Point Line Poly

A Technical Friend for PC ARC/INFO® Users
And Other Desktop GIS Software

Volume **7** Number **4**

Inside:

Icon Key

PC ARC/INFO

ArcView

Idrisi

Farewell to Paper

As of this issue, PLP goes digital! At the PLP OnLine site, two files are offered for viewing and download:

1) a PDF file (Adobe Acrobat 4.0) of the issue, containing full quality graphics and suitable for printing

2) a zip file containing all source code given in the issue, plus occasional bonus code

A link to download Acrobat Reader is offered as well.

The new electronic subscription rate is $10 for 6 issues. An e-mail address is required. Electronic subscribers will receive the Web address and password to each issue.

Special Offer

Current subscribers are invited to convert their print subscription to an electronic one. Every print issue remaining to your current subscription will be converted to 4 electronic issues! All you have to do is e-mail Pierssen Publishing (see the back cover for additional info).

What does this mean for subscribers without e-mail or web access? New or renewed subscriptions for printed copies of PLP will no longer be accepted. If you cannot or do not choose to convert to an electronic subscription, you will continue to receive printed copies of PLP until your subscription expires. Courtesy subscriptions will only continue electronically.

To support secure internet credit card orders, Pierssen Publishing will be starting up its own web server. The Linux box is up and running, but the frame relay connection is not yet in. Hopefully by the next issue the server will be on line. The current web site will also post a redirection when it happens.

Also, past issues (V7N1-V7N3) will be converted to PDF format.

New Directions, New Features

Due to the lack of contributors, PLP will no longer support ArcCAD or AtlasGIS. Until further notice, supported applications will be PC ARC/INFO, ArcView, and Idrisi.

A new series, "Hacker's Corner", begins this issue. While example code is designed for Microsoft Visual C++ programmers, the concepts covered should be applicable to other development platforms. Let's get our hands dirty!

PLP

HACKER'S CORNER

PC ARC/INFO and Automation

Currently, PC A/I only has three commands that facilitate communication with other Windows applications: WIN RUN/RUNW and WIN CB. Because (OLE) automation is not intrinsically supported, in order to communicate with a server application it is necessary to develop a client front-end that PC A/I can invoke.

For this exercise, let's develop an automation server that manages arrays of unsigned integers. Thanks to MFC, creating an array handler class is a fairly simple task. Let's look at the class declaration first:

ahandler.h

```
#include <afxtempl.h>
#include <afxcoll.h>

class AHandler
{
public:
   AHandler();
   UINT Add(UINT size);
   BOOL Remove(UINT arraynum);
   BOOL Set(UINT arraynum, UINT recnum,
UINT value);
   UINT Get(UINT arraynum, UINT
recnum);
   ~AHandler();
protected:
   UINT NewKeyValue;
   CMap<UINT, UINT, CUIntArray*,
CUIntArray*&> theMap;
};
```

A CMap object is used to maintain CUIntArray objects, while a UINT is used to supply unique key values. In this example, the array size must be declared up front. Additional methods/code could facilitate more flexible array handling. The code for the class is as follows:

POINT LINE POLY is published by Pierssen Publishing, 3125 West Wilson Drive, Flagstaff, AZ USA 86001. Electronic subscriptions: 6 issues $10 in US funds. Single Issues $2. E-mail and Web access is required. Send subscriptions and fulfillment questions to *PLP*, piersen@primenet.com. Phone/FAX (520) 774-7905. Or mail to the above address. Submit material and technical questions to The Editor, at one of the above addresses.

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? "Don't play anymore, but they understood sometimes an animal sick in Daddy the noises go thin dies and Fluffy with wave the sound good my mind's been listening you down to drown. I'm going to call the police. They even brought in the boy's body. He'll kill the ocean, thank you." William S. Burroughs, *My Education*

ahandler.cpp

```
#include "ahandler.h"

AHandler::AHandler() {
   NewKeyValue = 0;
   theMap.InitHashTable(257);
   return;
};
UINT AHandler::Add(UINT size) {
   CUIntArray * a = new CUIntArray;
   UINT NewKey = NewKeyValue;
   NewKeyValue++;
   a->SetSize(size);
   theMap.SetAt(NewKey,a);
   return NewKey;
};
BOOL AHandler::Remove(UINT arraynum) {
   CUIntArray * a;
   if (theMap.Lookup(arraynum,a) == 0)
      return FALSE;
   a->RemoveAll();
   delete a;
   theMap.RemoveKey(arraynum);
   return TRUE;
};
BOOL AHandler::Set(UINT arraynum, UINT
recnum, UINT value) {
   CUIntArray * a;
   if (theMap.Lookup(arraynum,a) == 0)
      return FALSE;
   if ((UINT) a->GetSize() <= recnum)
      return FALSE;
   a->SetAt(recnum,(UINT) value);
   return TRUE;
};
UINT AHandler::Get(UINT arraynum, UINT
recnum) {
   CUIntArray * a;
   if (theMap.Lookup(arraynum,a) == 0)
      return 0;
   if ((UINT) a->GetSize() <= recnum)
      return 0;
   return (UINT) a->GetAt(recnum);
};
AHandler::~AHandler() {
   POSITION pos = theMap.
GetStartPosition();
   while(pos != NULL)
   {
      CUIntArray * a;
      UINT KeyNum;
      theMap.GetNextAssoc(pos,KeyNum,
a);
      a->RemoveAll();
      delete a;
   }
   theMap.RemoveAll();
}
```

Note the array destruction code; this helps assure that the object model is leak-proof[1]. A more universal array handler might manipulate CObArray objects, in which case it would be necessary to assure the destruction of each object in the CObArray.

