GISDK Basics

presented to

Caltrans

presented by

Sean McAtee,
Paul Ricotta

CAMBRIDGE SYSTEMATICS

May 9, 2016

Think Forward

GISDK ~~Programming~~ Scripting Language

- C-like?
  - Do … end instead of { … }
    - Macro … EndMacro
    - Dbox … EndDbox
    - Etc…
  - Variable type is handled automatically
    - Single-element variables do not need to be declared
  - Code can must be contained in
    - Macros (similar to functions)
    - Dialog Boxes
    - Objects
      - Can have methods and properties
  - Must be complied prior to running

Editing Resource code

- Syntax Highlighting
  - » Function Names
  - » Item Start/End
  - » Strings
  - » Numbers
  - » Comments
- Various programs available
  - » Notepad++ (Free, open source)
  - » Ultra Edit (not free)
  - » Various others…
- Basic, no highlighting: notepad.exe

Language Elements

- Variables
  - » Very flexible (dynamic typing)
  - » Integer, string, real
  - » Many more…

// indicates end of line comments

- Arrays

```
MyArray = {2, 4, 6}    //MyArray[1] = 2
MyArray[5] = 1  //This will result in an error (out of bounds)
MyArray[0] = 1  //This will always result in an error (GISDK is one based)
```

  - » Each element can have a different type

```
MyArray = {1, "Two", 3}
```

  - » Arrays can be nested

```
MyArray = {{1, 2, 3}, {"a", "b", "c"}}      //MyArray[2][1] = "a"
```

Language Elements

- Arrays
  - Arrays may need to be defined:

```
MyArray = {2, 4, 5}  //Creates a new array
YourArray[1] = 2  //Does not work unless YourArray is already an array
Dim YourArray[10]  //Creates an empty 10-element array
Dim YourArray[10, 10] // Creates a new 10x10 array
```

```
MyArray = {2, 4, 6}
x = MyArray.Length //x = 3
x.Length = 5  //This will result in an error
```

```
MyArray = {{2, 4, 6}, {8, 10, 12}}
RefArray = MyArray[1]
RefArray[1] = "Two"
x = MyArray[1][2]  //x = "Two"
```

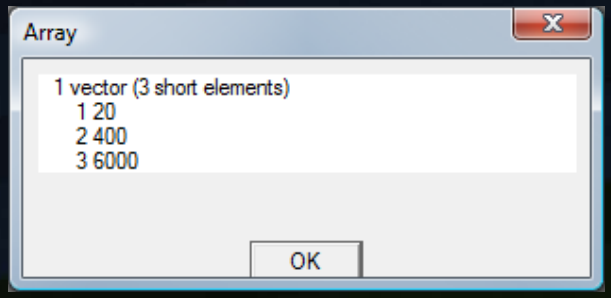
This can get you in trouble!

```
MyArray = {{2, 4, 6}, {8, 10, 12}}
RefArray = CopyArray(MyArray[1])
RefArray[1] = "Two"
x = MyArray[1][2]  //x = 2
```

Language Elements

- Vectors
  - Contain a row or column of data (1 dimension only)
  - All elements are the same type
  - Allow for efficient storage and fast operations

```
//V2 and V3 are already vectors...
V1 = ArrayToVector({2, 4, 6})
V2 = ArrayToVector({10, 100, 1000})
V = V1 * V2
ShowArray({V})
```



  - Read and set vectors from matrices and data tables

Language Elements

- Matrices
  - Referenced with “Matrix Currencies”
  - Point to a matrix file and core
  - efficient Two dimensional data storage

```
MC := MC1 + MC2  //Perform element-by-element addition of two matrices
```

  - New in TransCAD 6: Memory Only matrices can be used for scratch data
    - Can be much faster
    - Can run out of memory with large matrices

