Script and Dialog Tools

Version 2.0016, Last revised April 2, 2007

TOPICS: ArcView 3.x, Script, Dialog, ODB, Project, View, Table, Extension

Aka: script_tools_jen.avx

AUTHOR: Jeff Jenness

Wildlife Biologist, GIS Analyst
Jenness Enterprises
3020 N. Schevene Blvd.
Flagstaff, AZ 86004 USA

Email: jeffj@jennessent.com
Web Site: http://www.jennessent.com
Phone: 1-928-607-4638

DESCRIPTION: This extension automates many of the functions I use regularly when I'm writing scripts or dialogs, plus provides several additional functions to most interfaces. In short, it offers the following:

1) *Project tools:* Tools to save all new scripts, dialogs, tools, buttons and menu items to an object database, and also to extract those objects into a new project. Includes a tool to open dialog editor documents (*.ded files) and a tool to copy complete documents (i.e. Views, Layouts, etc.) and to easily back up your project file.

2) *Dialog tools:* Tools to help you create, save and maintain dialogs, rearrange the control tab order, generate basic scripts and tedious code, and produce reports describing all aspects of your dialogs. Also generates several standard dialogs automatically.

   Version 2.x includes tools to help convert existing ArcView 3.x dialogs into VB6 Forms.

3) *Script tools:* Tools to compile all scripts, close all scripts, shrink all scripts, shrink individual scripts, and search all scripts for specified text. Also generates a variety of new scripts and code snippets automatically and generates reports describing interrelationships between scripts. Also provides keystroke shortcuts to find text, shift script text to the right or left, comment/uncomment code, and select all text in the script. Also provides a tools to send you to a particular character location in your script, and to describe and number your script. Several tools to identify errors in the project.

   Version 2.x includes tools to analyze and export VB6 Projects.

4) *Table tools:* Tools to delete multiple fields, identify table source and linked table sources, describe field information, add record number fields and unjoin tables. Also generates code that would either find or make fields identical to those in the table. Also provides an "Unlink" button and tools to export and import tables from Excel.

5) *View tools:* Tools to show all theme legends, hide all theme legends, set all themes active or inactive, and generate graphic color bars of a theme legend to enhance layouts or legends. Also a button to undo the "Undo Zoom" function, and several tools to convert theme shape types and to convert graphics to themes.

REQUIRES:

This extension requires that the file "avdlog.dll" be present in the ArcView/BIN32 directory (or $AVBIN/avdlog.dll) and that the Dialog Designer extension be located in your ArcView/ext32 directory, which they usually are if you're running AV 3.1 or better. The Dialog Designer

doesn't have to be loaded; it just has to be available. If you are running AV 3.0a, you can download the appropriate files for free from ESRI at:

http://support.esri.com/index.cfm?fa=downloads.patchesServicePacks.viewPatch&PID=25&MetaID=483

RECOMMENDED CITATION FORMAT: For those who wish to cite this extension, the author recommends something similar to:

Jenness, J. 2007. Script and Dialog Tools (script_tools_jen.avx) extension for ArcView 3.x, v. 2.0. Jenness Enterprises. Available at: http://www.jennessent.com/arcview/script_dialog_tools.htm.

Please let me know if you cite this extension in a publication (jeffj@jennessent.com). I will update the citation list to include any publications that I am told about.

Table of Contents

General Instructions:

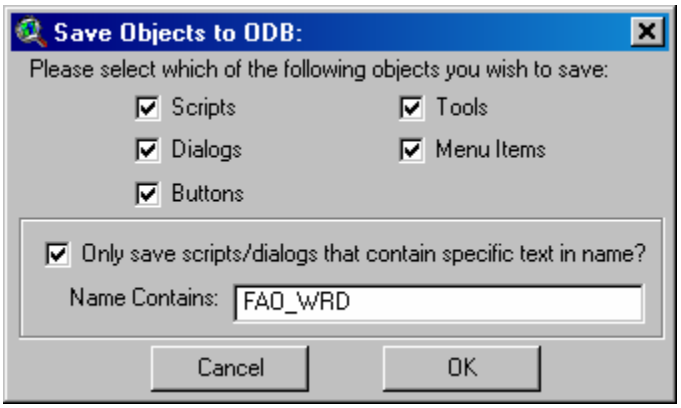
1) Begin by placing the "script_tools_jen.avx" file into the ArcView extensions directory (../../Av_gis30/Arcview/ext32/).

2) After starting ArcView, load the extension by clicking on **File --> Extensions… ,** scrolling down through the list of available extensions, and then clicking on the checkbox next to the extension called "Script/Dialog Tools."

Project Buttons: 

This extension adds three buttons to the Project button bar:

The  button saves things into an object database. It also cleans and compiles all scripts and dialogs before saving them. Click it and you'll see the following dialog (at the moment it doesn't save tool menus):

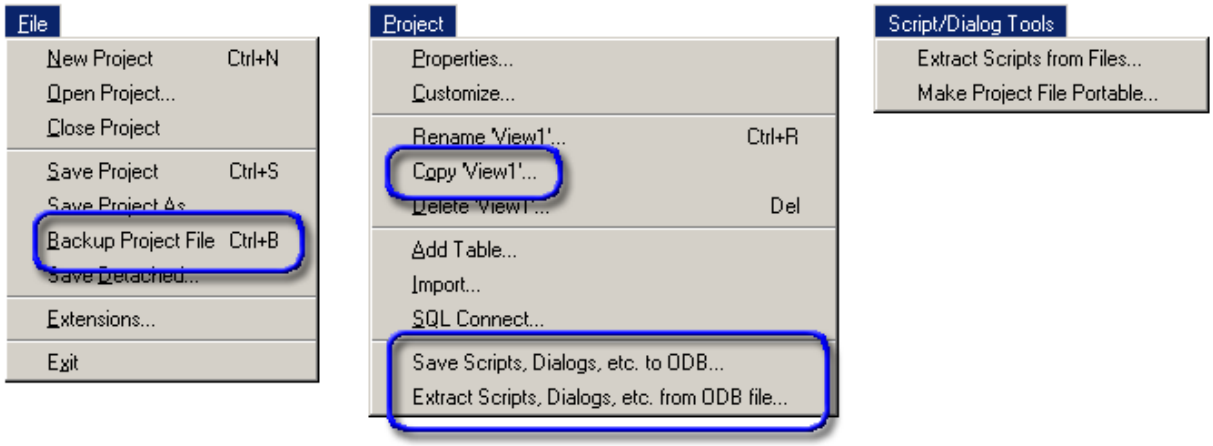


This is a useful function if you want to take all your tools, buttons, menus, scripts and dialogs and put them into a fresh new project file (for example, if you want a clean project file to use for building an extension).

The  button extracts all the scripts, dialogs, buttons, tools and menu items from the object database and installs them into your current project. It won't install a component that already exists, so you shouldn't get multiple copies of the same script or button.

The  button opens a Dialog Editor document file (*.ded file) on the hard drive. This is the same function as the  button in the Dialog Editor button bar, but the advantage of offering it in the Project button bar is that you don't have to create a new empty dialog before you can access the button.

Project Menu Items:



- *Backup Project File:* This saves a copy of your current project file to a new name in the same directory as your current project file. It appends the date and time to the new name. For example if your project was named "this_project.apr" and you clicked the backup function at 10:30:23 on June 20, 2003, this function would save a copy of the current state of your project to "this_project_06202003_103023.apr". This function does not save the current state of your project to "this_project.apr" though! Use [Control]-S for that. This function is repeated in the File menu of all the documents..
- *Copy 'Document Name':* This makes a copy of the current selected document ("View1" in the illustration) and pastes it into the project. In this example, clicking this menu item would generate a new View in your project named "View1_copy", which would have all the same themes, graphics, etc. as "View1" did.
- *Save Scripts, Dialogs, etc. to ODB…:* This function is identical to the  button described above.
- *Extract Scripts, Dialogs, etc. from ODB file…:* This function is identical to the  button described above.
- *Extract Scripts from Files:* Lets you extract scripts from Project files (*.apr), Extensions (*.avx), Avenue source code (*.ave), Object Database files (*.odb) and text files. You can choose multiple files at one time and this tool will query you as to which specific scripts you want to extract.
- *Make Project File Portable:* Goes through a project file and replaces any pathnames (except those referring to a relative location, like $AVHOME) with relative pathnames, referring to a single folder of data that will be distributed with the extension. If the data is not already in that folder, this function will copy the data into the folder as it modifies the path

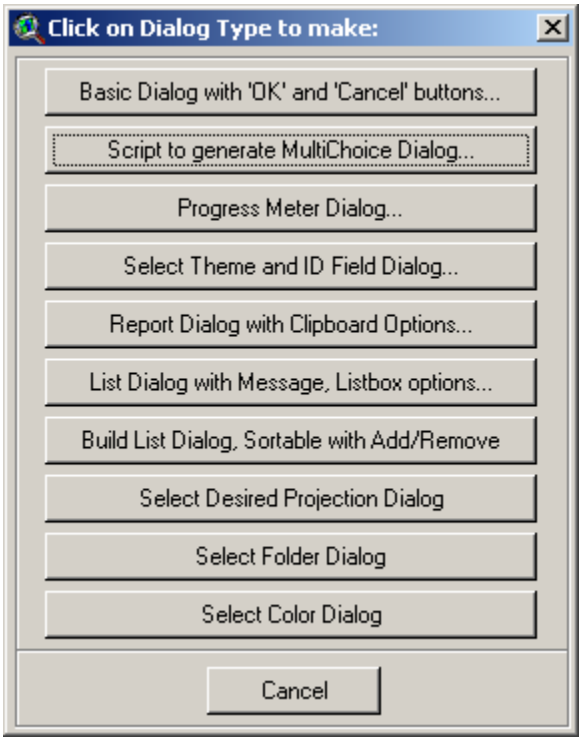
Dialog Buttons: 

This extension adds 11 buttons to the Dialog button bar:

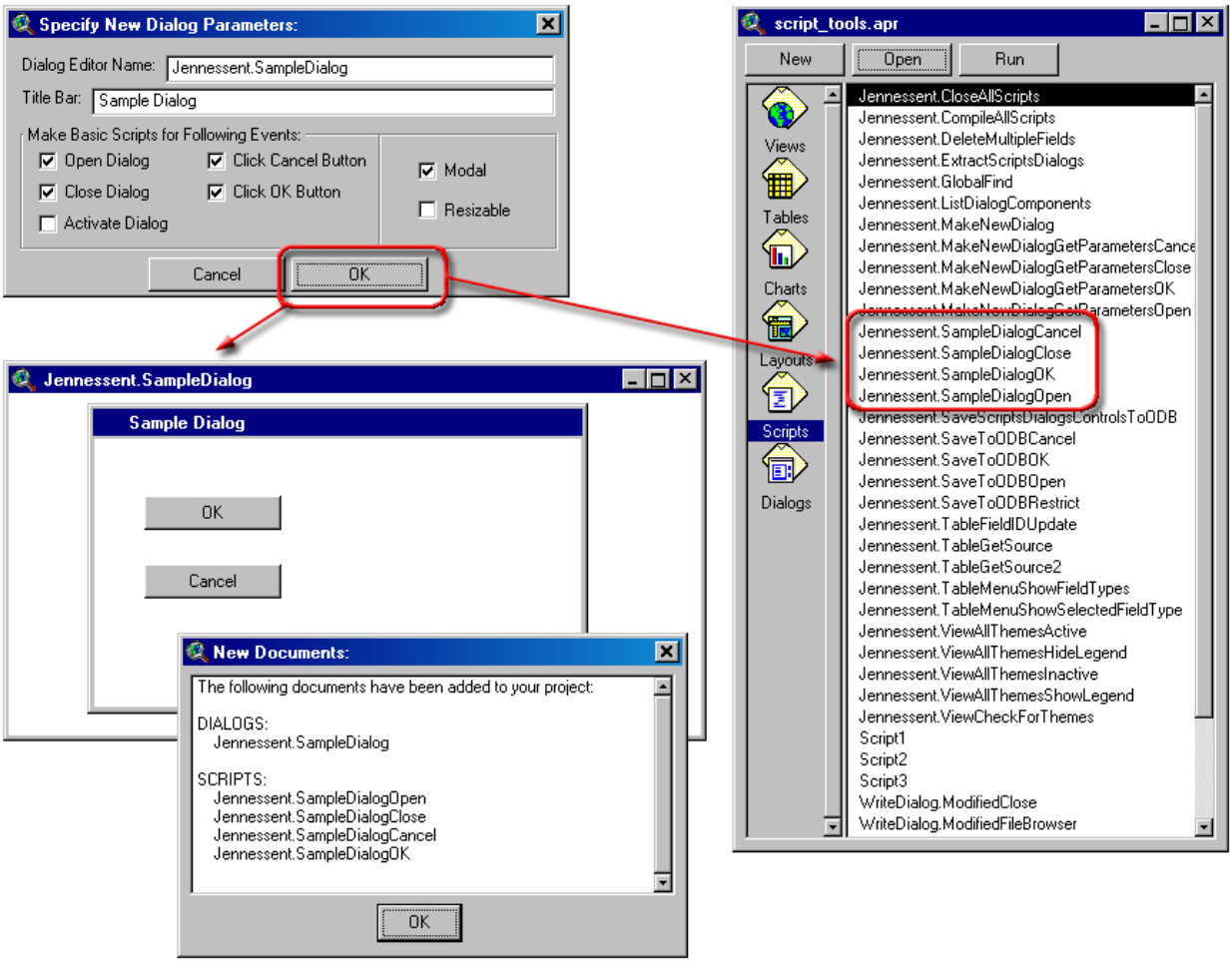
The  button compiles all dialogs.

The  button is a modified version of the "Save Dialog" function. The difference is that it does a much more extensive search for attached scripts, plus it has a different suggested default name for the extension. Regarding the scripts, it goes through all attached scripts, then looks for all scripts referenced by those scripts, then continues digging for referenced scripts until it's gone through a maximum of 30 levels. It takes a little longer than the original version of the button, but I think it's worth it.

The  button lets you generate one of five basic dialog types. Click this button and you'll be prompted to specify which dialog type to create:

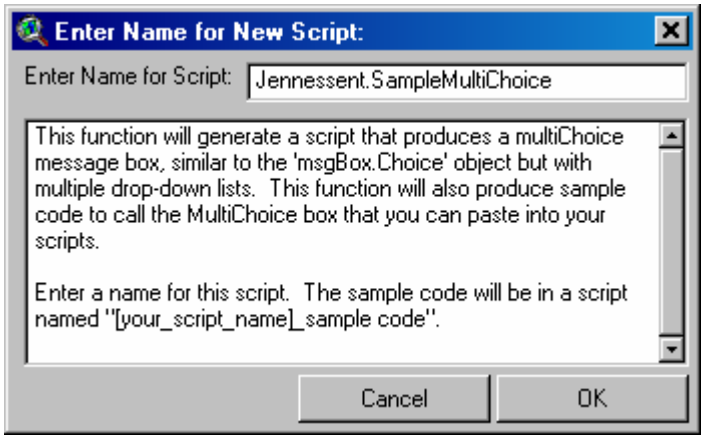


Basic Dialog with 'OK' and 'Cancel' buttons…: The first option simply makes a new basic dialog containing an "OK" and a "CANCEL" button with attached scripts that close the dialog. It also makes Dialog Open, Close and Activate scripts and attaches them to the dialog. When you click on the button, you'll be queried for the name of the dialog and the title to appear on the dialog. Your new dialog will be automatically created, compiled and added to your project. You have the option to create generic scripts for the Dialog Open, Activate and Close events and the OK/Cancel button Click events. The generic Open script centers the dialog and identifies the OK and Cancel buttons. The generic Close script sets all object tags to nil. You also have the option to make the dialog modal and/or resizable. The generic Cancel button sets the modal result to nil if the dialog is modal.

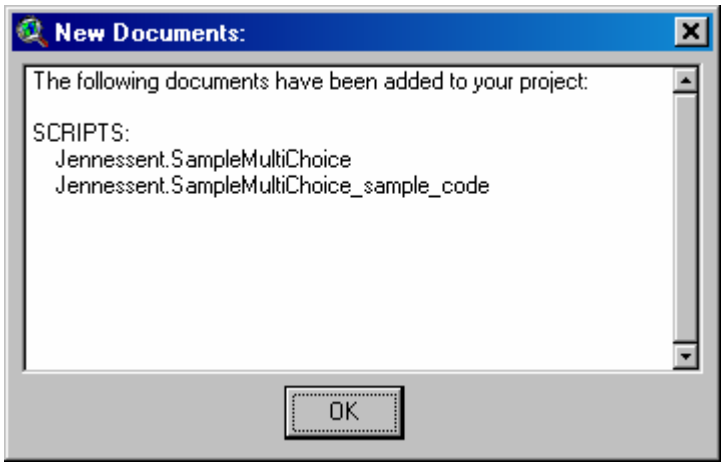


See "Basic Dialog Scripts" in the appendix for examples of these scripts.

Script to generate MultiChoice Dialog: This option doesn't actually produce a Dialog Editor in your project, but rather a script that you can call to generate a MultiChoice dialog on the fly. This multichoice dialog is kind of a cross between the "MultiInput" message box and the "Choice" message box. It has multiple drop-down boxes generated from a set of lists that you send to it. This option also generates a script of sample code to show you how to use the function. Click this button and you'll be prompted for the name you would like to call the script:



When you click the "OK" button you'll see a list of the scripts that were added to your project:

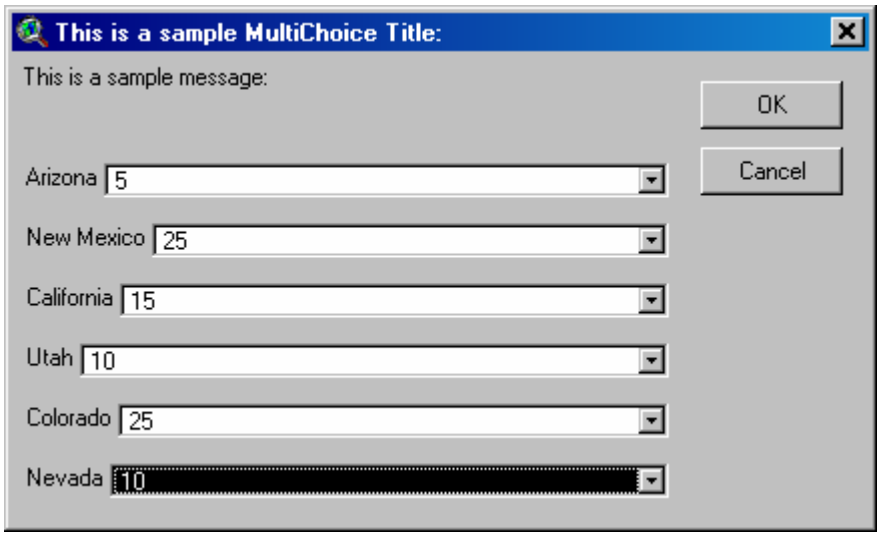


Review the "Sample Code" script for some examples on how to use this script. Essentially you send 4 objects to the script consisting of a message, a title, a list of labels and a list of lists for the drop-down combination boxes. The script then generates a MultiChoice dialog on-the-fly based on how many lists you send it. The lists of objects for the drop-down listboxes can contain any type of objects. They are not limited to string or number objects.