The array handler server application is a hidden CWinApp designed to support multiple users. A UINT is used to track the number of users. By providing methods to add, drop, and get the number of users, a client can detect whether the server is being used by others before closing it. The class declarations are as follows:

ah.h

```
#define VC_EXTRALEAN
#include <afxwin.h>
#include <afxext.h>
#include <afxdisp.h>
#include "ahandler.h"

class CAhApp : public CWinApp
{
public:
   virtual BOOL InitInstance();
};
class CAhWin : public CFrameWnd
{
public:
   CAhWin();
   afx_msg void UserAdd();
   afx_msg void DropUser();
   afx_msg UINT NumUsers();
   afx_msg UINT AddArray(UINT size);
   afx_msg BOOL RemoveArray(UINT
arraynum);
   afx_msg BOOL SetRecord(UINT
arraynum, UINT recnum, UINT value);
   afx_msg UINT GetRecord(UINT
arraynum, UINT recnum);
   afx_msg void Close();
   BOOL RegisterActive();
protected:
   BOOL m_bAutoDelete;
   DWORD m_dwRegister;
   AHandler ah;
```

```
   UINT NumOfUsers;

   virtual ~CAhWin();
   virtual void PostNcDestroy();
   DECLARE_DISPATCH_MAP()
   DECLARE_DYNCREATE(CAhWin)
   DECLARE_OLECREATE(CAhWin)
};

#ifndef IMPLEMENT_OLECREATE2
#define IMPLEMENT_OLECREATE2
(class_name, external_name, l, w1, w2,
b1, b2, b3, b4, b5, b6, b7, b8) \
   AFX_DATADEF COleObjectFactory
class_name::factory(class_name::guid, \
   RUNTIME_CLASS(class_name), TRUE, _T
(external_name)); \
   const AFX_DATADEF GUID class_name::
guid = \
   { l, w1, w2, { b1, b2, b3, b4, b5,
b6, b7, b8 } };
#endif // IMPLEMENT_OLECREATE2
```

The server application is also designed to exist as only one instance. Additional attempts to launch it are not honored:

ah.cpp

```
#include "ah.h"

CAhApp theApp;

BOOL CAhApp::InitInstance()
{
   if (FindWindow(NULL,"Array
Handler") != NULL)
      return FALSE;
   if (!AfxOleInit())
   {
      TRACE("OLE Initialization
failed");
      return FALSE;
   }
   if (RunEmbedded() || RunAutomated())
   {
      COleTemplateServer::RegisterAll
();
      return TRUE;
   }
   COleObjectFactory::UpdateRegistryAll
();
   CAhWin * ahw = new CAhWin();
   m_nCmdShow = SW_HIDE;
   ahw->ShowWindow(m_nCmdShow);
   ahw->UpdateWindow();
   ahw->RegisterActive();
   m_pMainWnd = ahw;
   return TRUE;
}
```

The application's window acts as the automation server. It keeps track of the number of users and acts as a front end for the AHandler object. We won't go into the code here—you can view it in the PLP OnLine code pack. (See file "ahwnd.cpp"; the IMPLEMENT_OLECREATE2 parameters were copied from a dry run of the Visual C++ MFC app wizard.)

Implementing a Front End Client

Now that we have a server, we need to develop a front end client that PC A/I can use to communicate with it. The client doesn't need a window, but it does need a dispatch driver. The client header file is as follows:

arrayf.h

```
#define VC_EXTRALEAN

#include <afxwin.h>
#include <afxext.h>
#include <afxdisp.h>

class CRemoteAhWnd : public
COleDispatchDriver
{
public:
   void UserAdd();
   void DropUser();
   UINT NumUsers();
   UINT AddArray(UINT size);
   BOOL RemoveArray(UINT arraynum);
   BOOL SetRecord(UINT arraynum, UINT
recnum, UINT value);
   UINT GetRecord(UINT arraynum, UINT
recnum);
   void Close();
};
class CArrayfApp : public CWinApp
{
public:
   virtual BOOL InitInstance();
   CRemoteAhWnd ah;
private:
```

```
   void SetFromFile(UINT arraynum,
LPCTSTR fName);
   void SetFromWinSel(UINT arraynum,
LPCTSTR fName);
   void PostMsg(LPCTSTR Msg);
   void PostResult(UINT Result);
   void PostBResult(BOOL BResult);
};
```

The CRemoteAhWnd object handles the dispatches (see file "ahtype.cpp" in the code pack). The numeric assignments for the InvokeHelper statements correspond to the order of the dispatch map statements in the server application[2].

The real meat is in the CWinApp code. First, let's look at the "InitInstance" method:

arrayf.cpp

```
#include "arrayf.h"

CArrayfApp ArrayfApp;

BOOL CArrayfApp::InitInstance()
{
   char *tok;
   char cmd[256];
   CLSID clsid;
   COleException e;
   HRESULT hr;
   LPDISPATCH lpDispatch;
   LPUNKNOWN lpUnk;
   UINT size, arraynum, recnum, value;

   if (strlen(m_lpCmdLine) == 0)
   {
      PostMsg("USAGE: ARRAYF [U/D/N/A/
R/S/G/F/W/C] {options}");
      return TRUE;
   }
   if (!AfxOleInit())
   {
      PostMsg("ERROR: OLE
Initialization failed");
      return FALSE;
   }
   if (CLSIDFromProgID(OLESTR("ah.
Server"), &clsid) != S_OK)
   {
      PostMsg("ERROR: Could not find
CLSID for AH");
      return FALSE;
   }
   // We don't want to create a
dispatch in this app
   if (GetActiveObject(clsid, NULL,
&lpUnk) != S_OK)
   {
      PostMsg("ERROR: Could not find
active AH");
      return FALSE;
   }
   hr = lpUnk->QueryInterface
(IID_IDispatch,
      (LPVOID*)&lpDispatch);
   lpUnk->Release();
   if (hr == NOERROR)
      ah.AttachDispatch(lpDispatch,
TRUE);
   if (ah.m_lpDispatch == NULL)
   {
      PostMsg("ERROR: Could not attach
to AH");
      return FALSE;
   }
   strcpy(cmd,m_lpCmdLine);
   tok = strtok(cmd," ");
   switch(tolower(tok[0]))
   {
      case 'u':
         ah.UserAdd();
         break;
      case 'd':
         ah.DropUser();
         break;
      case 'n':
         PostResult(ah.NumUsers());
         break;
      case 'a':
         tok = strtok(NULL," ");
         size = atoi(tok);
         PostResult(ah.AddArray(size));
         break;
      case 'r':
         tok = strtok(NULL," ");
         arraynum = atoi(tok);
         PostBResult(ah.RemoveArray
(arraynum));
         break;
      case 's':
         tok = strtok(NULL," ");
         arraynum = atoi(tok);
         tok = strtok(NULL," ");
    recnum = atoi(tok);
         tok = strtok(NULL," ");
         value = atoi(tok);
```

```
         PostBResult(ah.SetRecord
(arraynum,recnum,value));
         break;
      case 'g':
         tok = strtok(NULL," ");
         arraynum = atoi(tok);
         tok = strtok(NULL," ");
         recnum = atoi(tok);
         PostResult(ah.GetRecord
(arraynum,recnum));
         break;
      case 'f':
         tok = strtok(NULL," ");
         arraynum = atoi(tok);
         tok = strtok(NULL,"\"");
         SetFromFile(arraynum,tok);
         break;
      case 'w':
         tok = strtok(NULL," ");
         arraynum = atoi(tok);
         tok = strtok(NULL,"\"");
         SetFromWinSel(arraynum,tok);
         break;
      case 'c':
         ah.Close();
         break;
      default:
         PostMsg("ERROR: Invalid
Option");
         return FALSE;
   }
   return TRUE;
}
```

Note that the client will not actually create an instance of the server; rather, the server must be launched separately. To understand the various options, let's cover a typical sequence of commands:

arrayf U = add a user
arrayf A [n] = add an array of size [n] (returns a key value [a])
arrayf S [a] [r] [v] = set element [r] of array [a] equal to [v]
arrayf G [a] [r] = return value of element [r] of array [a]
arrayf R [a] = remove array [a]
arrayf D = drop a user
arrayf N = return the current number of users
arrayf C = close the server application

There are two additional commands:

arrayf F [a] [f] = populate array [a] from file [f]
arrayf W [a] [w] = populate array [a] from file [w]

File [f] is an ASCII file with each line containing an integer string. The "SetFromFile" method reads the file and populates the array:

```
void CArrayfApp::SetFromFile(UINT
arraynum, LPCTSTR fName)
{
   FILE *infile;
   char input[81];
   UINT recnum = 0;
   UINT value;
   BOOL result;

   if ((infile = fopen(fName,"r")) ==
NULL)
   {
      PostMsg("ERROR: Could not open
file");
      return;
   }
   while (fgets(input, 80, infile) !=
NULL)
   {
      value = atoi(input);
      result = ah.SetRecord(arraynum,
recnum,value);
      recnum++;
   }
   fclose(infile);
   PostBResult(TRUE);
   return;
}
```

File [w] is a WIN SEL file as generated by ARCPLOTW. Before looking at the "SetFromWinSel" method, let's discuss the structure of a WIN SEL file. It's easiest to think of one as a set of bits:

```
Bit: 0       7 8      15 16     23
Set: 00101000 00011111 11111111
```

In the above file (binary values are backwards for illustrative purposes), there are a total of 11 records. Records 3 and 5 (bit 0 = record 1) are selected. The remaining 5 bits of byte 2 are padding, and byte 3 is the closing byte.

Okay, now let's look at the actual code:

```
void CArrayfApp::SetFromWinSel(UINT
arraynum, LPCTSTR fName)
{
   FILE *infile;
   BYTE input;
   UINT recnum = 0;
   UINT value = 1;
   BOOL result;

   if ((infile = fopen(fName,"rb")) ==
NULL)
   {
      PostMsg("ERROR: Could not open
file");
      return;
   }
   while (fread(&input, sizeof(input),
1, infile) != 0)
   {
      for(int i = 0;i < 8;i++)
      {
         if (input & 1)
         {
            result = ah.SetRecord
(arraynum,recnum,value);
            recnum++;
         }
         input = input >> 1;
         value++;
      }
   }
   fclose(infile);
   PostBResult(TRUE);
   return;
}
```

Note that, for simplicity, the routine will attempt to set values beyond the actual end of the array. It doesn't really matter, because the AHandler object is smart enough to ignore such attempts[3].