Language Elements

- Functions
  - Built-in functions

```
ShowMessage("Hello World!")    //Show a simple message
ShowArray({2, 4, 6})           //Display the contents of an array
RunMacro("My Macro")           //Runs a macro
RunDbox("My Dialog")           //Runs a dialog box
Return(True)                   //Returns (ends) a macro or dialog box
//... many many more...
```

  - Functions may return values

```
x = Round("2.1", 0) //x = 2.0
```

- Operators
  - *Most* standard math operators apply:

```
x = 1 / 2 + 5 * 2    //x = 10.5
x = Pow(10, 2)       //x = 100 (x = 10^2 does not work)
```

Language elements

- Loops
  - For Loops:

```
dim arr[5]
for i = 1 to arr.Length do
    arr[i] = i * 2
end
//arr[4] = ??
```

Avoid looping over long arrays or vectors to perform simple operations. Vector math is much quicker

  - While Loops:

```
ans = "Yes"
While ans = "Yes" do
    ans = RunMacro("Again?")   //User input macro
end
```

Language Elements

- If Statements
  - » Evaluate a True/False condition
  - » Uses = in a different context
    - – There is not a separate assignment operator (except for matrices)

```
MyVal = True
if MyVal = True then do
    ShowMessage("It's True!")
end
```

    - – Relational Operators

```
a = b   a < b    a > b   a <= b   a >= b   a <> b
1 between 1 and 5              //returns True
"GIS" Like "GI?"               //returns True
"TransCAD" contains "CAD"      //returns True
```

    - – and, or, & not (Lazy evaluation or call by need)

```
if x = "Platypus" or (x <= 100 and y > 500) then do
    if not x = "Platypus" then ShowMessage("It's not a platypus...")
    else ShowMessage("It's a platypus, but x and y were not checked")
end
//Shortcuts:  and --> &  or --> |  not --> !
```

Macros

- Macros contain code
  - Operations are performed in sequence as called
  - Macros end when a value is returned
  - Arguments can be passed to macros

```
1  Macro "Hello World"
2
3      ShowMessage("Hello World")
4
5  EndMacro
6
7  Macro "Multiply 5" (num)
8
9      ShowMessage("Your Number multiplied by 5 is " + string(num * 5))
10     Return(num * 5)
11
12 EndMacro
13
14 Macro "Call Multiply"
15     x = RunMacro("Multiply 5", 5)   //Can have up to 8 arguments
16     //x = 25
17 EndMacro
```

Dialog Boxes

- Dialog boxes interact with the user
  - Dialog box items contain code

```
20 Dbox "Interface" Title: "Five Times"
21
22     //This section is run when the dialog box starts
23     init do
24         ShowMessage("Starting up!")
25     enditem
26
27     //This edit item can only contain an integer value.
28     //The value typed in the box is stored in the variable num
29     //A prompt is placed to the left of the edit field
30     //the field is placed at location x=1, y=1
31     //the field is 10 characters wide and 1 character high
32     edit integer "Num" 1, 1, 10, 1 variable: num prompt: Number
33
34     //The do items (optional) are executed with the item is activated
35     button "Multiply" 1, 3, 10, 1.5 do
36         RunMacro("Multiply 5", num)
37     enditem
38
39     button "Exit" 12, 3, 10, 1.5 do
40         Return()  //Returns null and exits the dialog box
41     enditem
42 EndDbox
```

Objects

- GISDK is allows creation and use of Objects
  - Objects behave like physical objects
    - Each object is of a certain type: **classes** of objects
    - They have attributes or characteristics: **properties**
    - They have things they can do: **methods**

Objects – Example: A car as an object

- **Class** “Car”
  - I can create a new object of type or **class** “Car”
  - In GISDK: MyCar = CreateObject(“Car”)
- A car can have **properties**
  - Each car can have different attributes
    - Color (Red, Blue, Green, etc)
    - Make, Model, etc.
  - In GISDK:
    - MyCar.Color = “Green”
    - MyCar.Make = “Volkswagen”
    - MyCar.Model = “New Beetle”

Objects – Example: A car as an object

- A car can have **Methods**
  - Methods let the car actually *do something*
    - Accelerate, decelerate, lock, unlock, “Roll down the windows”
  - In GISDK:
    - MyCar.Accelerate(15) //Speed up by 15 mph
    - MyCar.Decelerate(5) //Slow down by 5 mph
- Check the property “Speed”
  - ShowMessage(“Speed is: “ + String(MyCar.Speed))
    - If the car started at 0 mph, what is the speed?