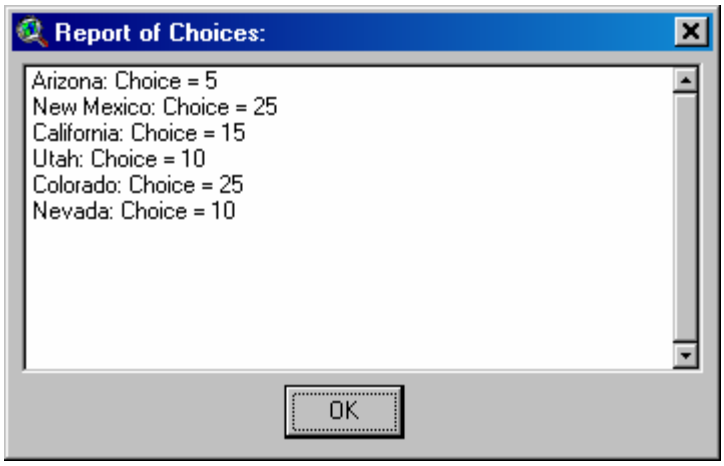
For example, if you had a list of 6 states and wanted to let the user select from a drop-down list for each of these states, you could use the following line of code to return the 6 selected values:

```
theChoices = MsgMultiChoice.DoIt({theMessage, theTitle, theListOfLabels, theListOfLists})
```

The script would then generate a MultiChoice dialog with 6 drop-down boxes:

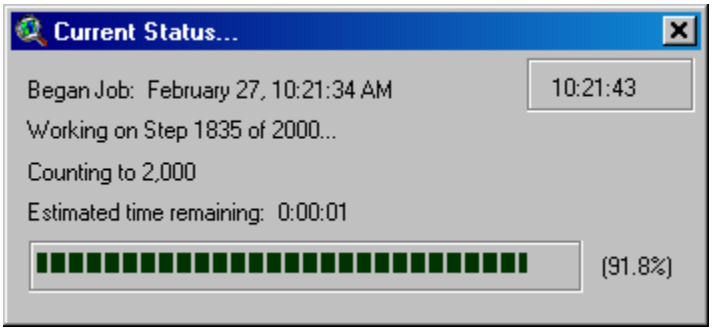


If you run the Sample Code script, it will show you a report of the 6 values you chose. This is not a function of the actual MultiChoice script, but rather just an illustration of it's use. You can do anything you want with the selected list of items. This dialog is also resizable by dragging on a corner, so you can expand the dialog if the list items or labels are too long to fit on the screen.

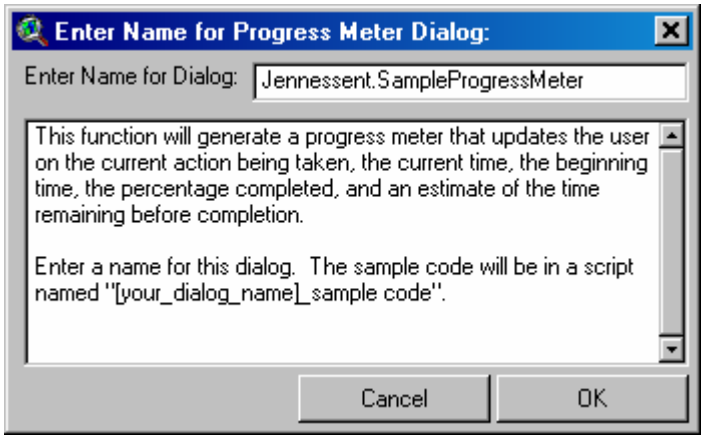


See “MultiChoice Scripts” in the appendix for examples of these scripts.

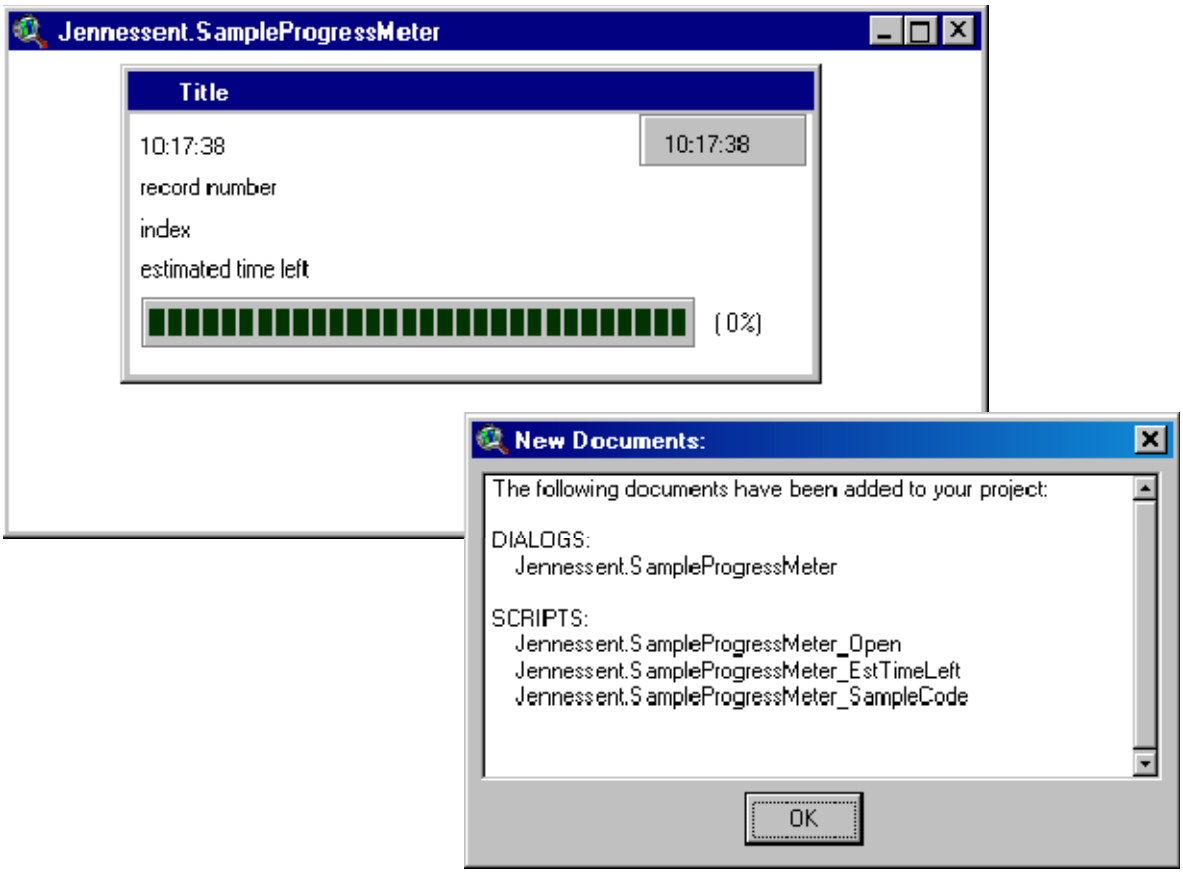
Progress Meter Dialog…: This Progress Meter is a modeless dialog you can incorporate into a script or extension to show what ArcView is currently working on, how long it's been working and how long it'll probably take to finish. It's especially well adapted for scripts that have lengthy "for each" loops that take hours, days or weeks to finish. This is basically the same dialog as you’ll find at http://www.jennessent.com/arcview/progress_meter.htm.



Click this button and you’ll be prompted for the name of the Progress Meter Dialog Editor document:



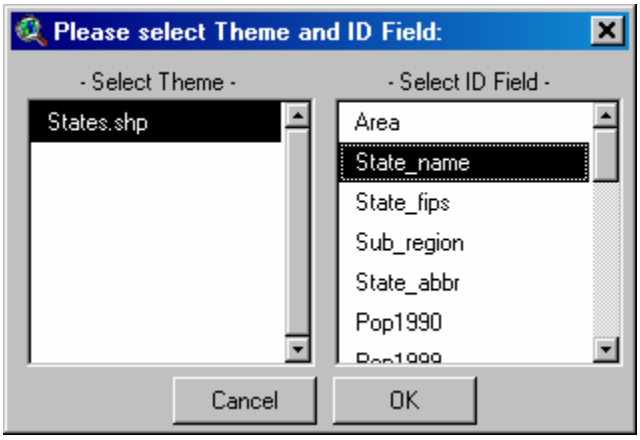
Click “OK” and the tool will produce a progress meter dialog and 3 scripts:



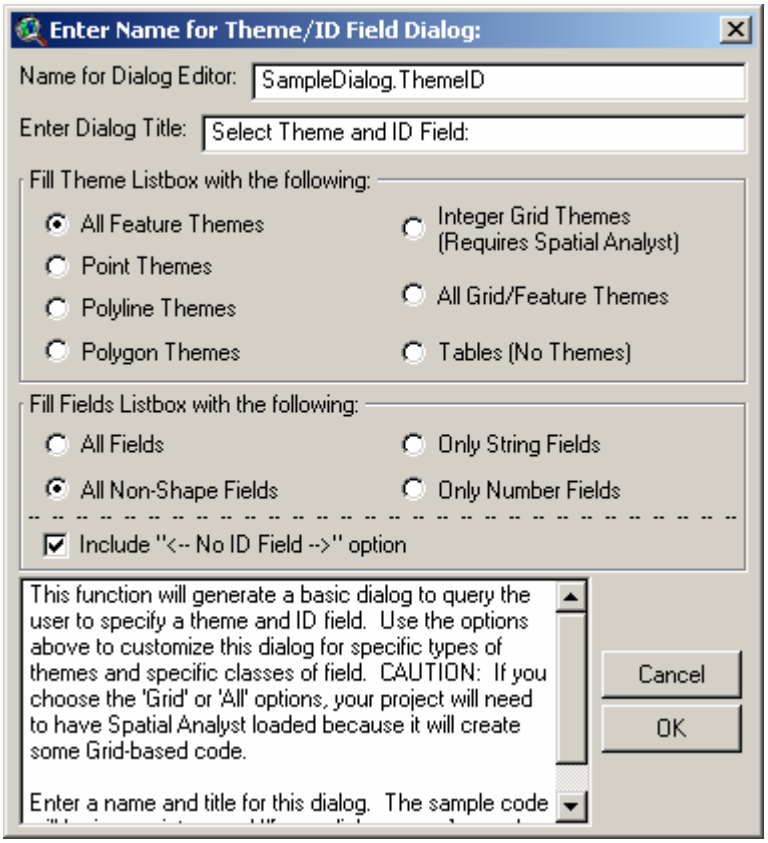
The "Open" script centers the dialog on the page and clears out any existing text from the labels. The "EstTimeLeft" script is the script you call to update the progress meter. You essentially send it data regarding what step you are on and how many steps the process is going to take, and the "EstTimeLeft" script estimates the time remaining and updates the progress meter.

The "Sample Code" script illustrates how to use the progress meter. You can run the sample code script to see the progress meter run through two functions; first by counting to 2,000 and updating itself at every increment, and then by counting to 40,000 and updating itself every second. See "Progress Meter Scripts" in the appendix for examples of the scripts.

Select Theme and ID Field Dialog…: This is a simple dialog that opens with two listboxes, a Cancel button and an OK button. The dialog is intended to query the user for a particular theme and a particular field from the theme attribute table (often an ID field). The first listbox is preset to list the themes in a view and the second listbox is set up to list the fields of the selected theme. Optionally, you can preset the first listbox to show all the tables in the document.



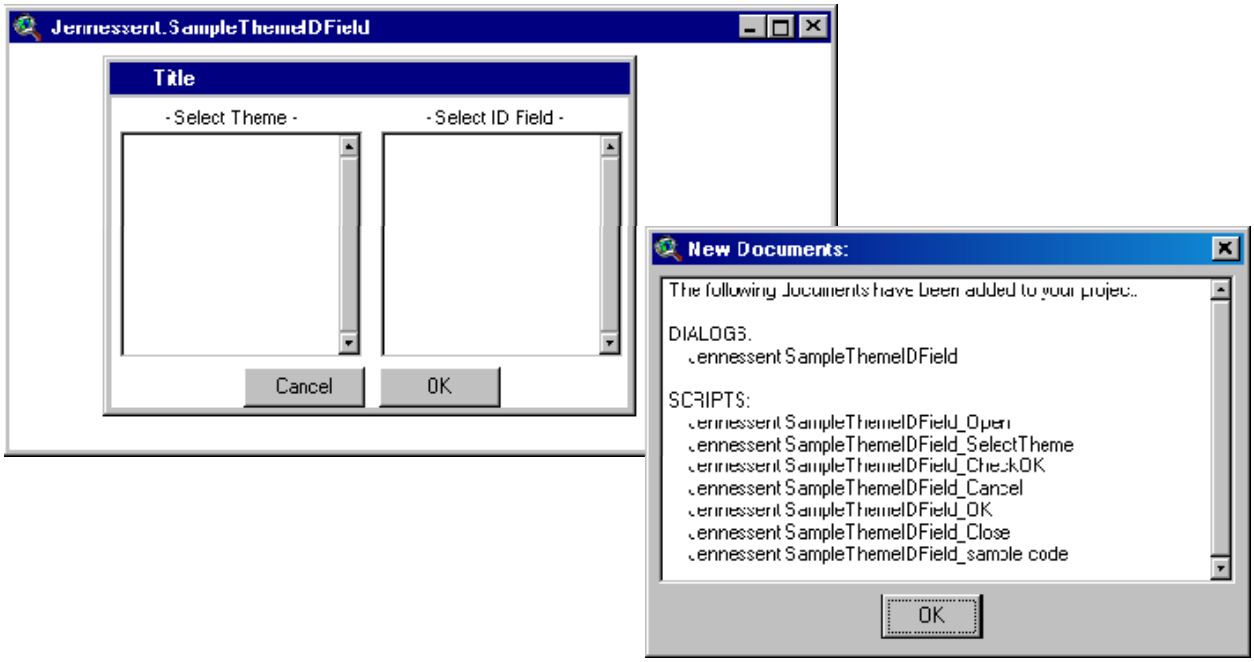
Click this button and you'll be prompted for several parameters:



The "Name for Dialog Editor" will be the name of the Dialog Editor document. The "Title" is the text that will appear in the blue bar at the top of the dialog. For some reason this title does not always show up in the Dialog Editor window, but the title appears correctly when the dialog itself is opened.

You can preset the dialog to show only a subset of the themes in the Themes listbox, and you can also restrict the fields to specific field types. If you want to change any of these after you have created the Dialog Editor, you can easily make the modifications in the scripts. These presets just save time up front if you know exactly what type of themes and fields you want to query for.

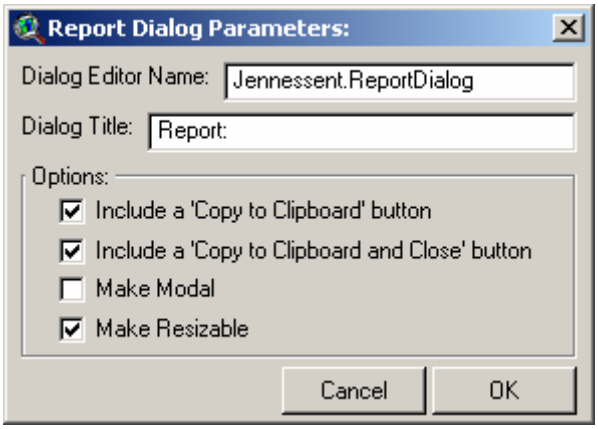
Click “OK” and the tool will generate the dialog and 7 scripts:



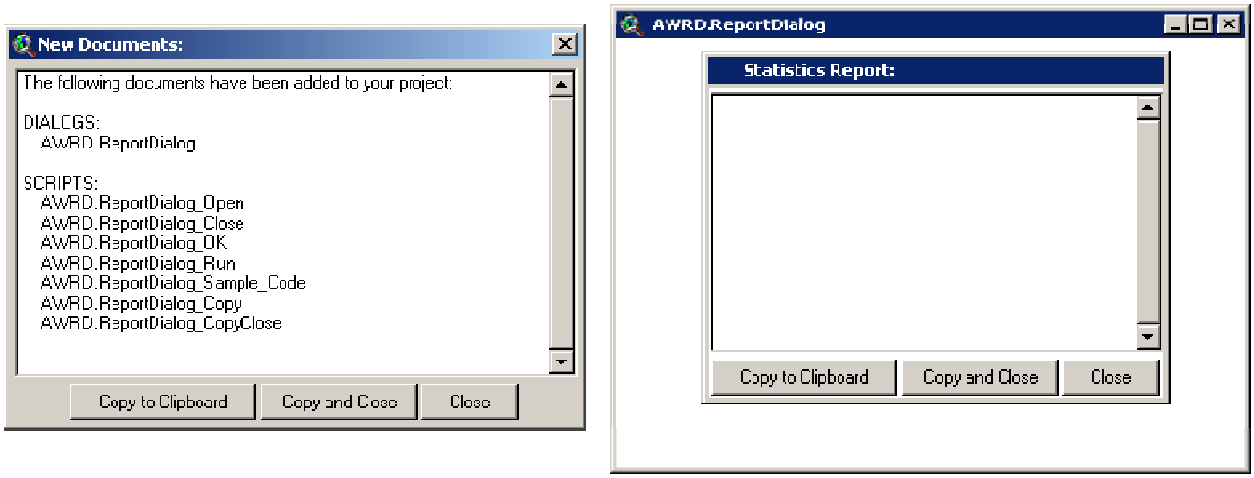
All the scripts except the “Sample Code” script pertain to the dialog itself and control what happens when you select a theme or field, or click on the OK or Cancel buttons. The “Sample Code” script illustrates how to use the dialog. For examples of these scripts, see “Theme and ID Field Scripts” in the appendix.