Finally, results are communicated via the clipboard:

```
void CArrayfApp::PostMsg(LPCTSTR Msg)
{
   HGLOBAL hg;
   int i;
   LPTSTR lp;

   if (!OpenClipboard(NULL))
   {
      THROW("Cannot open the
Clipboard");
      return;
   }
   EmptyClipboard();
   i = strlen(Msg);
   hg = GlobalAlloc(GMEM_DDESHARE,
       (i + 1) * sizeof(TCHAR));
   if (hg == NULL)
   {
      THROW("Unable to allocate
handle");
      CloseClipboard();
      return;
   }
   lp = (LPTSTR) GlobalLock(hg);
   memcpy(lp, Msg, i * sizeof(TCHAR));
   lp[i] = (TCHAR) 0;
   GlobalUnlock(hg);
   SetClipboardData(CF_TEXT,hg);
   CloseClipboard();
   return;
}
void CArrayfApp::PostResult(UINT
Result)
{
   char r[20];

   _itoa(Result,r,10);
   PostMsg(r);
   return;
}
void CArrayfApp::PostBResult(BOOL
BResult)
{
   if (BResult)
      PostMsg(".TRUE.");
   else
      PostMsg(".FALSE.");
   return;
}
```

See p. 8 for examples using both AH and ARRAYF with PC A/I.

[1]For example memory leak testing code, see "leaktest.cpp" in the PLP OnLine code pack.
[2]Normally this is only a concern if, as in this example, the Visual C++ MFC app and class wizards are bypassed in developing the applications.
[3]To be frank, I'm not entirely happy with the bit testing algorithm, and can't help but feel that a better approach is possible. If anybody knows one (Joe, are you reading this?), I'd be glad to hear about it and will report it in a future issue.

PLP

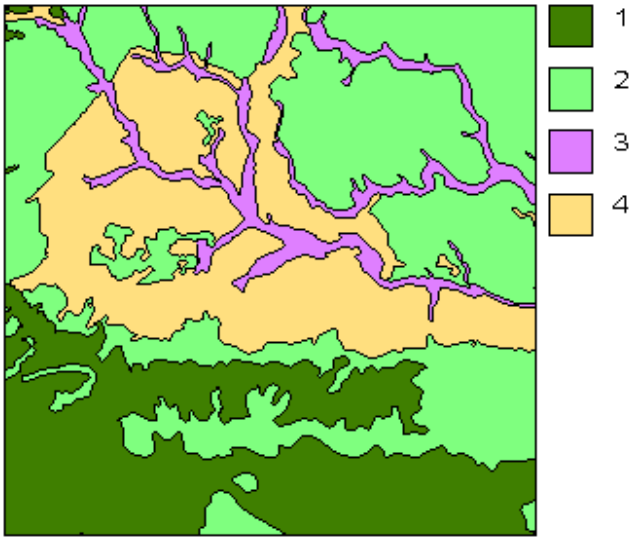
Faking Cursors

Cursor processing allows the analysis of a selected set of objects, one record at a time. This is particularly useful for querying and/or setting attributes on a case-by-case basis.

In the simplest case in TABLES or ARCPLOTW—processing all records—a loop may be set up to reselect each record for processing (see "ap_all.r" in the PLP OnLine code pack). The situation, however, is considerably more complex when dealing with a particular selection set. In such a case the cursor routine must find a way to store and manipulate the selection set[1]. Also, because in ARCEDITW record numbers change upon editing, a mechanism should exist for updating the array.

The array handler applications developed on p. 2 provide a means to store record numbers in memory and retrieve or modify them as needed. The disadvantage of this approach is that communication is only possible via WIN RUN/RUNW and WIN CB. Thus this approach cannot be used in TABLES[2].

EXAMPLE ONE: Generating Boundary Attributes in ARCPLOTW



Polygon coverage VEG has four vegetation types:

```
TYPE TYPE_TXT

1     Conifer
2     Pinyon-Juniper
3     Sagebrush
4     Grassland
```

Arcs comprising vegetation boundaries are to be classified as follows (lookup table LUT.VEGB):

```
TYPE TYPE_TXT

1     Conifer/PJ
2     Conifer/Sagebrush
3     Conifer/Grassland
4     PJ/Sagebrush
5     PJ/Grassland
6     Sagebrush/Grassland
```

First, TYPE and TYPE_TXT are added to VEG.AAT:

```
ADDITEM VEG.AAT VEG.AAT TYPE 1 1 I
ADDITEM VEG.AAT VEG.AAT TYPE_TXT 20 20 C
```

Routine "ap_curs" is a "generic" routine for cycling through a selection set in ARCPLOTW:

```
&routine ap_curs

&define cover -1 &var
&define feature -2 &var
&define numrec -11 &var
&define arraynum -12 &var
&define i -13 &var
&define id -14 &var
&define wksp -18 &var
&define temp -19 &var
&if &eq "x[cover]" "x" &do
   &delim < >
   &type "Usage:  &r ap_curs [cover]
[feature]"
   &return
   &delim [ ]
&end
&rem **** initialize cursor
&value [wksp] WKSP
&value [temp] ARC
WIN SEL W [cover] [feature] [wksp]t
```

```
$sel.lis
SHOW RESELECT [numrec] 0
&if &eq [numrec] 0 &do
   &type "ERROR: No records in
selection set"
   & DEL [wksp]t$sel.lis
   &return
&end
WIN PATH [temp]\APPS
&type "Starting server..."
WIN RUN ah
&type "Initializing cursor..."
WIN RUNW arrayf U
WIN RUNW arrayf A [numrec]
WIN CB R
&value [arraynum] 1
WIN RUNW arrayf W [arraynum] [wksp]t
$sel.lis
&rem **** loop through record set
&sv [i] 0
&while &rn [i] 0 %<[numrec] - 1> &do
   WIN RUNW arrayf G [arraynum] [i]
   WIN CB R
   &value [id] 1
   RES [cover] [feature] $RECNO IN
{[id]}
   &rem **** manipulate record
   &type "Processing record [id]..."
   &run ap_doit
   ASEL [cover] [feature]
   &inc [i]
&end
&rem **** remove cursor
&type "Removing cursor..."
WIN RUNW arrayf R [arraynum]
WIN RUNW arrayf D
WIN RUNW arrayf N
WIN CB R
&value [i] 1
&if &eq [i] 0 &do
   &type "Closing server..."
   WIN RUNW arrayf C
&end
&rem **** restore selection set
WIN SEL R [cover] [feature] [wksp]t
$sel.lis
& DEL [wksp]t$sel.lis
&return
```