Objects – Example: A car as an object

```
1  //StartClass
2  Class "Car"
3
4      //This acts as an object "Constructor" and is run each time a new instance
5      //   of the object is created
6      init do
7
8          //An object can have Properties. Properties are attributes of the
9          //  object.
10         //
11         //All properties must be specified or set to null from within the
12         //  object before they can be set or accessed externally.
13         self.Color = "Beige"
14         self.Make = null
15         self.Model = null
16         self.Speed = 0 //Start at rest
17
18     enditem
19
20     //A class can have Methods that perform tasks
21     macro "Accelerate" (AddSpeed) do
22         if AddSpeed > 0 then self.Speed = self.Speed + AddSpeed
23     enditem
24
25     macro "Decelerate" (DecSpeed) do
26         if DecSpeed > 0 then self.Speed = Max(self.Speed - DecSpeed, 0)
27     enditem
28
29 EndClass
```

Objects – Example: A car as an object

- Create a new car object and adjust the speed
  - See what happens if we try to access an undefined property

```
Macro "Go"
    MyCar = CreateObject("Car")
    MyCar.Accelerate(15)
    MyCar.Decelerate(5)
    ShowMessage("Speed is: " + String(MyCar.Speed))

    //This will cause an error.  FuelType is not a valid property for
    //  the "car" class.
    MyCar.FuelType = "Diesel"

EndMacro
```

Objects – Example: A car as an object

- New properties can be defined from within the object, but not externally

```
31      macro "InitFuelType" do
32          self.FuelType = null
33      enditem
```

```
50      MyCar.InitFuelType()
51      MyCar.FuelType = "Diesel"
52      ShowMessage("Fuel Type is: " + String(MyCar.FuelType))
```

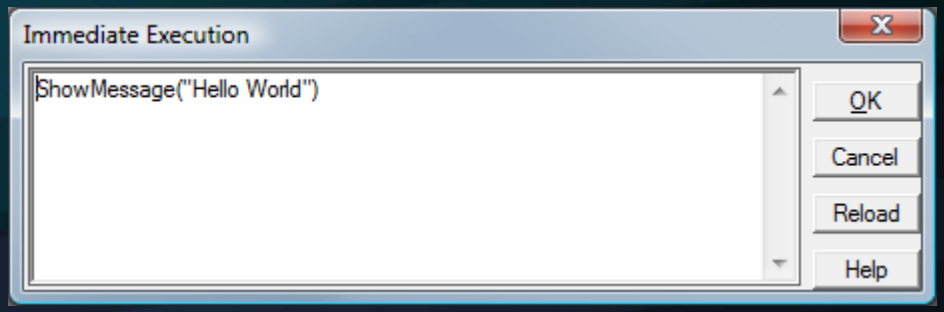
- All variables in class macros and the “init” constructor are local and cannot be accessed from another method
- The **self** variable in a class macro provides access to object properties and methods