Report Dialog with Clipboard Options…: This produces a dialog composed primarily of a text box, with optional buttons to copy the text to the clipboard. The advantage of this dialog over the basic “msgBox.Report” is that this can be either modal or non-modal, so you can keep it open and updating as the user works:

Click the button and you’ll be prompted for a Dialog Editor name, a Dialog title, and which options you prefer:

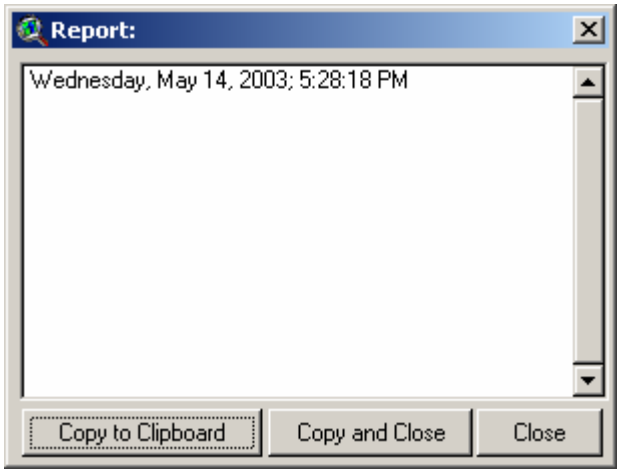


After you click “OK”, the extension will produce a dialog editor and several scripts for you. For examples of these scripts, see “Report Dialog Scripts” in the appendix.



Review the "Sample_Code" script for an easy way to operate the report dialog. Basically you can run the "Run" script with two parameters and the "Run" script will open the dialog for you. For example, if you wanted to make a report stating the current date and then let the user close the dialog, you would do it as follows

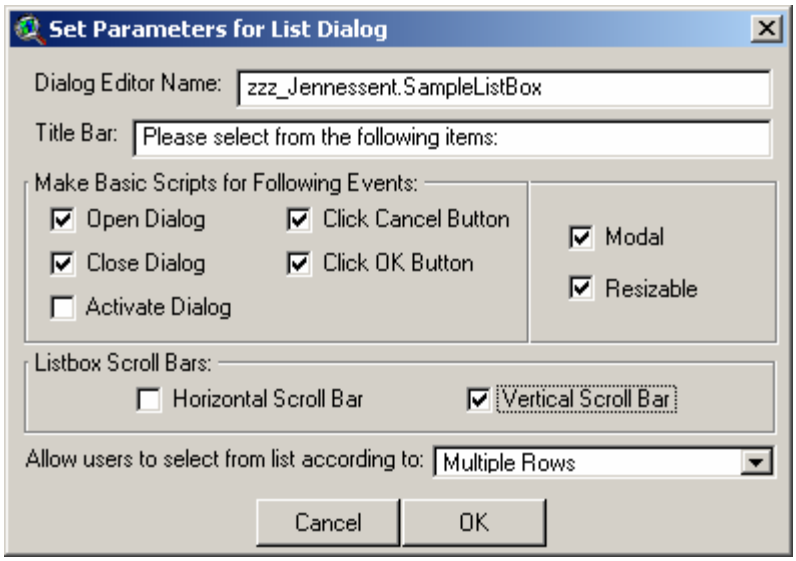
```
theDialog = av.FindDialog("[the name you specified"]
MakeReport = av.FindScript("[the dialog name you specified]_Run")

theTitle = "Report:"
theText = (Date.Now.SetFormat("dddd, MMMM d, yyyy; h:m:s AMPM").AsString)
MakeReport.DoIt({theText, theTitle})
```



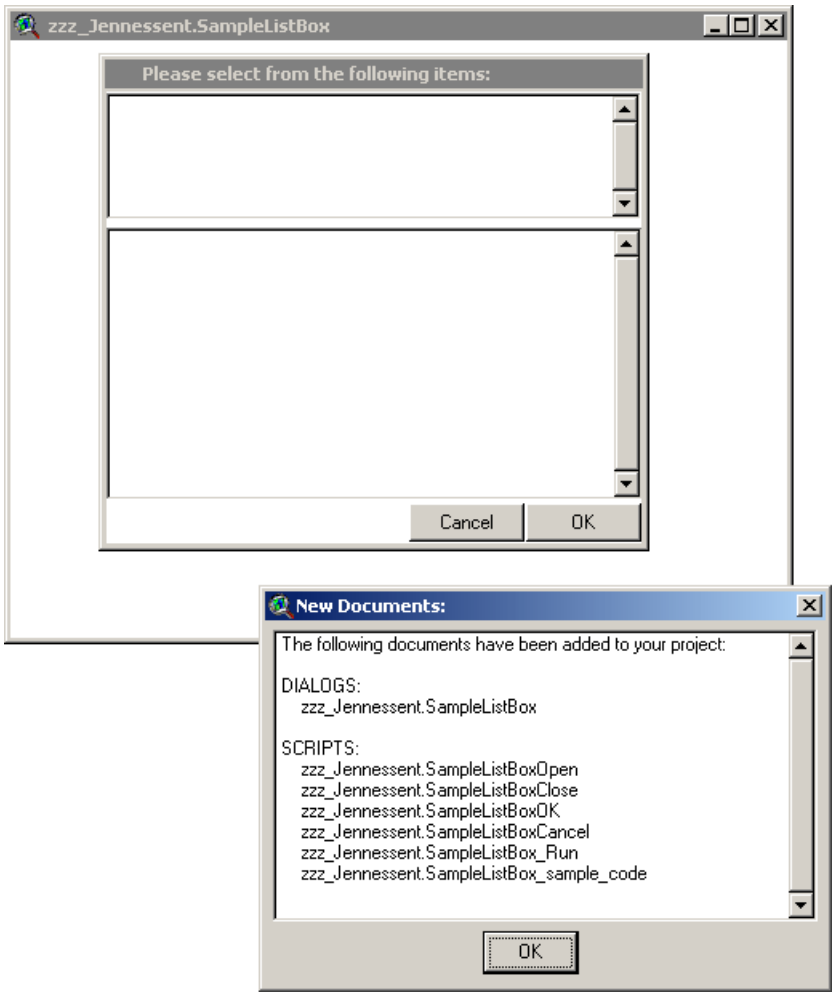
The dialog can be closed either by clicking the "Close" button or, if it is a non-modal dialog, by the "Close" request. Other potential uses of this dialog might be to generate a running report of feature attributes that a user might click on.

List Dialog with Message, Listbox options…: This provides an alternative to the standard "Msgbox.List" option, with the advantage that your listbox can have multiple rows and columns and that the list can contain more than just strings and numbers.

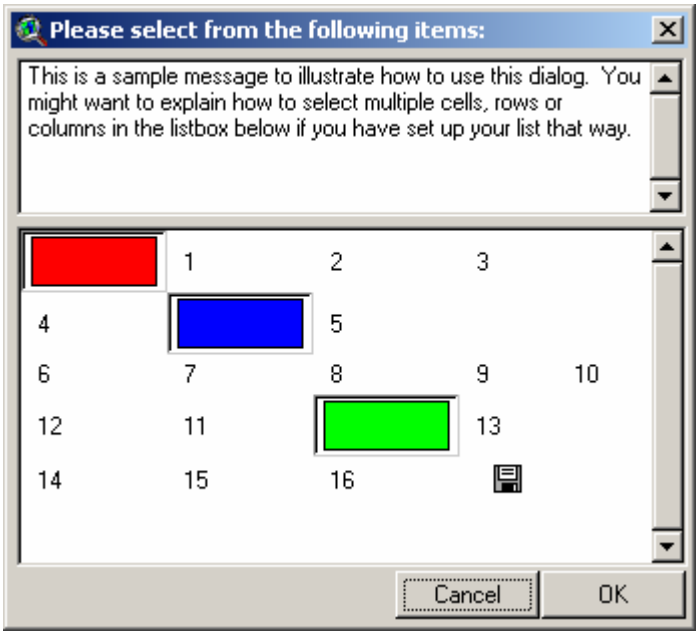
Click the button and you'll be prompted to give a name for the Dialog Editor document and a title for the dialog, plus several parameters:



Click “OK” and you’ll get a new dialog editor plus a report on the new scripts that were created:

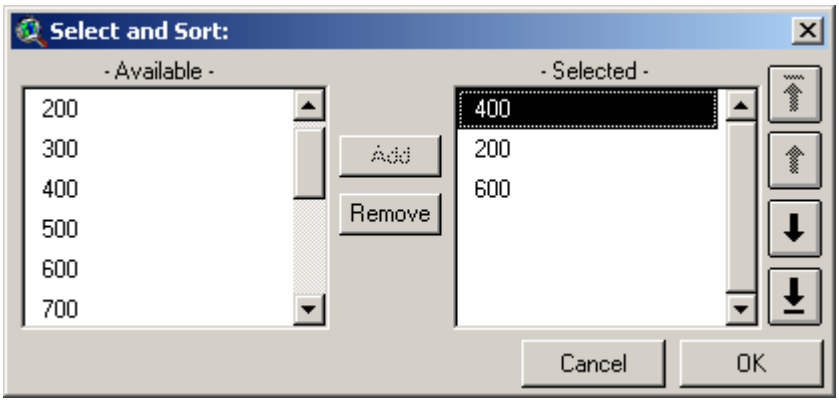


If you open and run the script "[your dialog name]_sample_code", you will get an illustration of how the dialog works. Notice that we were able to add symbols and icons to the listbox., which is not possible with the standard "msgBox.List" function offered with Avenue.



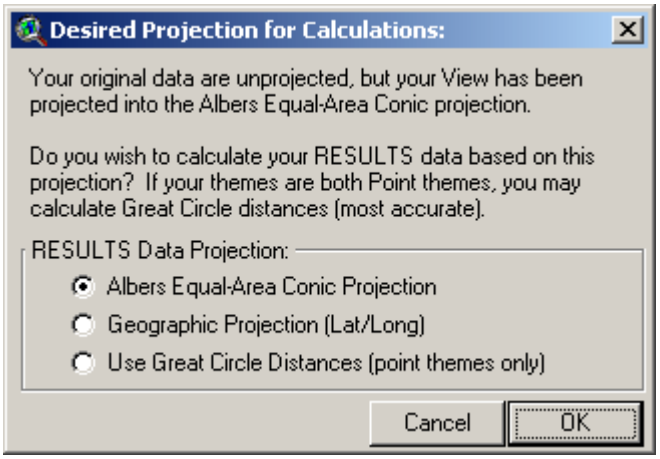
Once you've made the basic list dialog, you will likely want to customize it further with scripts describing what happens when certain cells get clicked on, or possibly customizing the cell height and/or width. For examples of the scripts that get generated automatically, see "List Dialog Scripts" in the appendix.

Build List Dialog, Sortable with Add/Remove: This generates a dialog where a user can select items from a list, and sort their selection. It returns a list of the selected objects. If you open and run the script "[your dialog name]_sample_code", you will get an illustration of how the dialog works.



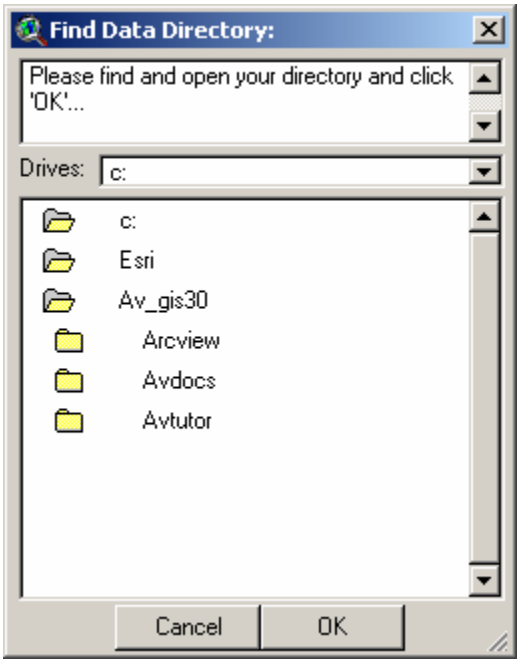
For examples of scripts that get generated automatically, see "Sortable List Scripts" in the appendix.

Select Desired Projection Dialog: This is intended for calculations in which the projection will affect the output, and in which it is unclear what projection the user wants to use for calculations.

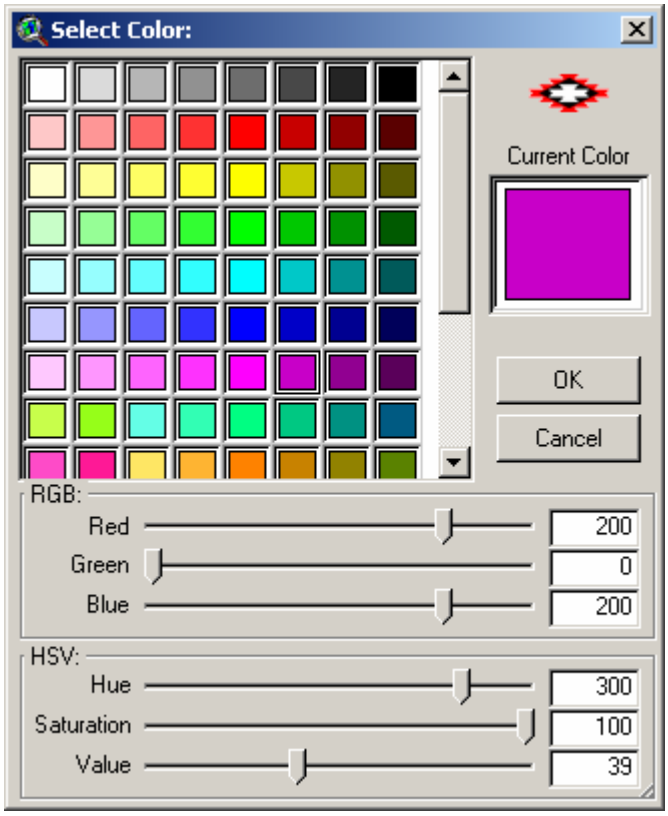


For examples of scripts that get generated automatically, see "Select Projection Scripts" in the appendix.

Select Folder Dialog: This opens up a simple dialog that allows you to select a folder, not a particular file. It is a little more intuitive that forcing the user to use the FileDialog to select a file in a folder, then extracting the directory of that file when you only need a folder.



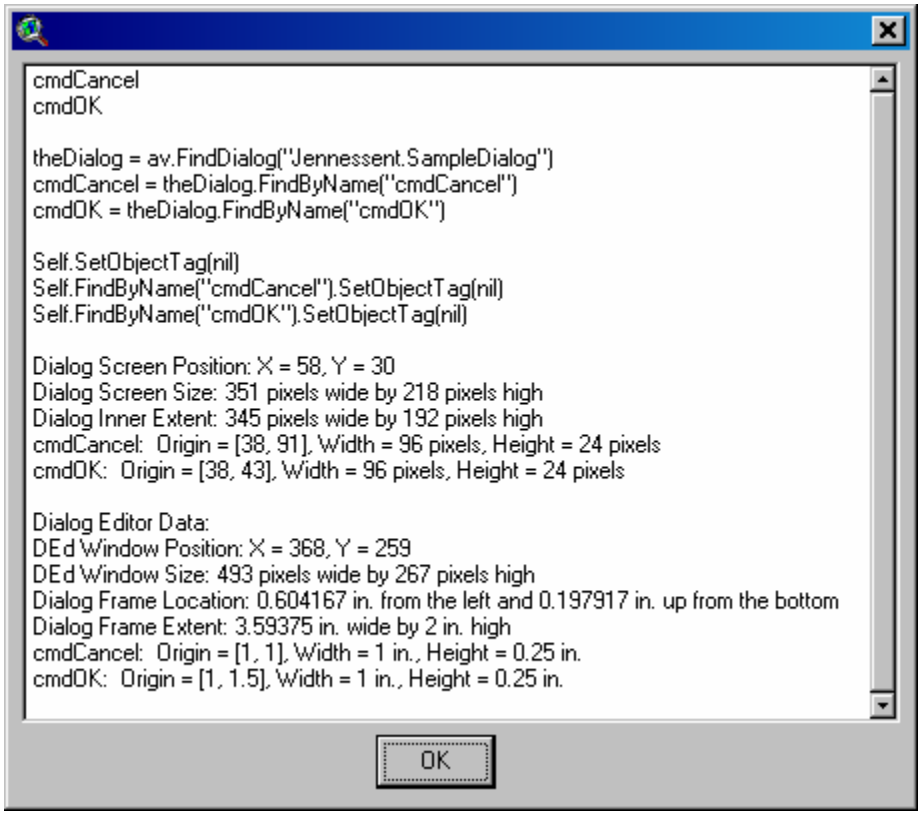
Select Color Dialog: This is a more advanced version of the standard ArcView color dialog. It has the advantage of letting you easily generate your own color within the primary dialog. It also lets you specify colors using either Red/Green/Blue (RGB) or Hue/Saturation/Value (HSV). This dialog includes a "Sample Code" script to illustrate its use.



The  button gives you information on the current dialog components. When you click it, it returns 5 lists of all the dialog components:

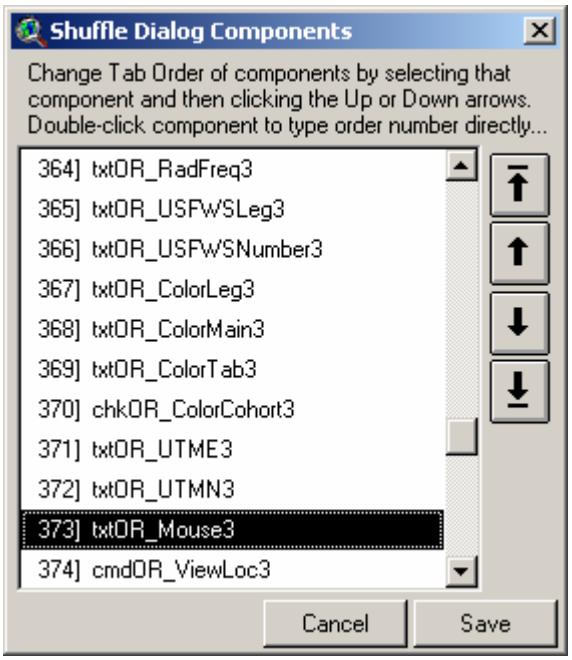
1) The first list simply names all the components.
2) The second list formats the names as "xxx = theDialog.FindByName('xxx')" and is intended to be copied and pasted into the headers of dialog scripts. This saves a lot of time when you want your dialog scripts to check/alter other dialog components.
3) The third list formats the names as "self.GetDialog.FindByName('xxx').SetObjectTag(nil)" and is intended to be pasted into the "Close" script. Use this only if you want to clear your dialog object tags upon closing in order to free up memory. There are times you want your dialogs to save the object tags so you won't use this all the time.
4) The forth list shows the exact position on the dialog of all the components. This is very useful if you want some dialog components to move when you click a button. Simply arrange the components in the locations you want, click the button and record the coordinates, then use those coordinates in your "MoveTo" requests.
5) The fifth list shows the exact position of the Dialog Editor graphic controls. This is useful if you want to generate dialog editors and correctly position the graphic controls.