Note that the routine expects the array handler applications to reside in the %ARC%\APPS directory. WIN SEL is used to save and restore the selection set. WIN RUN is used to start the server application, WIN RUNW is used to execute the client front end, and WIN CB is used to get the results. Routine "ap_doit" performs the record manipulation. Before we look at that, however, let's look routine "ap_run":

```
&routine ap_run

ADDITEM VEG.AAT VEG.AAT T_VEG 11 11 I
ADDITEM VEG.AAT VEG.AAT T_LT 1 1 I
ADDITEM VEG.AAT VEG.AAT T_RT 1 1 I
ARCPLOTW ap_proc
DROPITEM VEG.AAT VEG.AAT T_VEG
DROPITEM VEG.AAT VEG.AAT T_LT
DROPITEM VEG.AAT VEG.AAT T_RT
&return
```

This routine is run from the ARCW prompt; it adds some temporary items, launches ARCPLOTW, and then drops the items.

```
&routine ap_proc

CALC VEG ARC TYPE = 0
MOVEITEM VEG ARC ' ' TO TYPE_TEXT
CALC VEG ARC T_VEG = VEG_
CALC VEG ARC VEG_ = LPOLY_
JOIN VEG.AAT VEG.PAT VEG_
CALC VEG ARC T_LT = #TYPE
CALC VEG ARC VEG_ = RPOLY_
CALC VEG ARC T_RT = #TYPE
JOIN OFF
RES VEG ARC T_LT <> 0 AND T_RT <> 0
&r ap_curs VEG ARC
JOIN VEG.AAT LUT.VEGB TYPE
MOVEITEM VEG ARC #TYPE_TEXT TO
TYPE_TEXT
JOIN OFF
ASEL VEG ARC
CALC VEG ARC VEG_ = T_VEG
QUIT
&return
```

This routine saves VEG_ to T_VEG so that it can be freed up to join to VEG.PAT. T_LT and T_RT are populated with TYPE from the left- and right-hand polygons, and arcs not corresponding to the edge of the project area are reselected. Then "ap_curs" is invoked for the selection set. Finally, TYPE_TEXT is calculated from the lookup table.

Now let's look at "ap_doit":

```
&routine ap_doit

CALC VEG ARC -1 = T_LT * T_RT
&if &eq %-1 2 &do
   CALC VEG ARC TYPE = 1
&elseif &eq %-1 3 &do
   CALC VEG ARC TYPE = 2
&elseif &eq %-1 4 &do
   CALC VEG ARC TYPE = 3
&elseif &eq %-1 6 &do
   CALC VEG ARC TYPE = 4
&elseif &eq %-1 8 &do
   CALC VEG ARC TYPE = 5
&elseif &eq %-1 12 &do
   CALC VEG ARC TYPE = 6
&else
   &echo &debug
&end
&return
```

There are various ways to approach this, but because multiplying two different TYPE values results in a unique number, that is used for the check.

This actually is not the best possible example, as it could have been handled using RESELECT statements. Nonetheless, the value of a cursor is evident when features need to be manipulated in a way that RESELECT and CALC/MOVEITEM just can't handle.

EXAMPLE TWO: Repositioning Points in ARCEDITW

A point coverage STATIONS represents wildlife monitoring stations along certain roads in the project area (see "Faking Dynamic Segmentation" in **V6N2**). The lead agency insists that the monitoring stations be offset from the roads by 200 meters in random directions. In this example, 0 degrees is north, 90 degrees is east, 180 degrees is south, and 270 degrees is west.

First, the coverage is copied to STATION2. Routine "ae_proc" is run within ARCEDITW:

```
&routine ae_proc

EDITC STATION2
EDITF LABEL
SHOW COORDINATE -18
COO KEY
SEL ALL
&cv -2 1 ran
&r ae_curs RESTORE
&rem UNSEL ALL
COO %-18
&return
```

Note the line seeding the random number generator. Routine "ae_curs" has some significant differences from "ap_curs":

```
&routine ae_curs

&define restore -1 &var
&define numrec -11 &var
&define arraynum -12 &var
&define i -13 &var
&define id -14 &var
&define wksp -18 &var
&define temp -19 &var
&if &eq "x[restore]" "x" &do
   &delim < >
   &type "Usage:  &r ae_curs [RESTORE/
NORESTORE]"
   &return
   &delim [ ]
&end
&rem **** initialize cursor
&value [wksp] WKSP
&value [temp] ARC
SHOW NUMBER SELECT [numrec]
&if &eq [numrec] 0 &do
   &type "ERROR: No records in
selection set"
   &return
&end
WIN PATH [temp]\APPS
&type "Starting server..."
WIN RUN ah
&type "Initializing cursor..."
WIN RUNW arrayf U
WIN RUNW arrayf A [numrec]
WIN CB R
&value [arraynum] 1
&sv [i] 1
&openw [wksp]t$sel.lis
&while &rn [i] 1 [numrec] &do
   SHOW SELECT %[i] [temp]
   &write [temp]
   &inc [i]
&end
&closew
WIN RUNW arrayf F [arraynum] [wksp]t
$sel.lis
```