Objects in the HCAOG Model

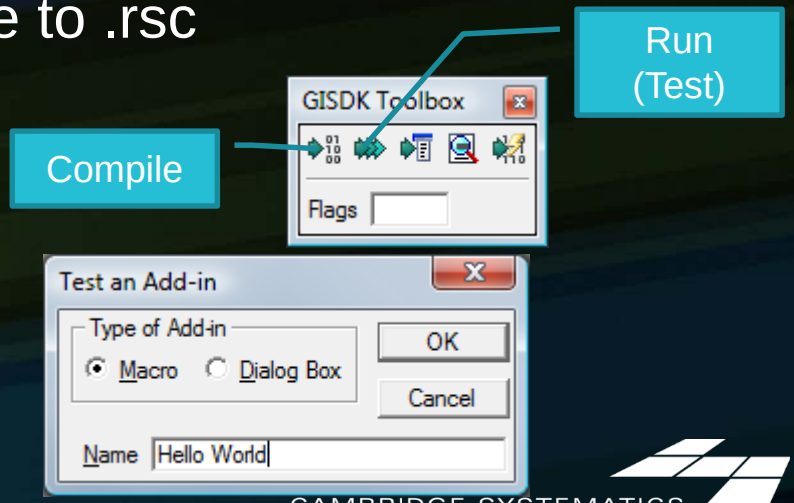
- Convenience:
  - The Utilities class → UT object in HCAOG
    - Easily accessed UT.MyFunction(Arg1, Arg2) Functionality
  - The Mapper class → MP object in the Dashboard
    - Set up properties for a desired map
    - Wrap up most of the functionality within the class, making map coding updates easier

Try It!

- Write a Hello Word line
  - » Tools → GIS Developer's Kit
  - » Immediate Execution



- Write a Hello World macro
  - » Create a new text file, rename to .rsc
  - » Write a macro
    - – Use the example above
    - – Compile the macro
    - – Run the macro



CAMBRIDGE SYSTEMATICS

Options Arrays

- Organized function input

```
1  Macro "Question"
2
3      Opts = null
4      Opts.Caption = "Please answer:"
5      Opts.Buttons = "YesNo"
6      Opts.Icon = "Question"
7      Opts.Default = 1
8
9      ans = MessageBox("Do you have a platypus?", Opts)
10
11     if ans = "Yes" then ShowMessage("How strange...")
12     else ShowMessage("Why not?")
13
14 EndMacro
```

Options Arrays

- Simply a set of {name, value} pairs

```
//Preferred syntax (usually)
Opts = null
Opts.Caption = "Please answer:"
Opts.Buttons = "YesNo"
Opts.Icon = "Question"
Opts.Default = 1

//Alternate syntax
Opts = {{"Caption",      "Please answer:"},
        {"Buttons",      "YesNo"},
        {"Icon",         "Question"},
        {"Default",      1}}
```

  - Options are often optional
    - The whole array can be null
    - Certain options can be missing (will revert to default)
  - Option names with spaces are allowed:
    - Opts.[Two Words]

Batch Mode

- Planning functions are run in Batch Mode:
  - » Record GUI input using the batch recorder
    - – Planning → Batch Editing
  - » This will create a rsc file
    - – Adjust option values as desired
    - – Replace option values with variables
  - » Compile and run a new rsc file

Batch Mode: Example

```
Macro "Batch Example"


    //Initialize the batch processor:
    //This must be done prior to calling batch procedures
    RunMacro("TCB Init")

    //(Put the directory in a variable for convenience)
    dir = "C:\\Users\\Sean\\Work\\OCTA\\Training\\SCAGLOCAL\\TripDistribution\\RunGrav\\"

    //Create an options array:
    Opts = null
    Opts.Input.[Input Matrix] = dir + "OfPkSkims.mtx"
    Opts.Input.[Target Core] = "Table 3"
    Opts.Input.[Core Name] = "Was Table 3"

    //run the procedure or operation:
    ret_val = RunMacro("TCB Run Operation", "Rename Matrix Core", Opts)

    //did the operation complete successfully or fail?
    // --> If success, the macro will continue (e.g., more functions)
    // --> After failure, the macro will goto quit: and exit
    if !ret_val then goto quit


    quit:

    //Close the batch processor
    //Argument 1 (ret_val) is True/False and indicates success/failure
    //Argument 3 shows the report on screen if true
    RunMacro("TCB Closing", ret_value, True)

EndMacro
```