The final report for our earlier sample dialog looks like this:



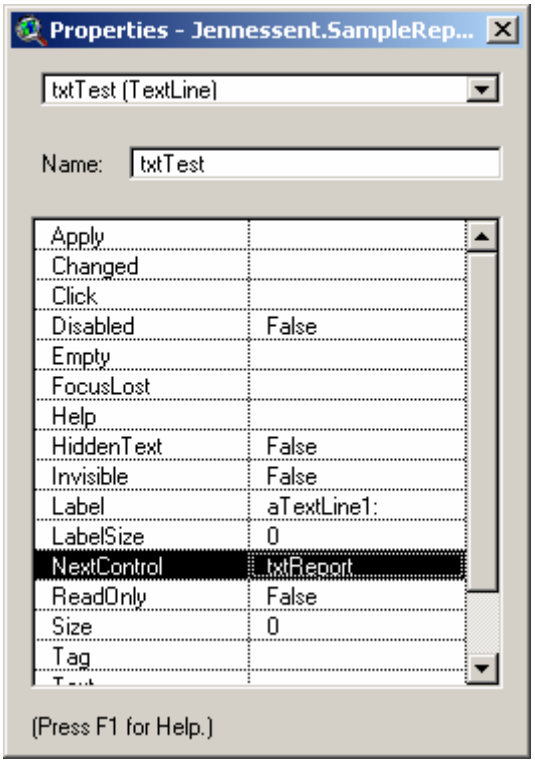
Simply copy the portions you want and then paste them into your scripts.

The  button gives you an easy way to adjust the tab order of the controls in your dialog. Click it and it'll show you a list of all the controls currently there, in their default tab order:



Select the control you would like to move in the tab order and then use the up and down arrows to shift it. Alternatively, you can double-click on the control name and be prompted to type in the correct order number:

IMPORTANT: This function does not override any custom "Next Control" settings you may have made:



This tool actually just rearranges the order of GraphicControl objects in the dialog editor's GraphicsList and then recompiles the dialog, which in turn affects the tab order. Therefore any "NextControl" designations you may have set will still be in effect. This tool has the advantage of letting you set the tab order of buttons, though, which you cannot do with the "NextControl" property.

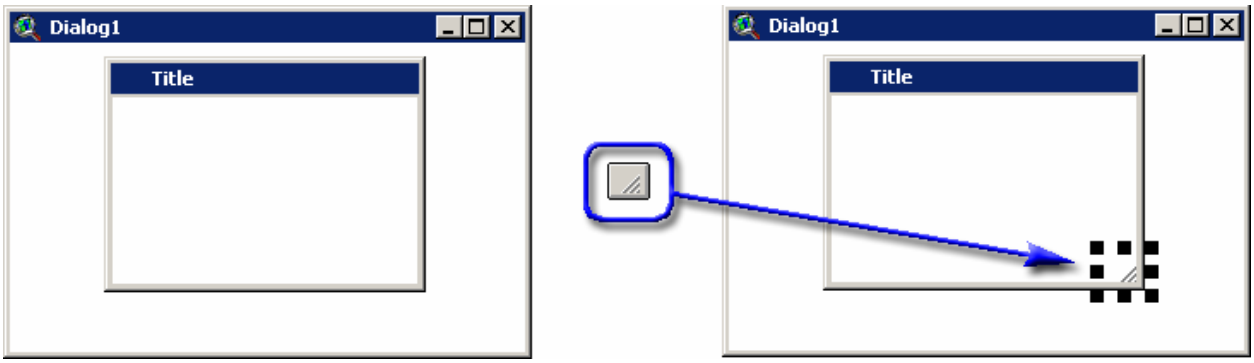
The  button is a kind of cleaning tool for all your dialogs (it's supposed to look like soap bubbles). If you click it, it'll go through all your dialogs and set all possible object tags to "nil" and set the servers to "nil".

The  button saves things into an object database. See the description in "Project Buttons" regarding saving components into an object database.

The  button extracts all the scripts, dialogs, buttons, tools and menu items from the object database and installs them into your current project. It won't install a component that already exists, so you shouldn't get multiple copies of the same script or button.

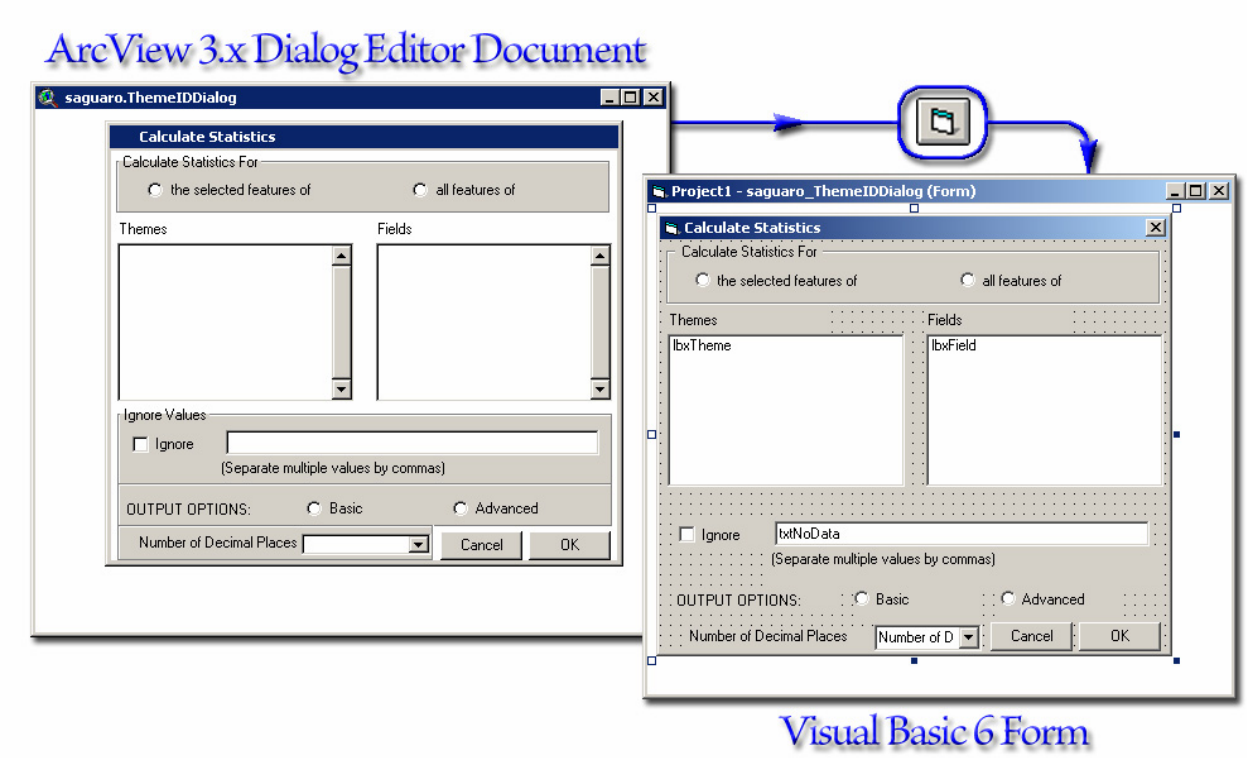
The  button is a global search tool. It searches through all your scripts for your search term and gives you a report of where they occur. See the discussion of Global Searching in the section on Script Buttons.

The  button adds corner bars to the lower right corner of your dialog and sets the dialog to be resizable. These corner bars are a common way to indicate that a dialog may be resized by dragging on a corner. This function also sets the corner bar icon fasteners such that it will maintain a constant width and height, and will maintain a constant distance from the right and bottom edges of the dialog.

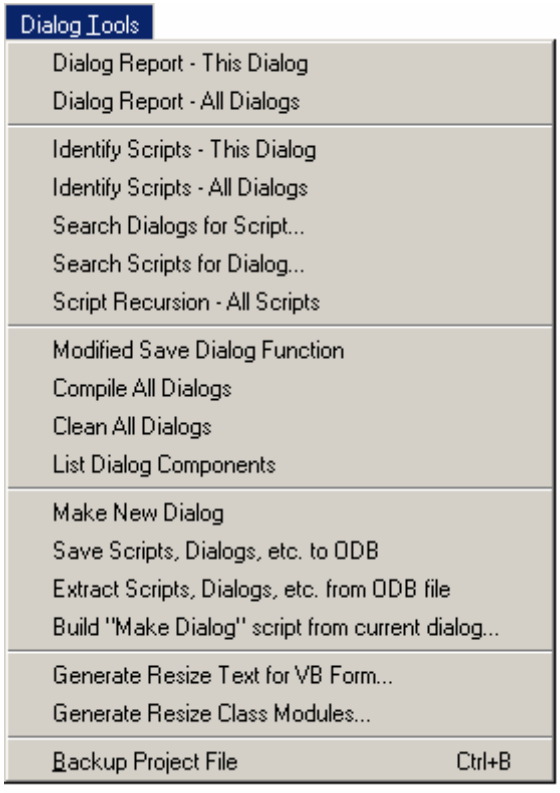


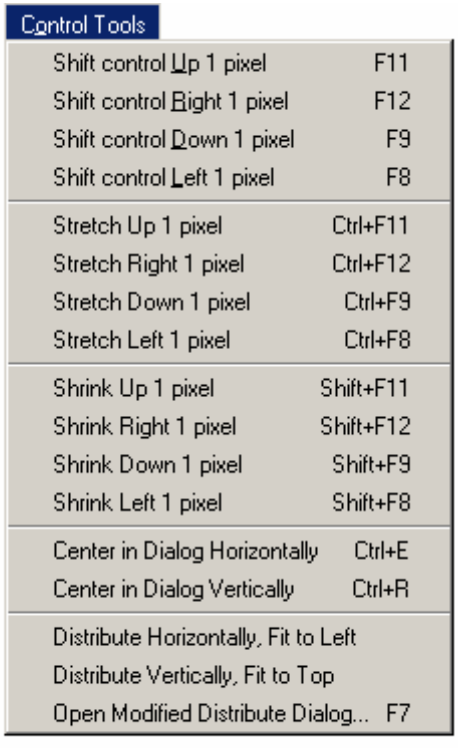
The  button attempts to export the Dialog Editor dialog into a VB6 form. It positions and indexes controls correctly, and includes notes in the "Private Sub Form_Load()" subroutine discussing additional changes you need to do (such as identifying images to attach to buttons or image boxes).

If the ArcView dialog is resizable, then this function will also insert code to attach Resize anchors to each control.

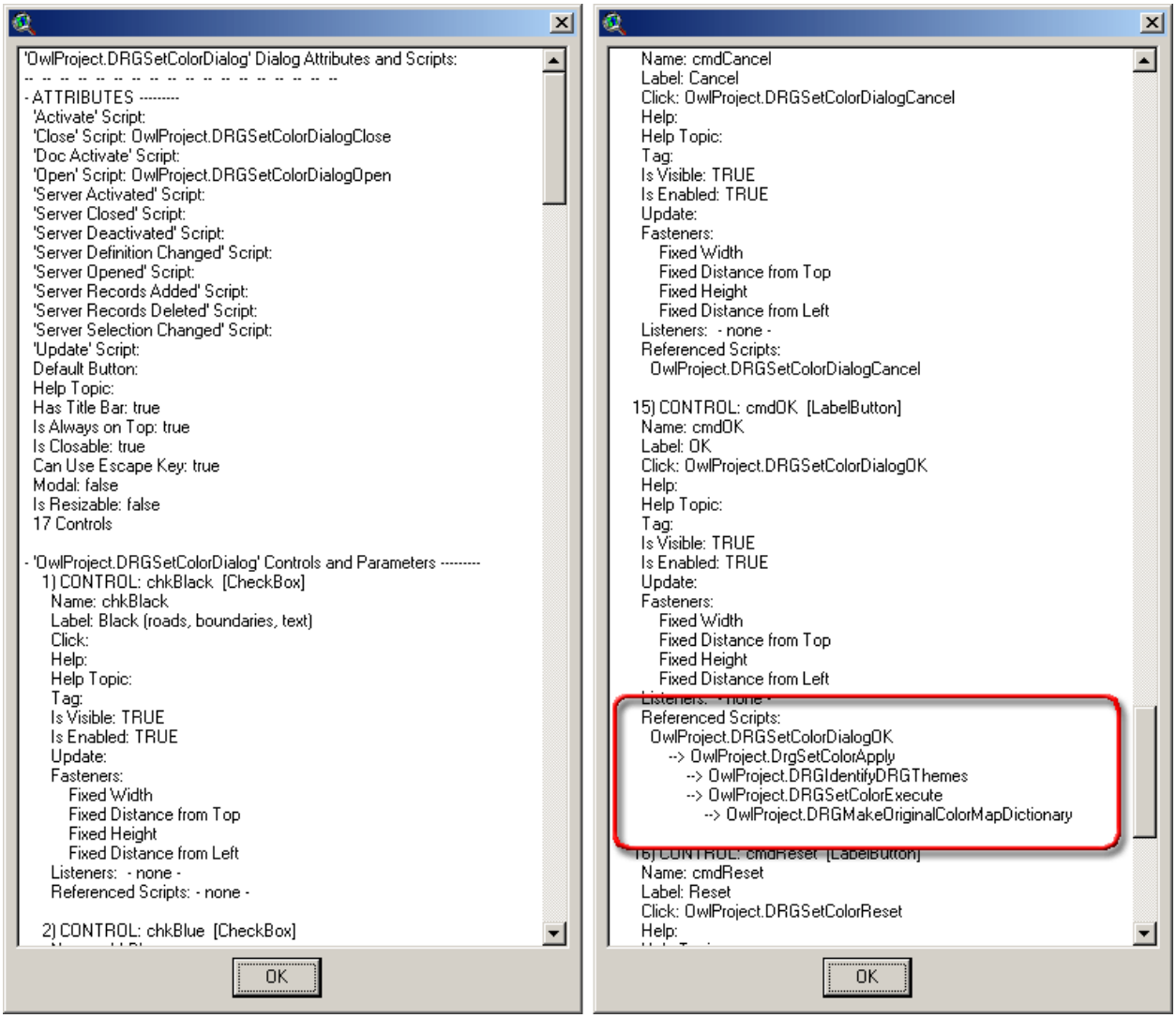


Dialog Menu Items:



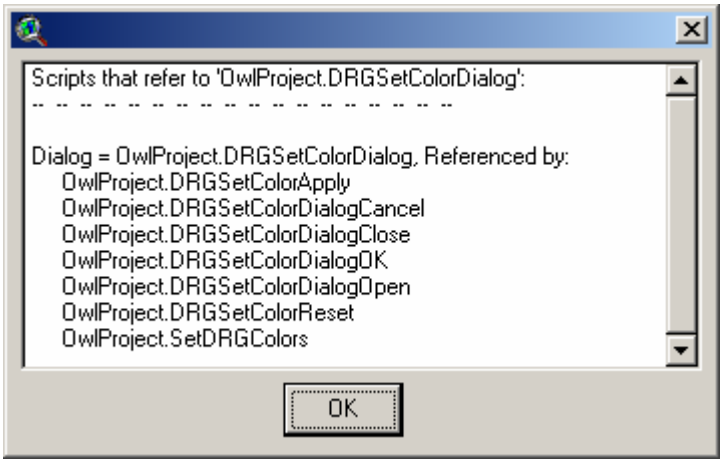


- *Dialog Report - This Dialog:* This function produces a report describing the dialog, all it's parameters and referenced scripts, and all the parameters and referenced scripts of all the controls on the dialog.

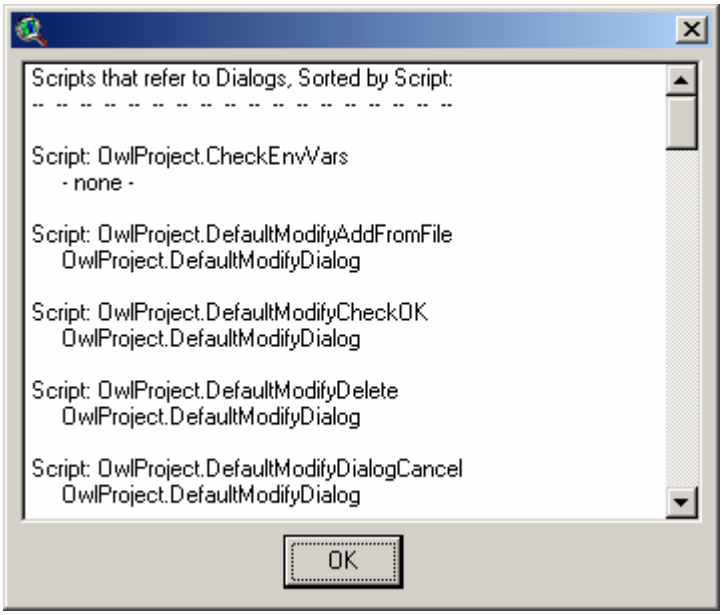


Notice that the list of referenced scripts also shows the additional scripts that each script references.