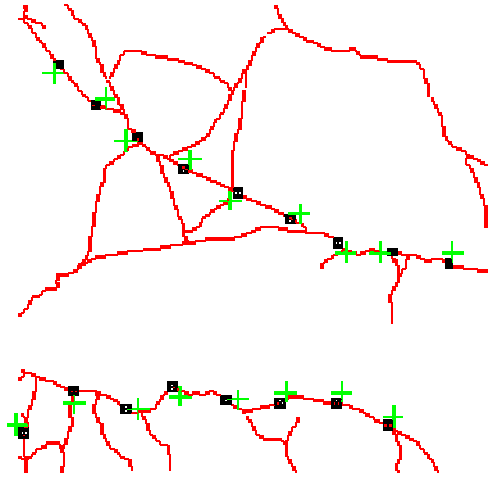
```
& DEL [wksp]t$sel.lis
&rem **** loop through record set
&sv [i] 0
&while &rn [i] 0 %<[numrec] - 1> &do
   WIN RUNW arrayf G [arraynum] [i]
   WIN CB R
   &value [id] 1
   SEL $RECNO IN {[id]}
   &rem **** manipulate record
   &type "Processing record [id]..."
   &run ae_doit
   &rem **** update record number
   SHOW SELECT 1 [id]
   WIN RUNW arrayf S [arraynum] [i]
[id]
   &inc [i]
&end
&if &eq RESTORE [restore] &do
   &rem **** restore selection set
   &sv [i] 0
   &while &rn [i] 0 %<[numrec] - 1> &do
      WIN RUNW arrayf G [arraynum] [i]
      WIN CB R
      &value [id] 1
      ASEL $RECNO IN {[id]}
      &inc [i]
   &end
&end
&rem **** remove cursor
&type "Removing cursor..."
WIN RUNW arrayf R [arraynum]
WIN RUNW arrayf D
WIN RUNW arrayf N
WIN CB R
&value [i] 1
&if &eq [i] 0 &do
   &type "Closing server..."
   WIN RUNW arrayf C
&end
&return
```

Because WIN SEL is currently unavailable in ARCEDITW, an ASCII file of record numbers is written and passed on to the client front end. Note also that the new record number is sent back to the array handler. That's not important in this particular application, but if you want to go both forward and backward it's necessary (see the forms application in the next issue). Without WIN SEL, restoring the selection set is slow and clunky, which is why there's an option not to restore it.

```
&routine ae_doit

&cv -1 200
&cv -2 360 * ( 0 ran )
&cv -3 %-1 * ( ( 90 - %-2 ) cos )
&cv -4 %-1 * ( ( 90 - %-2 ) sin )
MOVE
1 0 0
1 %-3 %-4
&return
```

Here's where the actual move is done. Very simple.



In the above figure, boxes represent the original station positions and crosses the new positions.

One final note: a Pentium-based PC running Windows NT really helps speed things up!

Next Issue: Cursors and Forms

[1]The example currently at http://www.primenet.com/~piersen/PC/arctips/advanced/cursor.htm stores record numbers within file-based global SML variables, but this places a limitation on the number of records that can be stored.
[2]This topic will be revisited after the release of PC A/I 4.0.

PLP

Handling Comma Separated Data

The ADD FROM command in TABLES expects comma separated values (CSV), with string data enclosed in single quotes. When designing an automated data input routine for TABLES, it may be useful to analyze a line of input to determine what items to create for the table. Routine "csv" will count the number of tokens in a line of CSV or will extract a specified token:

```
&routine csv

&define i -9 &var
&define tok -10 &var
&define opt -11 &var
&define in -12 &var
&define toknum -13 &var
&define num -14 &var
&define len -15 &var
&define chr -16 &var
&define quote -17 &var
&if &eq "x%-1" "x" &do
   &delim < >
   &type "Usage:  &r CSV [COUNT/
EXTRACT] [string] {token_number}"
   &delim [ ]
   &return
&end
&extract [opt] -1 1
&if &eq "x[opt]" "x#" &do
   &sv [opt] COUNT
&end
&if &ne "COUNT" "[opt]" &and &ne
"EXTRACT" "[opt]" &do
   &type "Option must be COUNT or
EXTRACT."
   &return
&end
&value [in] -2
&if &eq "x[in]" "x" &do
   &type "Input string is blank."
   &return
&end
&value [toknum] -3
&if &eq "EXTRACT" "[opt]" &and &eq "x
[toknum]" "x" &do
   &type "Token number not specified."
   &return
&end
&sv [i] 1
&sv [tok] 1
&sv [num] 1
&length [len] "[in]"
&sv [quote] .FALSE.
&while &rn [i] 1 [len] &do
   &value [chr] [in] [i] [i]
   &if &eq "'" "[chr]" &do
      &if &eq [quote] .TRUE. &do
         &value [chr] [in] %<[i] + 1> %
<[i] + 1>
         &if &eq "'" "[chr]" &do
            &inc [i]
         &else
            &sv [quote] .FALSE.
         &end
      &else
         &sv [quote] .TRUE.
      &end
   &elseif &eq "," "[chr]" &do
      &if &eq [quote] .FALSE. &do
         &if &eq "EXTRACT" "[opt]" &do
            &if &eq [toknum] [num] &do
               &value [chr] [in] [tok]
%<[i] - 1>
               &return "[chr]"
            &else
               &cv [tok] [i] + 1
            &end
         &end
         &inc [num]
      &end
   &end
   &inc [i]
&end
&if &eq EXTRACT [opt] &do
   &if &eq [toknum] [num] &do
      &value [chr] [in] [tok] [len]
      &return "[chr]"
   &else
      &type "Bad token number."
      &return
   &end
&end
&return [num]
```

Use COMPSML csv N to create "csv.sml". See "csv_test.r" in the PLP OnLine code pack for an example of its use.