Practice: Batch Recorder

- Make a simple model dialog box
  - » Run the gravity model interactively
    - – Turn on the batch recorder
    - – Note the Full run/dry run/no run option
    - – Use a gamma function
  - » Save the recorded results
  - » Create a dialog box
    - – Add gamma parameter text boxes
      - ➢ Consider including default values
    - – Add an OK and Cancel button
  - » Run the gravity model on OK
  - » Compile and run

Resource Code

HCAOG Structure

HCAOG Resource Code Files

- Humboldt.rsc
  - Primary resource code
  - Contains dbox “HCAOG Mode”
- Humboldt_Dashboard.rsc
  - Mapping functions
- Humboldt_Summary.rsc
  - Summary Report code
- Humboldt_Utilities.rsc
  - Technical Code (i.e., subroutines)
  - Commonly used functions (e.g., table modification, file access)
- Humboldt_scen9.rsc
  - Scenario management
  - Please limit distribution

The Default Scenario File

- DefaultScenario.ini
  - » Default scenario settings
  - » Place to change default filenames, parameters

```
; ------- Scenario Settings -------
[Info]
Name = New Scenario   ;Scenario Name
Input Directory = C:\HCAOG Model\Inputs\ ;TODO - not currently used
Output Directory = C:\HCAOG Model\Outputs\ ;TODO - not currently used

; ------- Input Files -------
[Input.Network]
Value = Humboldt_Network.dbd
Desc = Roadway Geographic File
[Input.TurnPen]
Value = TPEN.bin
Desc = Turn Penalty File
Split = Optional
Choice = Optional
[Input.Database]
Value = Humboldt_Database.mdb
Desc = Model Input Database
```

The Default Scenario File

- Contains all of the default scenario information
  - Filenames (input/output)
  - Parameters
  - Tables (parameter arrays)
  - Access Database Table Names
- Each item has a description
  - This is shown in the scenario editor description
- New scenarios are created based on this file
- Old scenarios are (partially) updated when the file changes
  - Items can be added/removed
  - Changed values are not reflected in old scenarios

Humboldt.rsc

1. HCAOG Model Defaults
   - Sets program environment (program directory, settings directory, filenames)
   - Creates utility objects
   - Defines model steps,
   - Loads default scenario information

2. HCAOG Model Step Info
   - Dialog button names
   - Model step names / macro names
   - Disable sub-steps
   - Enable/disable sub-steps by default
   - Progress bar settings

Humboldt.rsc

3. HCAOG Model Dialog Box
   - » Call this to start the model (The defaults and step info are called from this dialog box)
   - » Contains dialog box setup (init)
   - » Contains dialog box items (dialog box layout)
   - » Contains dialog box macros (controls model runs)

4. Model Macros

Start Here
(The earlier stuff doesn't usually change)

   - » Consistent with model step info
   - » Contain the basic model code
     - – Some Detailed technical code is contained in the utilities file

Walkthrough – Step 1.1

- In-Editor Walkthrough

Debugging the Code

- TransCAD has an interactive debugger
- Two ways to use:
  - » Set a break in the code
  - » Turn on “debug mode” when the model is crashing
- Try It: Set a break on the line shown below
  - » Identify the line number in your editor
  - » Ctrl-F, then find the line shown below

```
NextStep = "Highway Network Setting"
SetStatus(1, NextStep, )

        //Add centroids and turn penalties
        Opts = null
        Opts.Input.Database = dbd_file
        Opts.Input.Network = net_file
        Opts.Input.[Centroids Set] = {dbd_file + "|" + node_lyr, node_lyr, "C", cent_qry}
        if GetFileInfo(turn_file) <> null then Opts.Input.[Spc Turn Pen Table] = {turn_file}
        Opts.Global.[Global Turn Penalties] = {0, 0, 0, 0}
```

Debugging the Code

- In the stopped code, step through each line
  - » Check the variable window, the watch window

- View a scenario array
  - » Save a scenario file
  - » Write a small macro
    - – Use LoadArray
  - » Debug the macro and view the contents