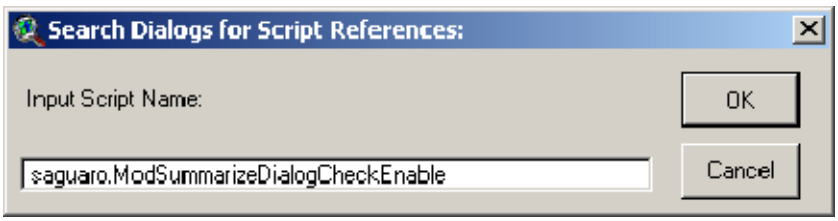
- *Dialog Report - All Dialogs:* This function does the same thing that "*Dialog Report - This Dialog*" does, except that it produces a single report describing all dialogs in the project.
- *Identify Scripts - This Dialog:* This function searches all the scripts in the project and identifies the ones that refer to this dialog:

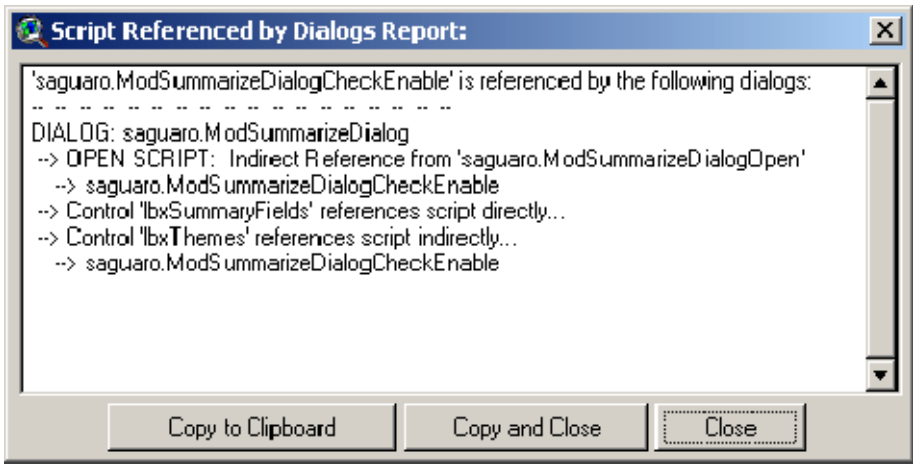


- *Identify Scripts - All Dialogs:* This function searches through all the scripts in the project and identifies which dialog each script refers to:

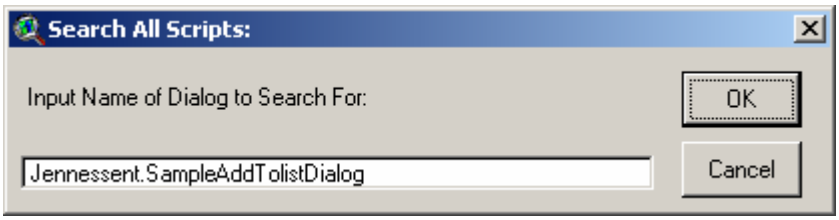


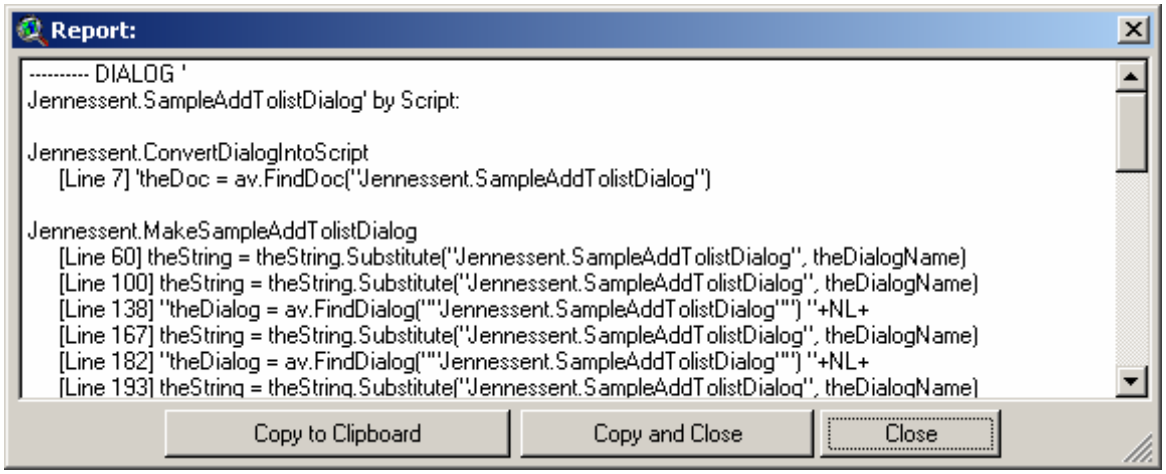
- *Search Dialogs for Script…:* Searches all the standard dialog event scripts for a script name, including all referenced scripts. Also searches all dialog control scripts and all of their referenced scripts.



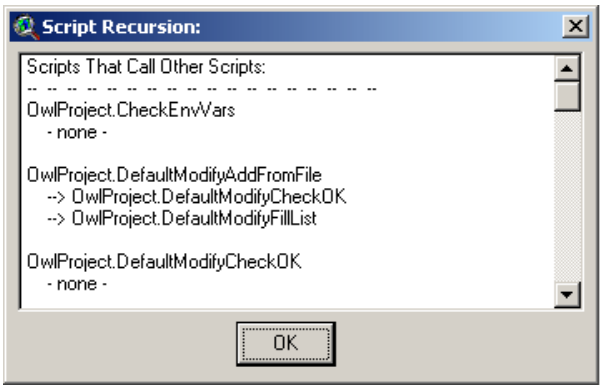


- *Search Dialogs for Script:* Basically does a "Search All Scripts for Text String" function, automatically entering the current Dialog name into the input box:





- *Script Recursion - All Scripts:* This function produces a report describing how the scripts in the project call each other. If a script calls other scripts, those other scripts are shown below the script name and indented 2 spaces:

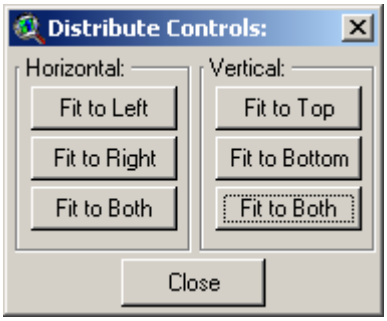


- *Modified Save Dialog Function:* This function is identical to the  button described above.
- *Compile All Dialogs:* This function is identical to the  button described above.
- *Clean All Dialogs:* This function is identical to the  button described above.
- *List Dialog Components:* This function is identical to the  button described above.
- *Make New Dialog:* This function is identical to the  button described above.
- *Save Scripts, Dialogs, etc. to ODB:* This function is identical to the  button described above.
- *Extract Scripts, Dialogs, etc. from ODB file:* This function is identical to the  button described above.
- *Build "Make Dialog" script from current dialog:* Generates the code necessary to create the current dialog on-the-fly.
- *Generate Resize Text for VB Form:* This function takes an existing VB form, analyzes it for resizable controls, and generates the appropriate code to insert into the "Form_Load" sub procedure. See appendix for sample of actual code generated. The resizing depends on 3 class modules, which can be autogenerated with the code below:
- *Generate Resize Class Modules:* This function generates the 3 VB class modules necessary to make the Resize functions work. The code is adapted from original code written by "neophile (n_e_o_p_h_i_l_e@yahoo.com)". See appendix for actual code.
- *Backup Project File:* This saves a copy of your current project file to a new name in the same directory as your current project file. It appends the date and time to the new name. For example if your project was named "this_project.apr" and you clicked the backup function at 10:30:23 on June 20, 2003, this function would save a copy of the current state of your project to "this_project_06202003_103023.apr". This function does not save the current state of your project to "this_project.apr" though! Use [Control]-S for that. This function is repeated in the File menu of all the documents.
- *Control Tools:* The first 14 options provide hot-keys to resize and move control graphics on the Dialog Editor document. The "Shift Control" functions move the controls by 1

pixel rather than 2 pixels like the arrow keys do, making it much easier to manually align controls. The resize functions also modify the controls by 1 pixel in width or height. In general, [F8] moves the control left, [F9] moves it down, [F11] moves it up and [F12] moves it right. The [Control] + [F-key] stretches the graphic control in the specified direction, and the [Shift] + [F-key] shrinks the control.

[Control]-R will shift the selected control graphics to the vertical center of the dialog, and [Control]-E will shift them to the horizontal center. The selected graphics will move as a group such that their positions relative to each other will not change.

- *Control Tools; Distribute Horizontally, Fit to Left:* This is similar to the control distribution function in the standard ArcView "Align" box, except that this function forces the controls to have a constant spacing between each other. The standard "Align" function will distribute the controls as evenly as it can, while locking the position of the two controls on the ends. This function may shift the control on the right side in order to maintain the constant separation distance.
- *Control Tools; Distribute Horizontally, Fit to Top:* This is similar to the control distribution function in the standard ArcView "Align" box, except that this function forces the controls to have a constant spacing between each other. The standard "Align" function will distribute the controls as evenly as it can, while locking the position of the two controls on the top and bottom. This function may shift the control on the bottom in order to maintain the constant separation distance.
- *Control Tools; Open Modified Distribute Dialog:* This gives you more control over how distributed controls are separated. The two options above describe 2 of the functions available on this modified Distribution dialog.



Script Buttons:

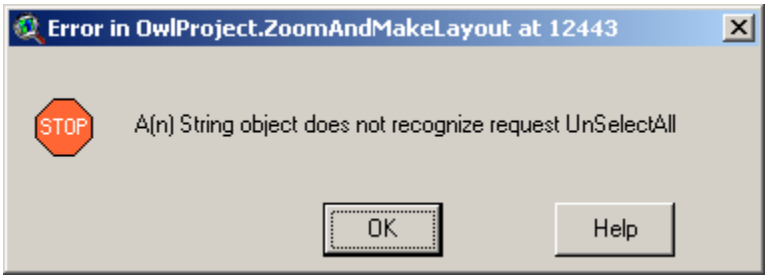
This extension also adds 10 buttons to the Script button bar:

The button inserts several lines of code into your script. This is intended for scripts called from Views and collects a set of commonly-used variables. I just got tired of typing this out every time so I made a tool to automate it.

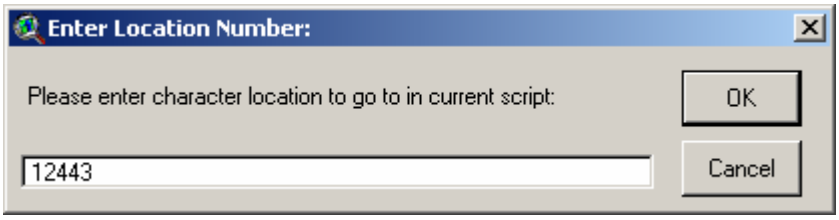
```
theView = av.GetActiveDoc
theDisplay = theView.GetDisplay
theThemes = theView.GetThemes
theGraphics = theView.GetGraphics
theProject = av.GetProject
thePrj = theView.GetProjection
```

```
theWorkDir = av.GetProject.GetWorkDir
theWorkDirStr = theWorkDir.AsString
theOS = System.GetOS

theFThemes = {}
for each aTheme in theThemes
  if (aTheme.Is(FTheme)) then
    theClassName = aTheme.GetFTab.GetShapeClass.GetClassName
    if (theClassName = "Polygon") then theFThemes.Add(aTheme) end
  end
end
```

The  button takes you to a specified location in the script. For example, if you get an error message looking something like:



then you know that the script crashed at character location 12,443,or 12,443 characters into the script. By the way, this kind of error message usually happens when somebody else is using your compiled extension; when you cause the crash yourself in the project where you're writing the code, ArcView just takes you directly to the location that triggered the crash (with a few exceptions; see "Insert Error Checking Code" on p. 32 for help in tracking down such hard-to-locate crashes as the "AVArray:" bug). You would use this tool to identify the script location indicated by the error message. When you click the tool, you'll be prompted to enter the location. The tool then will position your cursor at that location.

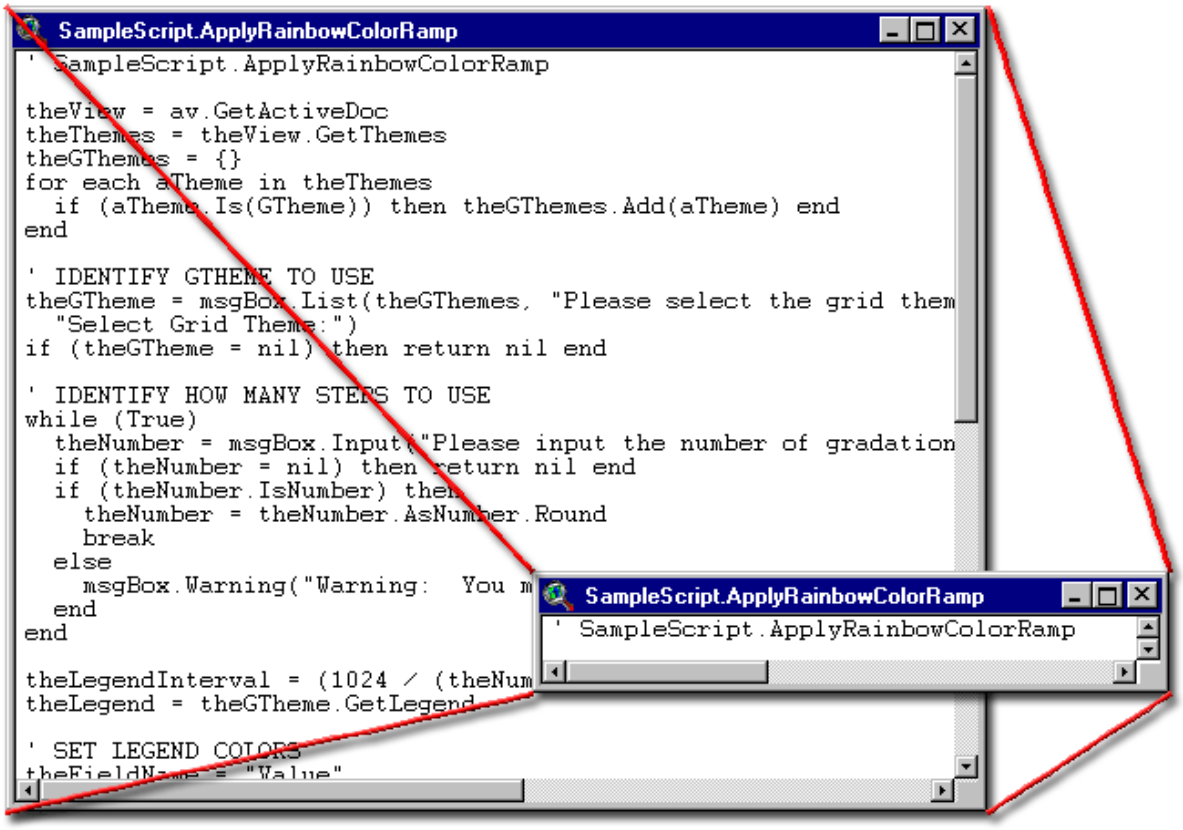


The  button makes new dialogs. See the description above regarding making new dialogs.

The  button compiles all scripts, and gives you a report listing all scripts that couldn't be compiled.

The  button closes all open scripts.

The  button shrinks the current script to the minimum "open" height, allowing you to minimize the script while still keeping it large enough to read the name:

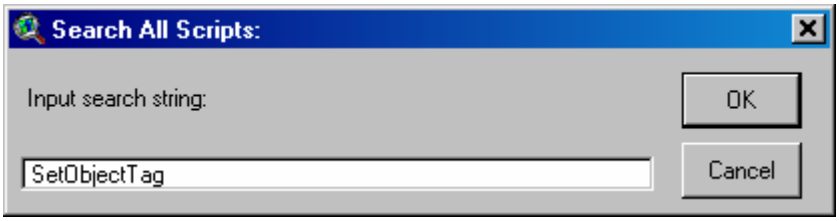


The  button shrinks all open scripts in the project.

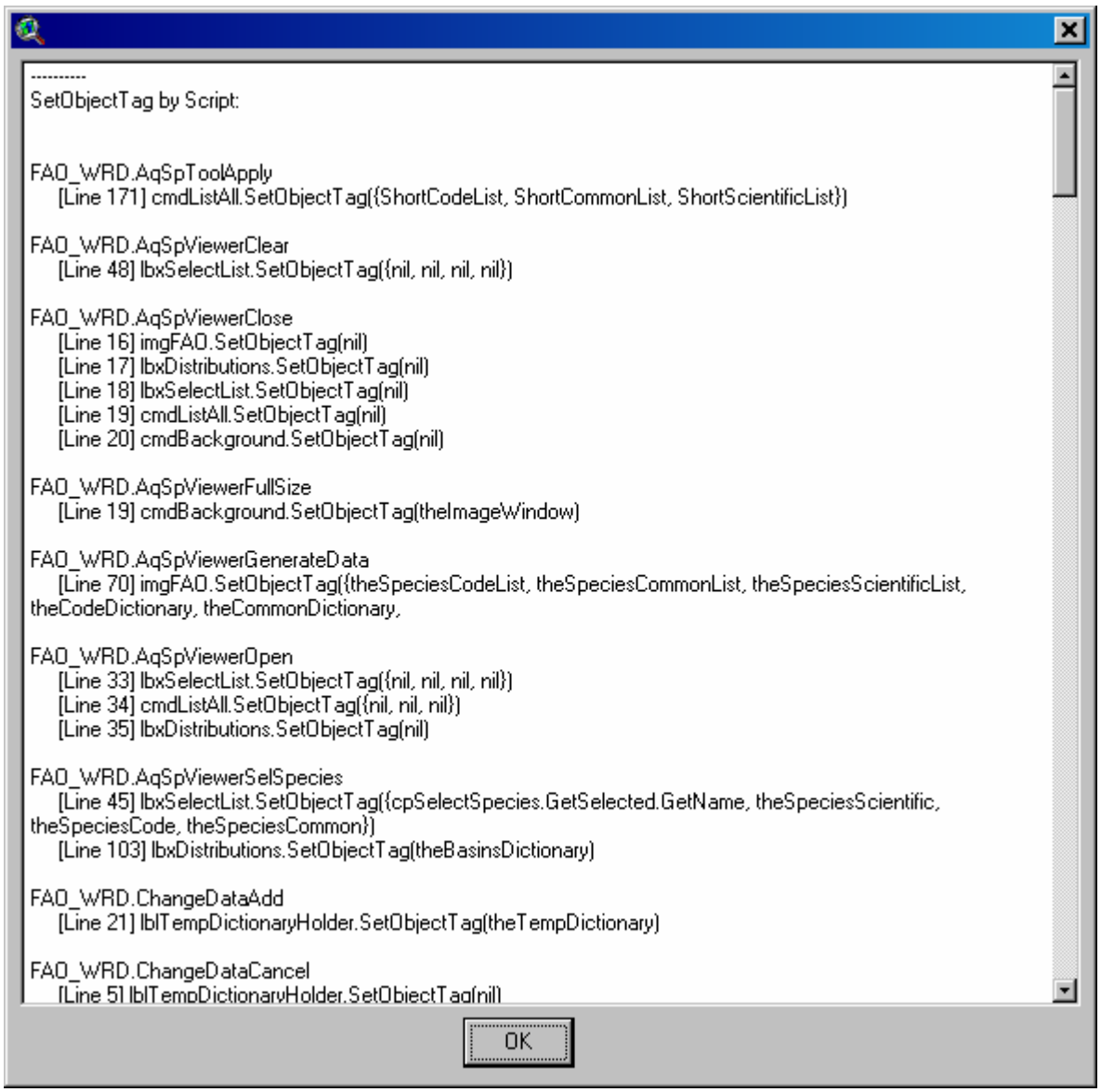
The  button saves things into an object database. See the description in "Project Buttons" regarding saving components into an object database.

The  button extracts all the scripts, dialogs, buttons, tools and menu items from the object database and installs them into your current project. It won't install a component that already exists, so you shouldn't get multiple copies of the same script or button.