PLP

Using Win32 DLLs in Avenue

If you go to ArcView's online help contents and navigate to "Customizing and programming ArcView with Avenue", "Creating an ArcView Application", "Integrating ArcView with other applications", "Dynamic link libraries (DLL)", "Using Microsoft DLLs" , you'll find a discussion of calling WIN32 API functions using ArcView's DLL and DLLProc Classes.

Here are three potentially useful examples: "mkdir.ave" makes a unique temporary directory under the $TEMP directory, "rmdir.ave" removes an existing (empty) directory, and "rename.ave" renames a file.

mkdir.ave:

```
thePrefix = Self.Get(0)
dllName = FileName.
FindInSystemSearchPath("kernel32.dll")
u32DLL = DLL.Make(dllName)
MkDir = DLLProc.Make
(u32DLL,"CreateDirectoryA",
#DLLPROC_TYPE_INT32,
   {#DLLPROC_TYPE_STR,
#DLLPROC_TYPE_VOID})
strNewDir = FileName.GetTmpDir.MakeTmp
(thePrefix,"").GetFullName
result = MkDir.Call({strNewDir,nil})
if (result = 0) then
   strNewDir = "ERROR"
end
return strNewDir
```

rmdir.ave:

```
strRmDir = SELF.Get(0)
success = true
dllName = FileName.
FindInSystemSearchPath("kernel32.dll")
u32DLL = DLL.Make(dllName)
MkDir = DLLProc.Make
(u32DLL,"RemoveDirectoryA",
#DLLPROC_TYPE_INT32,
   {#DLLPROC_TYPE_STR})
result = MkDir.Call({strRmDir})
if (result = 0) then
   success = false
end
return success
```

rename.ave:

```
strOldFN = SELF.Get(0)
strNewFN = SELF.Get(1)
success = false
dllName = FileName.
FindInSystemSearchPath("winmm.dll")
u32DLL = DLL.Make(dllName)
Rename = DLLProc.Make
(u32DLL,"mmioRenameA",
#DLLPROC_TYPE_INT32,
   {#DLLPROC_TYPE_STR,
#DLLPROC_TYPE_STR, #DLLPROC_TYPE_VOID,
#DLLPROC_TYPE_INT32})
result = Rename.Call({strOldFN,
strNewFN, nil, 0})
if (result = 0) then
   success = true
end
return success
```

(One other example in the PLP OnLine code pack, "getsysd.ave", retrieves the Windows System directory.)

Remember that Avenue cannot handle all types of data structures. For example, the second argument of "CreateDirectoryA" is a SECURITY_ATTRIBUTES structure. Fortunately, for this particular function the second argument is optional and NULL may be passed instead.

For an in-depth discussion of the Win32 API (a.k.a. "Platform SDK"), you can purchase the MSDN Library CD-ROM (free with Visual Basic or C++ pro editions), or one of several books available on the market. A list of the Win32 API declarations for Visual Basic may be downloaded at:

http://www.microsoft.com/officedev/o-free.htm

Finally, "rmfiles.ave" in the PLP Online code pack doesn't use DLL calls. It's just a reminder that you can use ReadFiles to find and remove files in a directory.

PLP

Sorting Objects

This is one of my all-time favorite Avenue scripts. An implementation of the quick sort algorithm[1], QSort can sort a list of ANY objects for which a comparison routine may be devised. There are two arguments: the list of objects to be sorted and the name of the comparison script to be used. The comparison script must be executable as follows:

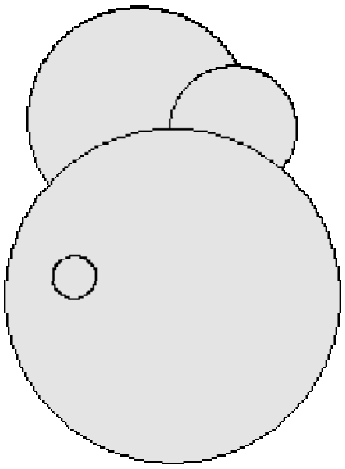
test = av.Run(ScriptName,{obj1,obj2})

and must return a number such that:

test < 0 when obj1 < obj2
test = 0 when obj1 = obj2
test > 0 when obj1 > obj2

QSort.ave:

```
theList = SELF.Get(0)
ScriptName = SELF.Get(1)
n = theList.Count
if (n < 2) then return theList end
l = 0
r = n - 1
theStack = Stack.Make
theStack.Push({l,r})
av.ShowMsg("Sorting...")
e = 0
nume = n
while (theStack.Depth > 0)
   e = e + 1
   av.SetStatus(e / nume * 100)
   arg = theStack.Pop
   l = arg.Get(0)
   r = arg.Get(1)
   i = l
   j = r
   pivot = theList.Get((l + r) / 2)
   NotDone = True
   while (NotDone)
      while (av.Run(Scriptname,
{theList.Get(i),pivot}) < 0)
         i = i + 1
      end
      while (av.Run(Scriptname,
{theList.Get(j),pivot}) > 0)
         j = j - 1
      end
      if (i <= j) then
         temp = theList.Get(i)
         theList.Set(i, theList.Get(j))
         theList.Set(j, temp)
         i = i + 1
         j = j - 1
      end
      NotDone = (i <= j)
   end
   if (i < r) then
      theStack.Push({i,r})
   end
   if (l < j) then
      theStack.Push({l,j})
   end
end
av.ClearMsg
av.ClearStatus
return theList
```



For example, given a shapefile with overlapping polygons, some of which are hidden:

Let's sort the records so that all polygons will be visible.

