLITEMS [file] [file]
[wksp]t$pull.sml
& DEL [wksp]t$pull.sml
&return

&label error
&type "I/O error"
```

Use COMPSML PULLIT N to create the SML file. Note that the routine creates a temporary SML that is executed within PULLITEM. After the internal items are added and the remaining items typed out, the SML stops execution and it is up to the user to complete the PULLITEM dialog.

```
(C:\0HOME\TEST\PLP)[ARC]&r pullit
Usage:  &r pullit [in_file]
(C:\0HOME\TEST\PLP)[ARC]&r pullit
topo.aat
[PC ARC/INFO 3.5.1 PULLITEM -
01/24/97]
Pulling items from topo.aat to create
topo.aat.

PULLITEM Ver 3.5.1
Copyright (C) 1996 by
  Environmental Systems Research
Institute
  380 New York Street
  Redlands, CA 92373
All Rights Reserved Worldwide.

Internal items added.
Remaining items:
DXF_LAYER
DXF_COLOR
DXF_THICKN
DXF_TYPE
DXF_ELEVAT
DXF_CURVE
ELEV
Enter the 8th item:
```

PLP