The  button is a global search tool. It searches through all your scripts for your search term and gives you a report of where they occur. Click the button and you'll see the following:

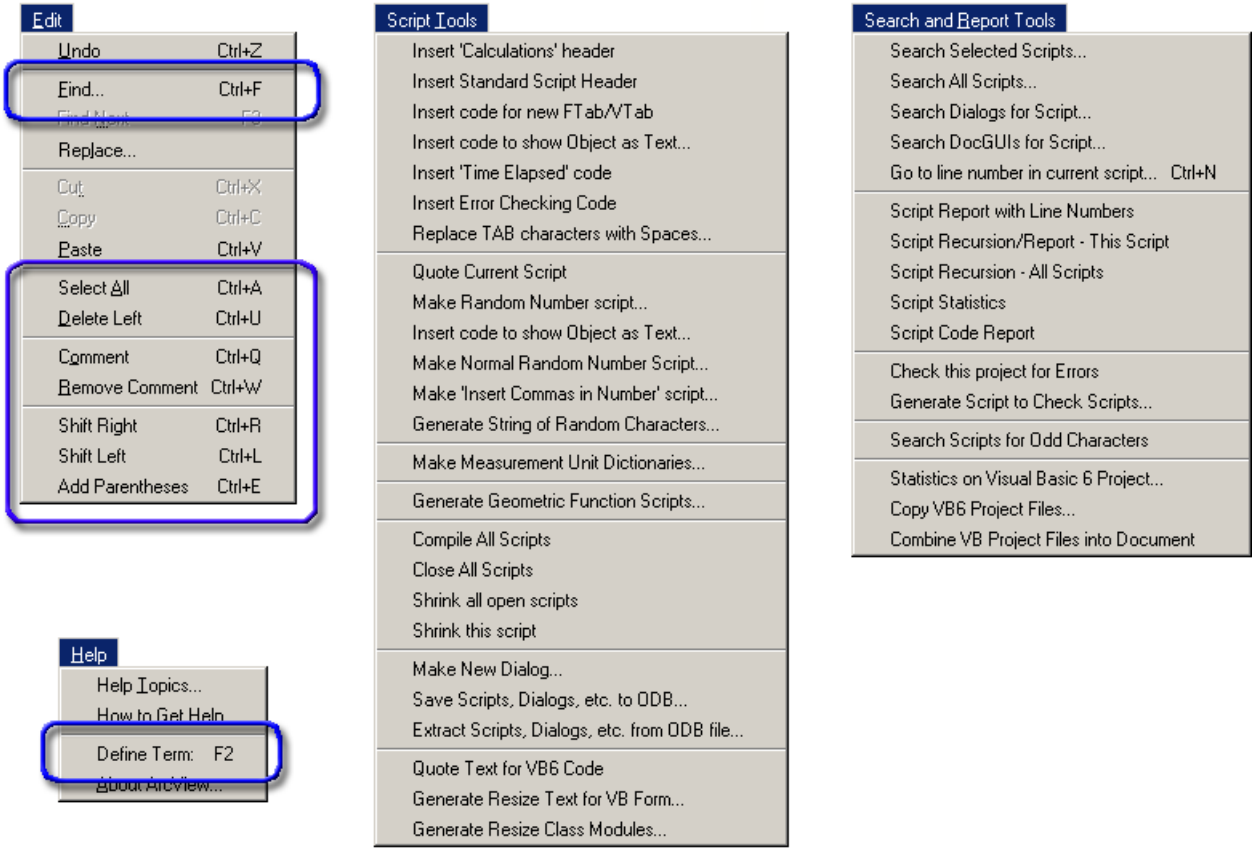


Doing a quick search for the term "SetObjectTag" gives me the following:



You can also do Pattern-style search by using the "*" and "&" wildcard characters. "*" stands for any string of indeterminate length while "&" stands for any character.

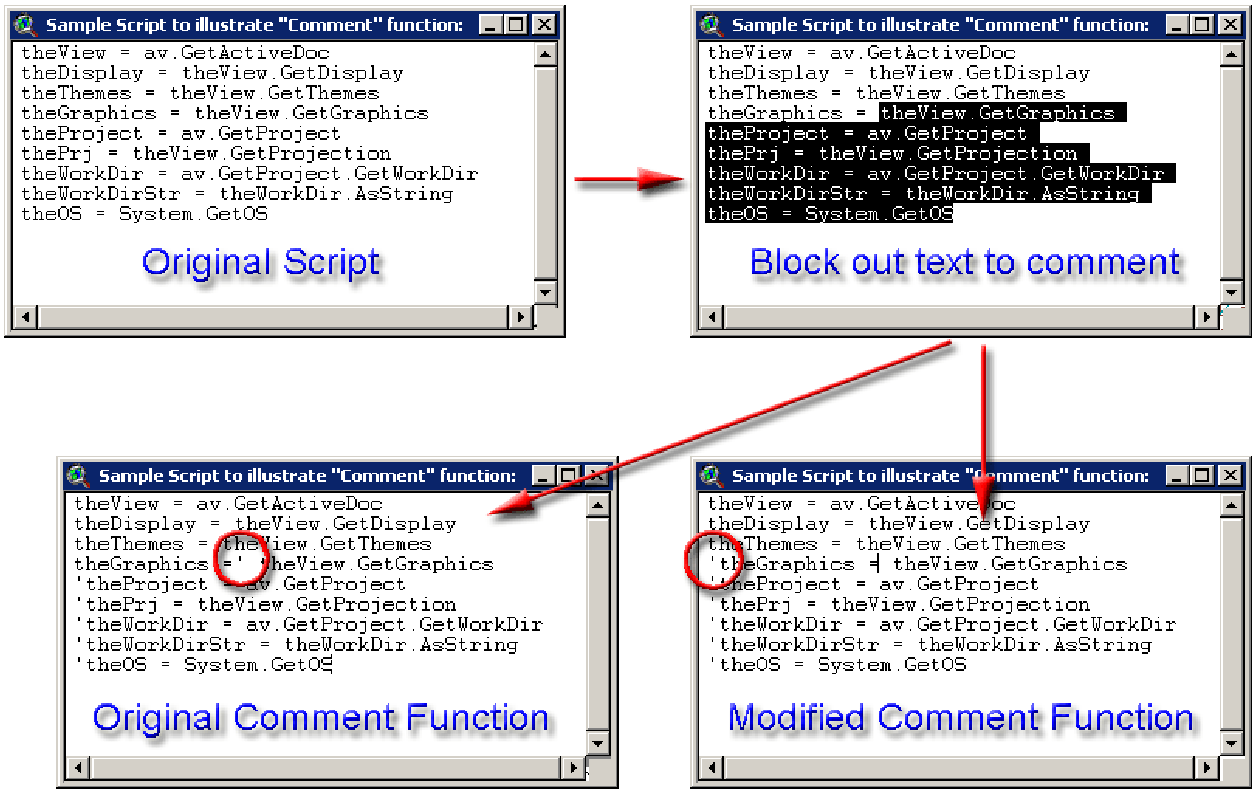
Script Menu Items:



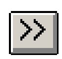
Edit Menu:

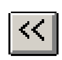
- *Find:* I added a shortcut key to this option so you can simply click [Control]-F to find text in this script. An easier way to use it is to block the text to look for first. If any text is selected, then the function searches for instances of that selected text string.
- *Select All:* I added a shortcut key to this option so you can select all the text in the script by clicking [Control]-A.
- *Comment:* I changed the "Comment" script so that it would behave differently. Previously, if you blocked out a section of text but started the block in the middle of the line, the "Comment" function would insert the comment symbol in the middle of the line. I modified it to put the comment symbol at the beginning of the line no matter where you start the comment block. I also modified the update script so that it would always be enabled. The update doesn't get fired off when you make a selection so often the comment function will be disabled when you want to use it.

  I also added a shortcut key so you could trigger the comment function by clicking [Control]-Q.



- *Remove Comment:* As with the comment function, I modified the update script so this function would always be enabled. I also added a shortcut key so you could trigger the remove comment function by clicking [Control]-W.

- *Shift Right:* This does exactly the same thing that the  button does. The advantage to making it a menu item is that we can assign keyboard shortcuts to it. Now you can shift the text to the right by using [Control]-R.

- *Shift Left:* This does exactly the same thing that the  button does. The advantage to making it a menu item is that we can assign keyboard shortcuts to it. Now you can shift the text to the right by using [Control]-L.

- *Add Parentheses:* This adds open- and close- parentheses around the selected text.

Help Menu:

- *Define Term:* This does the same thing as the  button, but a little more conveniently with a keyboard shortcut. Select the term you are interested in and click the F2 button to open up the help files for that term.

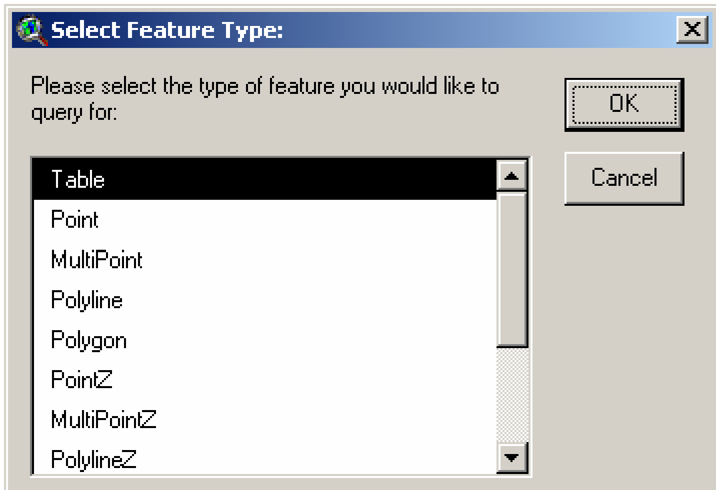
Script Tools Menu:

- *Insert 'Calculations' header:* This just inserts a clear commented out message indicating that the calculation portion of the script is about to begin. This type of break helps when you have a long script that takes a lot of code to gather the parameters for the calculation.

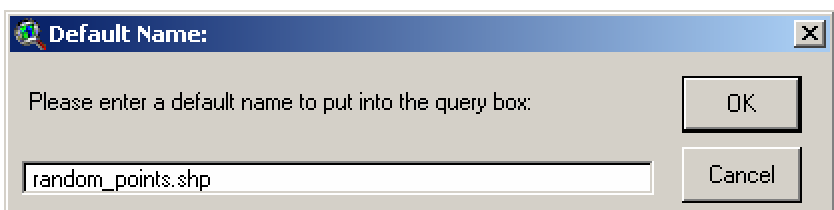
```
'~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
'////////// |                  | \\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\
'---------- |   CALCULATIONS   |  ----------------------------------------------------------------------
'\\\\\\\\\\ |                  | //////////////////////////////////////////////////////////////////////////
```

```
'~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
```

- *Insert Standard Script Header:* This does the exact function as the  button described above.
- *Insert code for new FTab/VTab:* This function generates the necessary code to query the user for the name of a new shapefile or dbf table. It customizes the code to generate the precise shape class you're interested in based on the feature type you select:



It then asks you to enter in a default name, which will be suggested to the user when the FileDialog opens for them:



It then generates the proper code to do the following (see Appendix for sample code):

1) Search the current working directory for an existing file by that name, and add a number to the end of it if such a file exists. This is exactly the same function as the "Filename.MakeTmp" request except that "MakeTmp" only allows you a maximum of 6 characters for the filename. This new version will replace "random_points.shp" with "random_points1.shp" or "random_points2.shp" if the file already exists.
2) It opens a filedialog prompting the user to either accept the suggested default name or enter a name of their own.
3) It generates the code that will create the new VTab or FTab, identifies the Shape field if it's an FTab, adds a numeric ID field named "ID", and adds code to calculate record numbers in the ID Field and stop editing.

- *Insert Code to Show Object as Text…:* This function adds a code snippet that will take an object and store it in an Object Database, then read that ODB in a report window and delete the ODB. This enables you to see an object as ArcView does, which can be useful sometimes if you need to create one from scratch or to look for problems with an object.

- *Insert 'Time Elapsed' code:* This adds a code snippet that produces a short report string describing the amount of time elapsed. Formats beginning and ending times and parses out time elapsed as "X hours, X minutes, X seconds…". The exact code that is inserted is as follows:

```
' ASSUMES THE VARIABLES BeginTime AND theReport HAVE ALREADY BEEN DEFINED

theElapsedTime = (Date.Now - BeginTime).AsSeconds
theNumHours = (theElapsedTime/3600).Truncate
theNumMinutes = ((theElapsedTime.Mod(3600))/60).Truncate
theNumSeconds = theElapsedTime.Mod(60)

theElapsedTimeString = "Time Elapsed: "
if (theNumHours > 0) then
  theElapsedTimeString = theElapsedTimeString+theNumHours.AsString+" hours, "+
  theNumMinutes.AsString+" minutes, "+theNumSeconds.AsString+" seconds..."
elseif (theNumMinutes > 0) then
  theElapsedTimeString = theElapsedTimeString+
  theNumMinutes.AsString+" minutes, "+theNumSeconds.AsString+" seconds..."
else
  theElapsedTimeString = theElapsedTimeString+theNumSeconds.AsString+" seconds..."
end

theReport = theReport+
     "Analysis Began: "+BeginTime.SetFormat("MMMM d, h:m:s AMPM").AsString+NL+
     "Analysis Complete: "+Date.Now.SetFormat("MMMM d, h:m:s AMPM").AsString+NL+
      theElapsedTimeString+NL+NL
```

- *Insert Error Checking Code:* This is a very handy function for identifying problems in scripts that are called by other scripts. ArcView reports the cause and location of a crash in the current script that is running, but not causes and locations in called scripts. Therefore this function will check to see if a script returns a "nil" value and, if so, will give you a report describing any errors found in that script. When a called script crashes, it returns a "nil" value to the calling script. ArcView in general will not crash unless that nil value causes the calling script to crash. This function will add the following code at the cursor location of the currently open script:

```
if (theResponse = Nil) then
  theScript = myScript
  msgBox.Report("Error Message: "+theScript.GetErrorMsg+NL++NL+
      "Error Position: "+(theScript.GetErrorPos.AsString),
      "'"+theScript.GetName+"' crashed:")
  return nil
end
```

  You need to substitute the text "myScript" with the script that is being called.

  This code snippet can also be modified to help you track down bugs that do not normally tell you where the problem occurred (such as the "AVArray: Index ___ not in range ___" message). As soon as the script crashes, and you are wondering where in the script the crash occurred, modify the code snippet as follows and run it in a separate script:

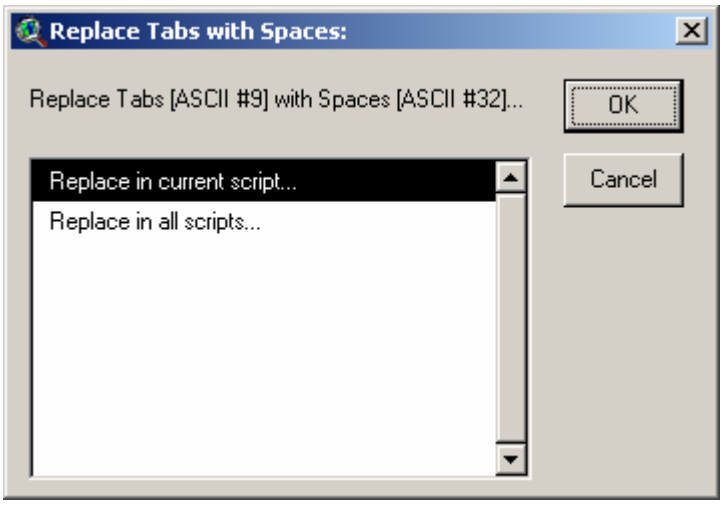
```
theScript = TheScriptThatCrashed
msgBox.Report("Error Message: "+theScript.GetErrorMsg+NL++NL+
    "Error Position: "+(theScript.GetErrorPos.AsString),
    "'"+theScript.GetName+"' crashed:")
```

  This snippet will produce a report telling you the character number where the crash occurred, and you can use the  button to take you to that location.

- *Replace TAB characters with Spaces:* In some cases scripts may have spaces which are actually TAB characters. This won't happen if you type the script up in ArcView, but it

might happen if you copy-and-paste from other documents. I am not aware if there is any problem with having TAB characters in the script, but I have run into some unexpected problems with Asian installations of ArcView and I suspect they are due to characters that are not translated correctly. Therefore I wrote this function to convert TAB characters to Spaces (basically doing a substitute operation, replacing "9.AsChar" with spaces.

You will be asked whether you want to replace TABs in only the current script or in all scripts:

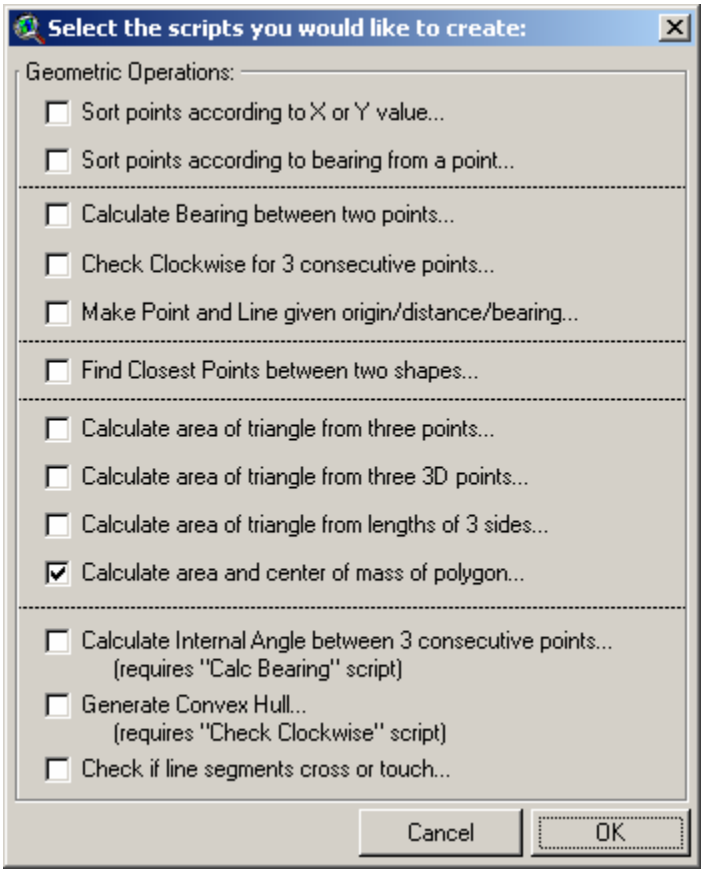


- *Quote Current Script:* Converts the current script to a string by inserting quote marks and "NL" in appropriate places. Generates a new compiled SEd document named "[Your Script Name]_Quoted" and containing the text:

```
theString =
    <theQuotedScript>
```

- *Make Random Number script…:* This function produces a script that takes advantage of some code by Bill Huber to more accurately generate random numbers. You send the script a minimum value, a maximum value and a desired number of decimal places, and this script generates a random number within that range. See the sample code for details on how this works.
- *Make Normal Random Number script…:* This function produces a script that generates 2 normally distributed random numbers based on a specified mean and standard deviation, using the Box-Muller transformation.
- *Make 'Insert Commas in Number' script…:* This function produces a script that takes a number in either string or numeric format (i.e. "123456789.012") and inserts commas into it (i.e. "123,456,789.012". See the sample code for details on how this works.
- *Generate String of Random Characters:* This function is intended to generate a character string to stand in as a placeholder, and will likely be substituted with other text later. For example, if you make a report of the analyses performed by a script, you might make a rough template of the report. You might then generate sub-reports throughout the running of the script, and then you can substitute the placeholder random text with the sub-report.