SortPoly.ave:

```
theTitle = "Sort Polygons"
theView = av.GetActiveDoc
theTheme = theView.GetActiveThemes.Get
(0)
inFTab = theTheme.GetFTab
' Specify the output shapefile...
fnDefault = FileName.GetCWD.MakeTmp
("shape","shp")
fnOutput  = FileDialog.Put
( fnDefault,"*.shp","Output Shape
File" )
if (fnOutput = nil) then exit end
' Use selected shapes if there are any,
otherwise iterate
' through the entire FTab...
if (inFTab.GetSelection.Count > 0) then
```

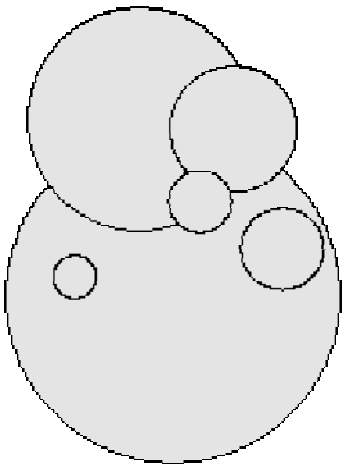
```
  colToProcess = inFTab.GetSelection
  nRecs = colToProcess.Count
else
  colToProcess = inFTab
  nRecs = colToProcess.GetNumRecords
end
'**** set global variable for the
comparison routine
_theFTab = inFTab
'**** generate list of records
rlist = List.Make
for each r in colToProcess
   rlist.Add(r.Clone)
end
'**** sort it
slist = av.Run("qsort",
{rlist,"CompPoly"})
_theFTab = nil
'**** now write out the new records
fnOutput.SetExtension("shp")
outFTab = FTab.MakeNew( fnOutput,
POLYGON )
inFields = inFTab.GetFields
newFields = List.Make
for each f in inFields
   if (f.GetName <> "shape") then
      newFields.Add(f.Clone)
   end
end
outFTab.AddFields(newfields)
nCount = 0
nRecAdded = 0
mpCount = 0
inSF  = inFTab.FindField("shape")
outSF = outFTab.FindField("shape")
for each r in slist
   nRecNew = outFTab.AddRecord
   for each inF in inFTab.GetFields
      fName = inF.GetName
      outF = outFTab.FindField(fName)
      val = inFTab.ReturnValue(inF,r)
      outFTab.SetValue(outF,nRecNew,
val)
   end
   nRecAdded = nRecAdded + 1
   nCount = nCount + 1
   av.SetStatus((nCount / nRecs) * 100)
end
av.ClearStatus
av.ClearMsg
if (MsgBox.YesNo("Add shapefile as
theme to a view?",
  theTitle, true).Not) then
  exit
end
thmNew = FTheme.Make(outFTab)
theView.AddTheme( thmNew )
```

Note the global variable "_theFTab", which is used in turn by the comparison routine:

CompPoly.ave:

```
a = SELF.Get(0)
b = SELF.Get(1)
sf = _theFTab.FindField("shape")
p1 = _theFTab.ReturnValue(sf,a)
p2 = _theFTab.ReturnValue(sf,b)
if (p1.Contains(p2)) then return -1 end
if (p2.Contains(p1)) then return 1 end
if (p1.ReturnArea > p2.ReturnArea) then
return -1 end
if (p2.ReturnArea > p1.ReturnArea) then
return 1 end
return 0
```

In the comparison routine, polygons to be drawn in the back are "lesser" than polygons to be drawn in the front. The result:



[1]This version is based on the algorithm at Larry Ogren's web site: http://www.azstarnet.com/~logren/qsort.htm Of several versions tested (including one from Microsoft's web site!), it was by far the best.

PLP

Rectifying an Image

When rectifying an image using RESAMPLE, finding the minimum and maximum X-Y values for the output image may seem a daunting task. One quick way to find them is to create a vector file representing the boundary of the image and then resampling it. For example, an image imported from a Windows bitmap (.BMP) file will be given X-Y values ranging from 0 to 1. A vector file representing the boundary would be as follows:

```
I01.VEC:

1 5
0 0
0 1
1 1
1 0
0 0
0 0

I01.DOC:

file title  : Frame file
id type     : integer
file type   : ascii
object type : line
ref. system : plane
ref. units  : meters
unit dist.  : 1.0000000
min. X      : 0
max. X      : 1
min. Y      : 0
max. Y      : 1
pos'n error : unknown
resolution  : unknown
```

When you resample the vector file, give arbitrary min and max values such as -999999 and 999999. Given the following correspondence file:

```
4
.0326804   .6750771     -11271   27746
.036572    .3038769     -11288   13876
.861845    .6800654      11271   27746
.8675624   .3096441      11288   13876
```

the resulting polygon will be:

```
1.0000000 5
-12392.7832031 2512.3005371
-12040.7099609 39913.6445313
15133.9785156 39671.3164063
14781.9052734 2269.9726563
-12392.7832031 2512.3005371
0 0
```

Thus if the desired resolution is 10, MinX = -12400, MaxX = 15140, MinY = 2260, MaxY = 39920, Columns = 2754, and Rows = 3766.

PLP

To convert your subscription from print to electronic:

E-mail Pierssen Publishing: piersen@primenet.com
Subject: Convert PLP Subscription

The number of issues remaining to your subscription will automatically be multiplied by 4. In the future, you will be notified by e-mail when a new issue of **PLP** is available, or when your subscription is expiring.

You will also be notified when Pierssen Publishing's new web site is on line.

PLP OnLine

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

User Name: plpv7n4
Password: 4oku9u6g