- *Make Measurement Unit Dictionaries…:* This function makes a script that returns two dictionaries of measurement units. The first dictionary sets the enumerations (i.e. "#UNITS_LINEAR_METERS") as the keys and the unit names (i.e. "Meters") as the elements. The second dictionary sets the unit names as the keys and the enumerations as the elements. See the sample code for the actual script.
- *Generate Geometric Function Scripts…:* This function generates a variety of geometric functions that I regularly incorporate into my extensions. These scripts are described briefly below. See Appendix for samples of actual scripts.



1) *Sort points according to X or Y value:* Given a list of points, this script will sort the points according to either the point X- or Y-coordinates, in ascending or descending order. It returns a list of sorted X- or Y-coordinates and a dictionary of points. The dictionary keys are the X- or Y-coordinate values and the dictionary elements are lists of points at that coordinate, sorted by the other coordinate.

2) *Sort points according to bearing from a point:* Given a single point and a list of points, this script will sort the list of points according to the bearing of that point from the single point. It returns a list of sorted bearing values and a dictionary of points. The dictionary keys are the bearing values and the dictionary elements are lists of points at that bearing, sorted in ascending order according to the distance from the single point.

3) *Calculate Bearing script:* Given two consecutive points, this script will return the compass bearing from the first point to the second point.

4) *Check Clockwise script:* Given three consecutive points, this script will return a Boolean (True/False) value reflecting where the third point lies in relation to a line extending from the first point to the second point. Returns "True" if the third point is to the right of the line (clockwise), and "False" if the third point lies to the left of the line (counterclockwise).

5) *Find Closest Points script:* Given two shapes (points, lines or polygons), this script returns a line object connecting the closest point on the first shape to the closest point on the second shape. You can use the start/end points of this line to identify the actual closest points on the respective shape and to identify the bearing between these closest points.

6) *Make Point and Line script:* Given an origin point, a distance and a bearing, this script returns a new point at the specified distance/bearing and a line connecting the origin to that new point.

7) *Triangle Area from Points script:* Given three points on a horizontal plane (as all non-PointZ points are), this script returns the area of the triangle formed by those three points.

8) *Triangle Area from 3D Points script:* Given three PointZ shapes, this script returns the area of the 3-dimensional triangle formed by those three points.

9) *Triangle Area from Sides script:* Given the lengths of three sides of a triangle, this scripts returns the area of the triangle bounded by those three sides. It returns a null value if those three lengths cannot form a triangle.

10) *Calculate area and center of mass of polygon:* Produces the true center of mass (centroid) of a polygon, which is different than the ESRI centroid. Also produces the area of the polygon, which is the same as the ESRI-derived area.

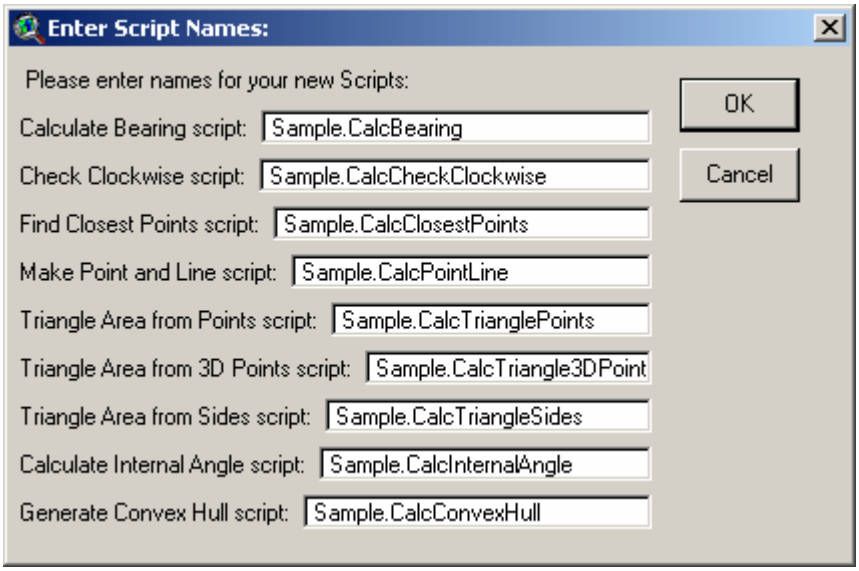
11) *Calculate Internal Angle script:* Given three consecutive points, this script returns the internal angle formed by the line connecting the first and second points and the line connecting the second and third points. This script also returns the angle of deviation, reflecting how much the bearing of the second line deviates from the bearing of the first line.

12) *Generate Convex Hull script:* Given a list of points, this script returns a convex hull polygon around the outermost points.

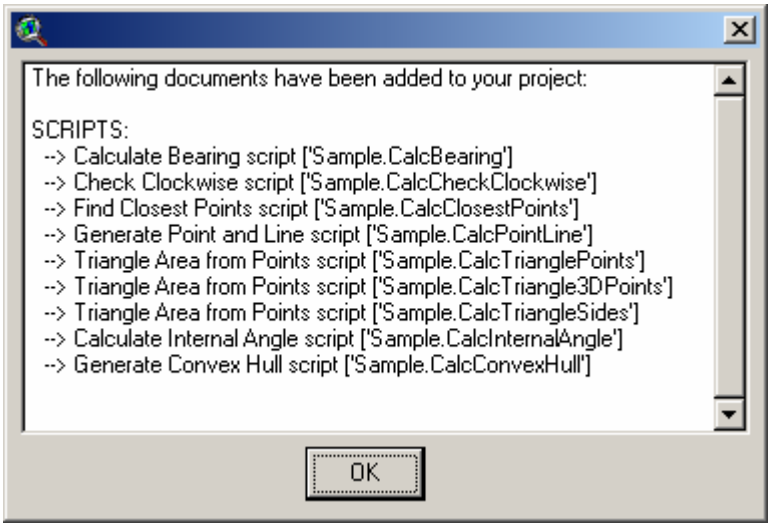
13) *Check of line segments cross or touch:* This checks two line segments to see if they overlap, touch, or are separate. The difference between this function and the basic Avenue "Shape.Intersects" request is that this script also checks to see if the lines just

touch each other vs. actually crossing over each other. To make it run slightly faster in most of my applications, I wrote the script such that you need to send it a list of 4 points rather than a list of two line segments. The 1st and 2nd points need to be the start and end points of the first line segment, and the 3rd and 4th points need to be the start/end points of the second line segment. The script then returns a 0 if the lines intersect, a 1 if the lines just touch each other, and a 2 if the lines do not intersect.

Select the scripts you're interested in and click "OK". You will then be prompted to give names for these scripts. Note that the Internal Angle script requires the "Calc Bearing" script, and the "Convex Hull" script requires the "Check Clockwise" script. The names you assign to the "Check Clockwise" and "Calc Bearing" scripts will be correctly written into the "Internal Angle" and "Convex Hull" scripts:



After you have identified the names and clicked "OK", the scripts are generated and you will see a report of the scripts that were added to your project:



- *Compile All Scripts:* This does exactly the same thing as the  button described above.

- *Close All Scripts:* This does exactly the same thing as the  button described above.
- *Shrink all open scripts:* This does exactly the same thing as the  button described above.
- *Shrink this script:* This does exactly the same thing as the  button described above.
- *Make New Dialog…:* This does exactly the same thing as the  button described above.
- *Save Scripts, Dialogs, etc. to ODB…:* This does exactly the same thing as the  button described above.
- *Extract Scripts, Dialogs, etc. from ODB file…:* This does exactly the same thing as the  button described above.
- *Quote Text for VB6 Code:* This substitutes the text in an existing script window with a quoted version of that text, in such a way that VB can read it. Because VB6 limits the number of line continuation characters you can use in a single script, this function sets each line of text as a new line of code (i.e. no line continuation characters). For example:

```
dim lngIndex as long
dim lngSum as long
for lngIndex = 1 to 10
  lngSum = lngSum + lngIndex
  msgbox "Hello!"
next lngIndex
```

  would be converted to the following:

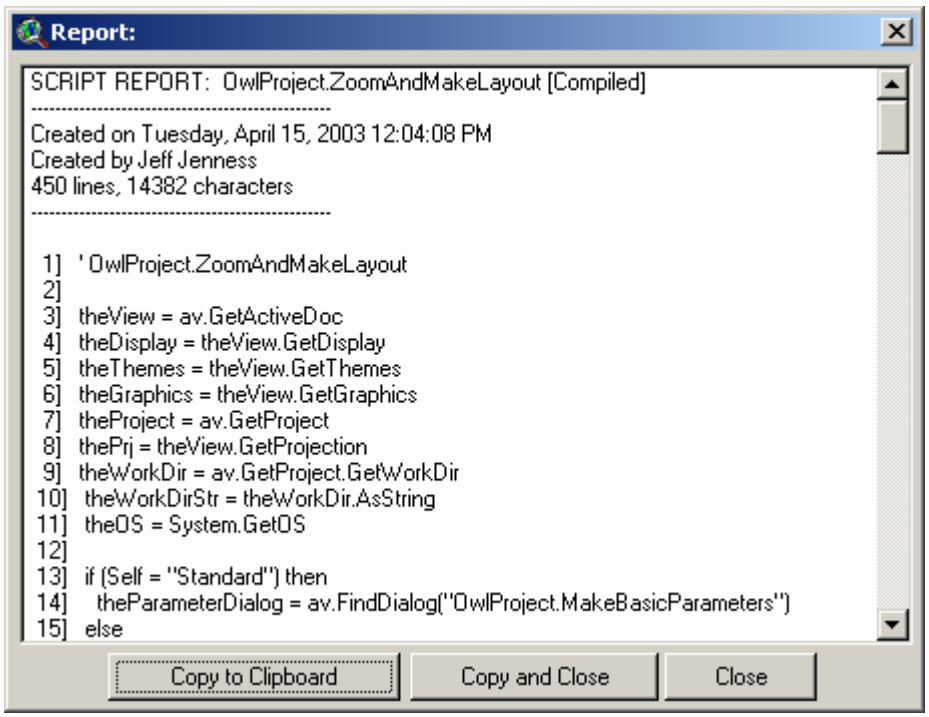
```
dim strBaseString as String
  strBaseString = ""
  strBaseString = strBaseString & "dim lngIndex as long" & vbNewLine
  strBaseString = strBaseString & "dim lngSum as long" & vbNewLine
  strBaseString = strBaseString & "for lngIndex = 1 to 10" & vbNewLine
  strBaseString = strBaseString & "   lngSum = lngSum + lngIndex" & vbNewLine
  strBaseString = strBaseString & "   msgbox ""Hello!""" & vbNewLine
  strBaseString = strBaseString & "next lngIndex" & vbNewLine
```

- *Generate Resize Text for VB Form:* This function takes an existing VB form, analyzes it for resizable controls, and generates the appropriate code to insert into the "Form_Load" subprocedure. See appendix for sample of actual code generated. The resizing depends on 3 class modules, which can be autogenerated with the code below:
- *Generate Resize Class Modules:* This function generates the 3 VB class modules necessary to make the Resize functions work. The code is adapted from original code written by "neophile (n_e_o_p_h_i_l_e@yahoo.com)". See appendix for actual code.

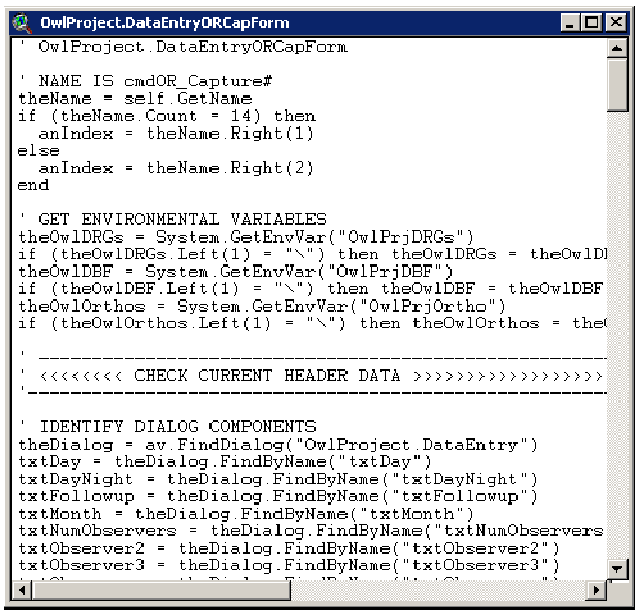
Search and Report Tools Menu:

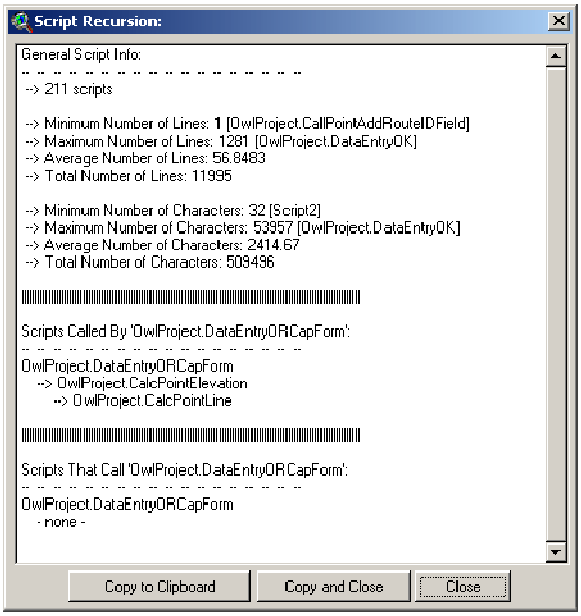
- *Search Selected Scripts:* Very similar to the Global Search Tool activated by the  button (see above) except that it only searches selected scripts instead of all scripts.
- *Search All Scripts…:* This does the same function as the  button (see above).
- *Search Dialogs for Script…:* Searches all the standard dialog event scripts for a script name, including all referenced scripts. Also searches all dialog control scripts and all of their referenced scripts.

- *Search DocGUIs for script:* Searches all DocGUI controls (menus, menu items, buttons, tools and tool menus) for a script name, and produces a report listing all controls that reference that script.
- *Go to line number in current script…:* Inserts the cursor at the beginning of the specified line number.
- *Script Report with Line Numbers…:* This gives you a report with the script name, compilation status, creation data and creator. It also produces a version of the script with line numbers appended to the beginning of each line. This text version can't be compiled because of the line numbers, but it can easily be copied and pasted into an empty script or word processing document to serve as a reference:

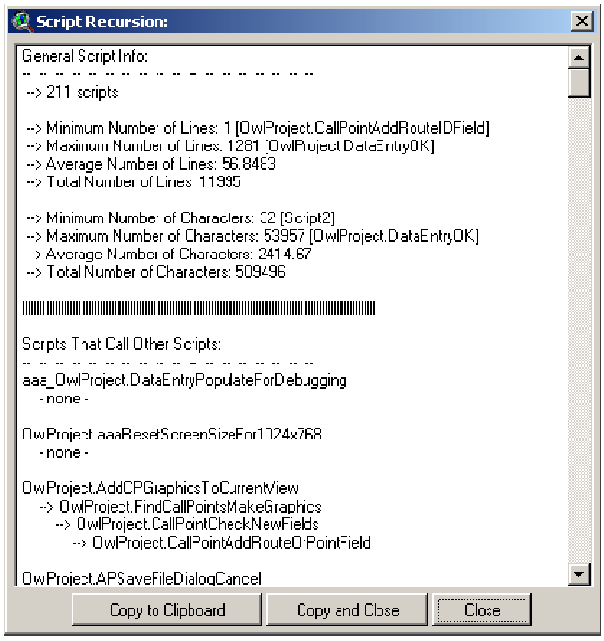


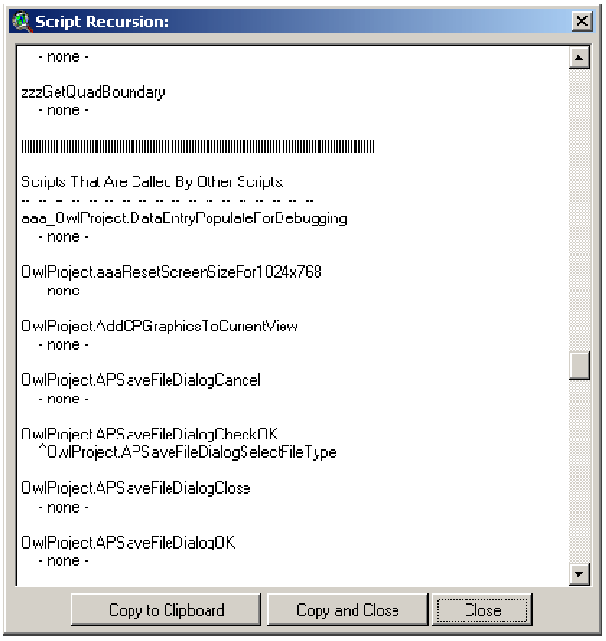
- *Script Recursion/Report - This Script:* This gives you a report of general script statistics, a report identifying all the scripts that this particular script references, plus all the scripts that those scripts reference, etc: It also searches all scripts to find which scripts call this particular script, and all DocGUI controls to see if any controls call this script.



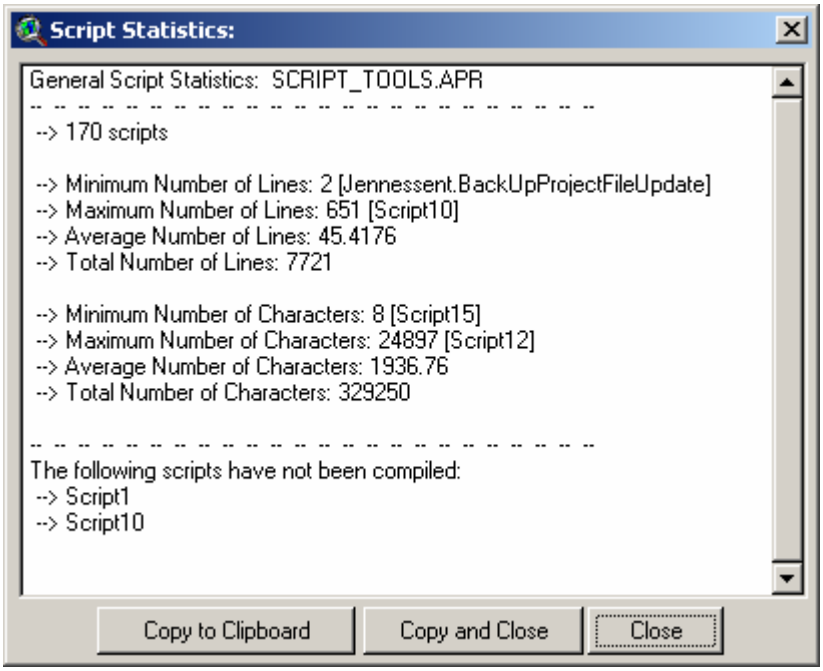


- *Script Recursion - All Scripts:* This function gives you a report of general script statistics and a report describing how the scripts in the project call each other. If a script calls other scripts, those other scripts are shown below the script name and indented 2 spaces:





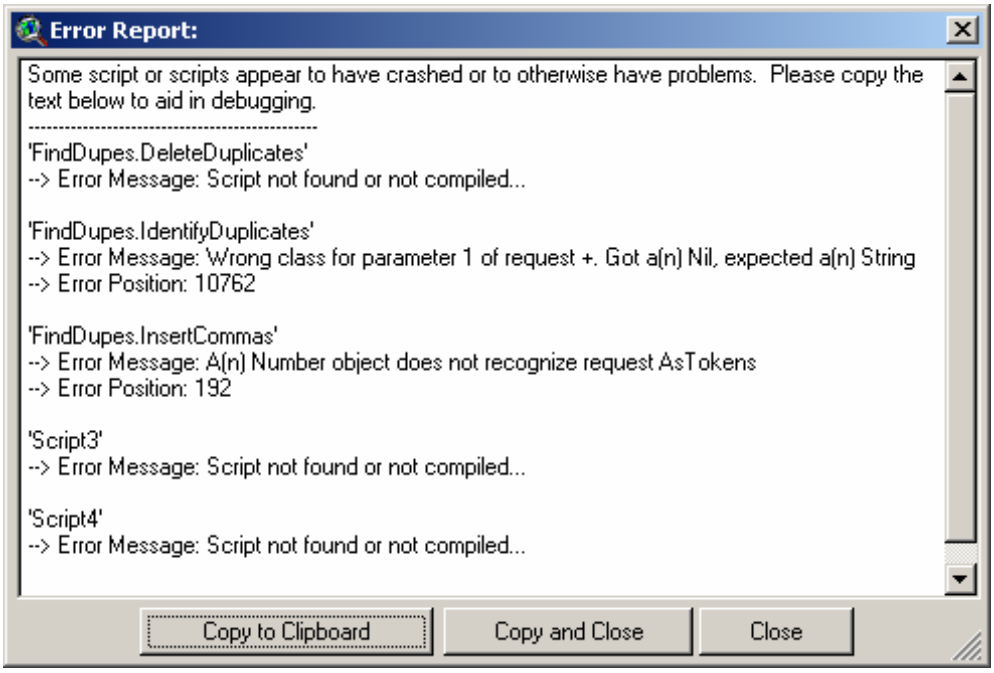
- *General Script Statistics:* This produces a quick report of the minimum, maximum, average and total numbers of lines and characters of code in your project, plus a list of the currently uncompiled scripts:
- *Script Code Report:* This generates a single text string of multiple scripts, allowing you to paste the actual script code into a word-processing document. The function first asks you which scripts to use, then puts all the scripts into a single report window with a "Copy to Clipboard" option. This function also allows you to flag script names with some specified string, to allow word-processing macros to find them.



- *Check this project for Errors:* When a script crashes, ArcView generally knows where in the script the crash occurred. It sometimes shows you the location of the crash (when the extension is compiled) or takes you directly to the location (if you are working in the project with the original scripts), but not always. In particular, the "AVArray" error message does not take you to the location or tell you where it is, even though ArcView does know the location.

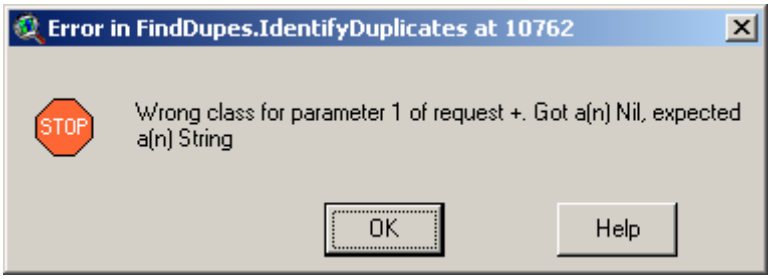
  Furthermore, if you have many interconnected scripts in your project, you may not know which script contained the original error. Script "A" might have crashed, resulting in a "nil" value being returned to Script "B", which might in turn cause Script "B" to crash. ArcView may tell you that Script "B" crashed when you really need to know that the problem occurred in Script "A".

  This function should be run immediately following a general crash. As soon as the overall crash occurs, all scripts have information attached to them stating whether an error occurred in them. This function examines all your scripts and tells you if any of them have such an error (plus the location), as well as if any scripts are uncompiled.



- *Generate Script to Check Scripts:* This does a similar function to that described above, but is intended more to help identify problems on a remote computer where somebody else is using your compiled extension. If someone reports some obscure error message to you and you are unable to determine the cause or location based on the user's description (such as an "AVArray" message), then use this function to generate a script for that user to run on their computer. This script will examine all the scripts that you specify.

  For example, if a user reported an error to you such as the following:



  You may be able to find the error easily enough by simply going to character number 10762 in the script "FindDupes.IdentifyDuplicates". However, this script may have crashed because of problems in other scripts, and then the bug hunt becomes much harder.

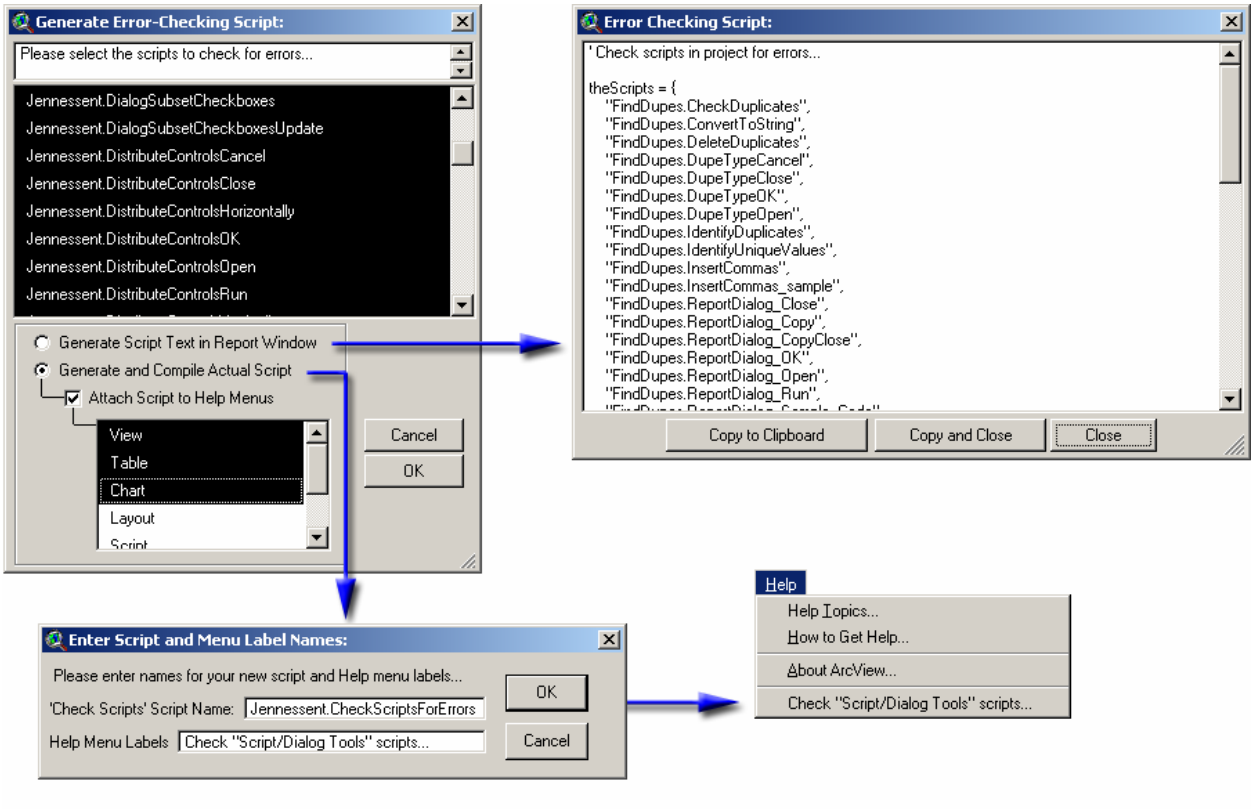
  This function allows you to search for problems in 2 different ways:

  1) You can generate a quick script that will search for errors, and you can then send that script to the user who is having problems. The user can run the script and send a report back to you.

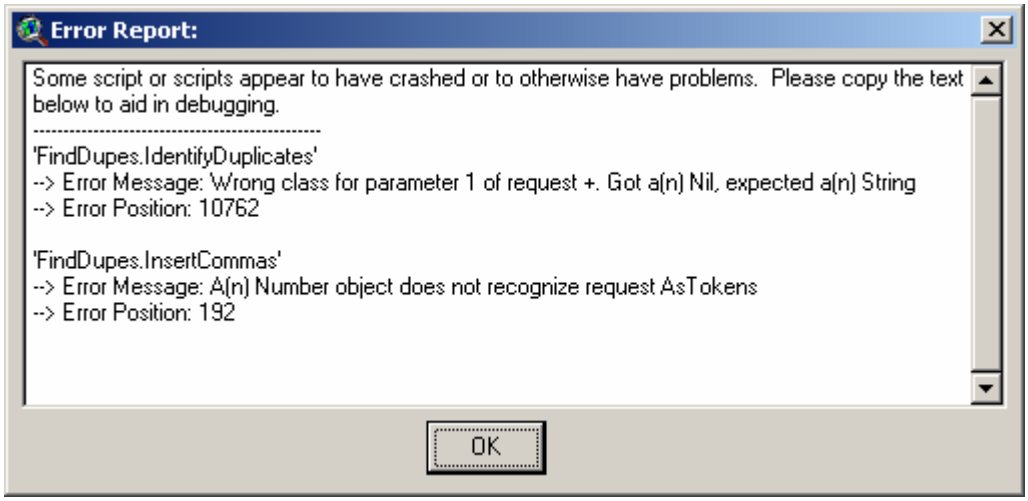
  2) You can automatically generate a script, install it in your project, and insert menu choices in the various GUI Help menus which will run the script. This option is more

useful if you wish to pre-install the error-checking functions before distributing your extension.

Click the "Generate Script to Check Scripts" menu item, select the scripts you want to be examined on the user's computer, and select your options.

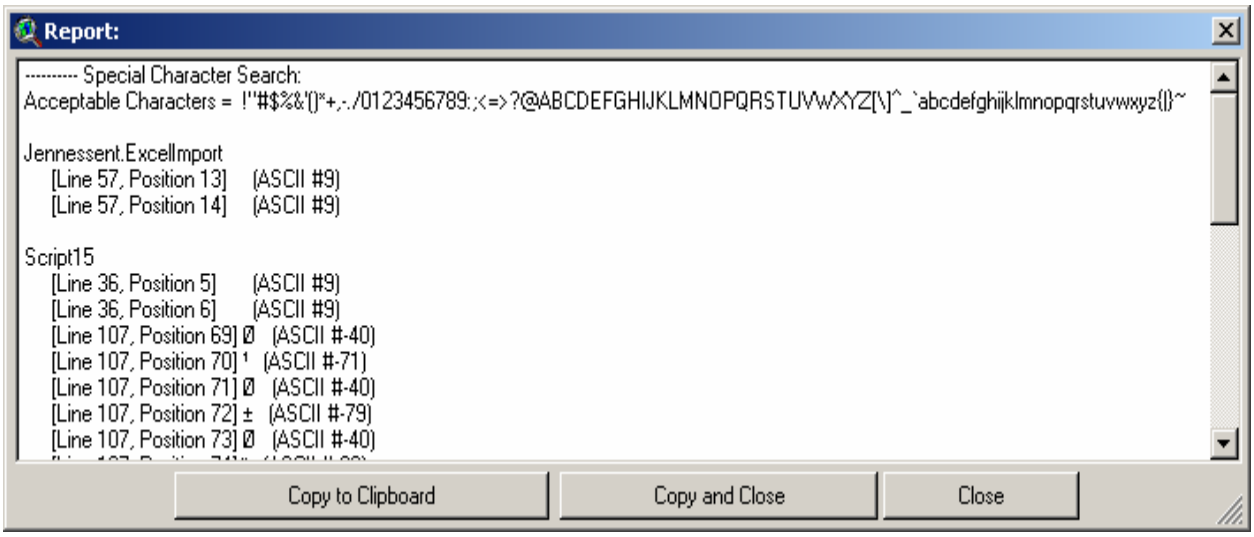


In any case, the script will produce a report similar to that below:

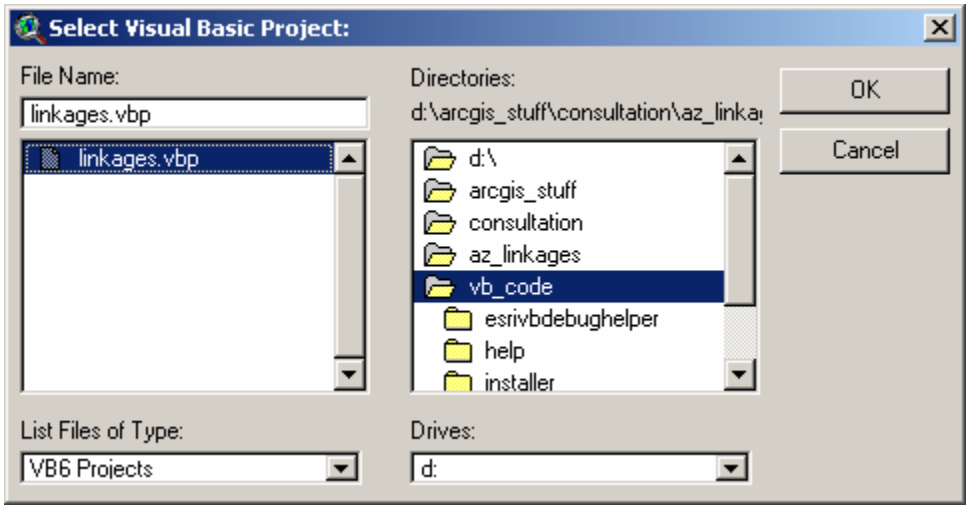


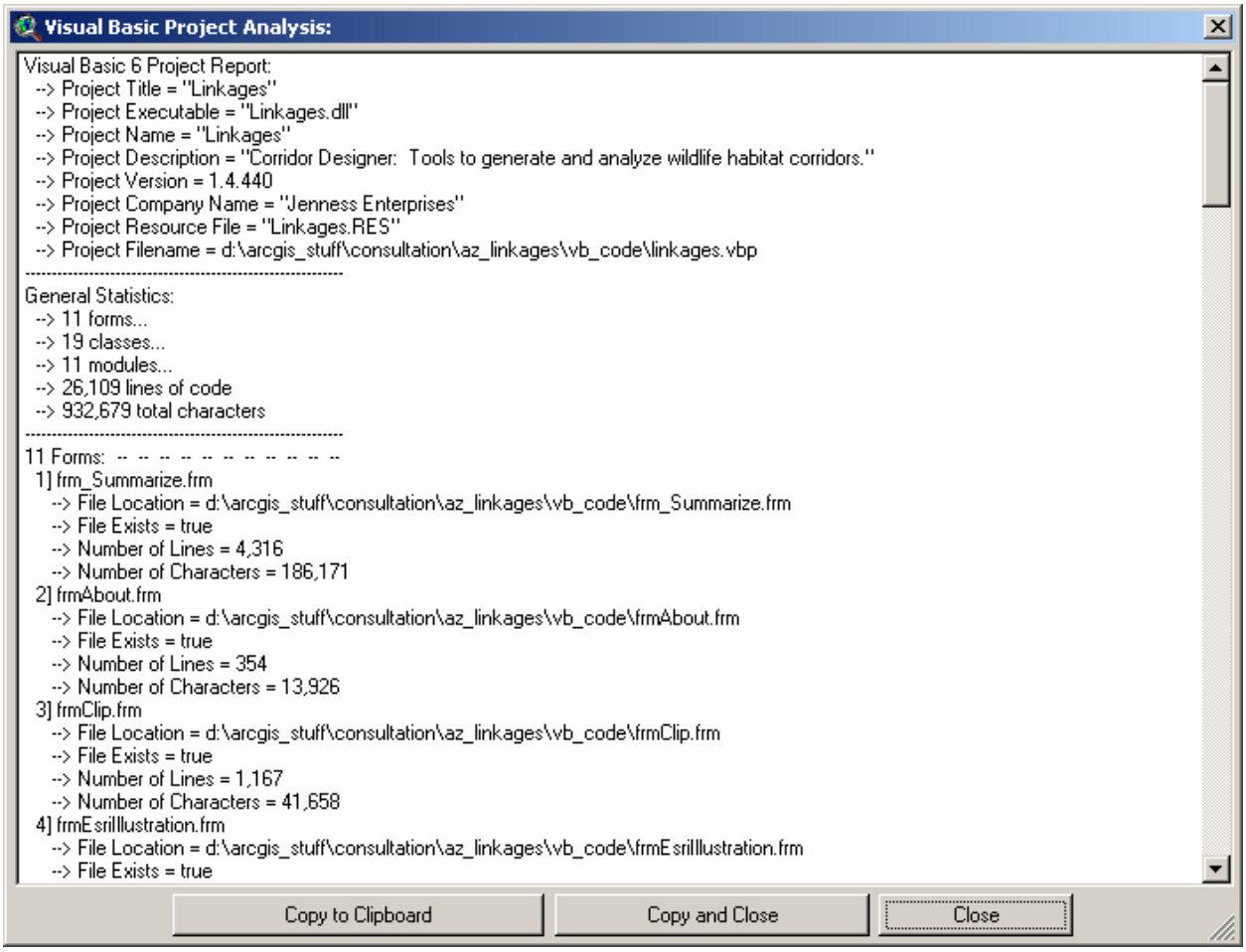
- *Search Scripts for Odd Characters:* I have had a few experiences where my extensions unexpectedly crashed on Chinese computers, which leads me to think that it is possible that unusual characters in the code might trigger the problem. Therefore this function searches your scripts for any character that might not be considered a "standard"

character. Basically it alerts you if it finds any characters with ASCII values that are not between 32 and 125.



- *Statistics on Visual Basic 6 Project:* This function allows you to select an existing Visual Basic 6 project and generate a set of basic statistics describing that project:





- *Copy VB6 Project Files:* This function examines a VB6 project file using the algorithm from the function above, then copies all project, form, class and module files into a new directory. This makes it easier to send the VB6 project file to others, without including extra unnecessary files that may exist in the directory.
- *Combine VB Project Files into Document:* This function combines all files into a single RTF document, including the report described above, where it can be opened using MS Word.

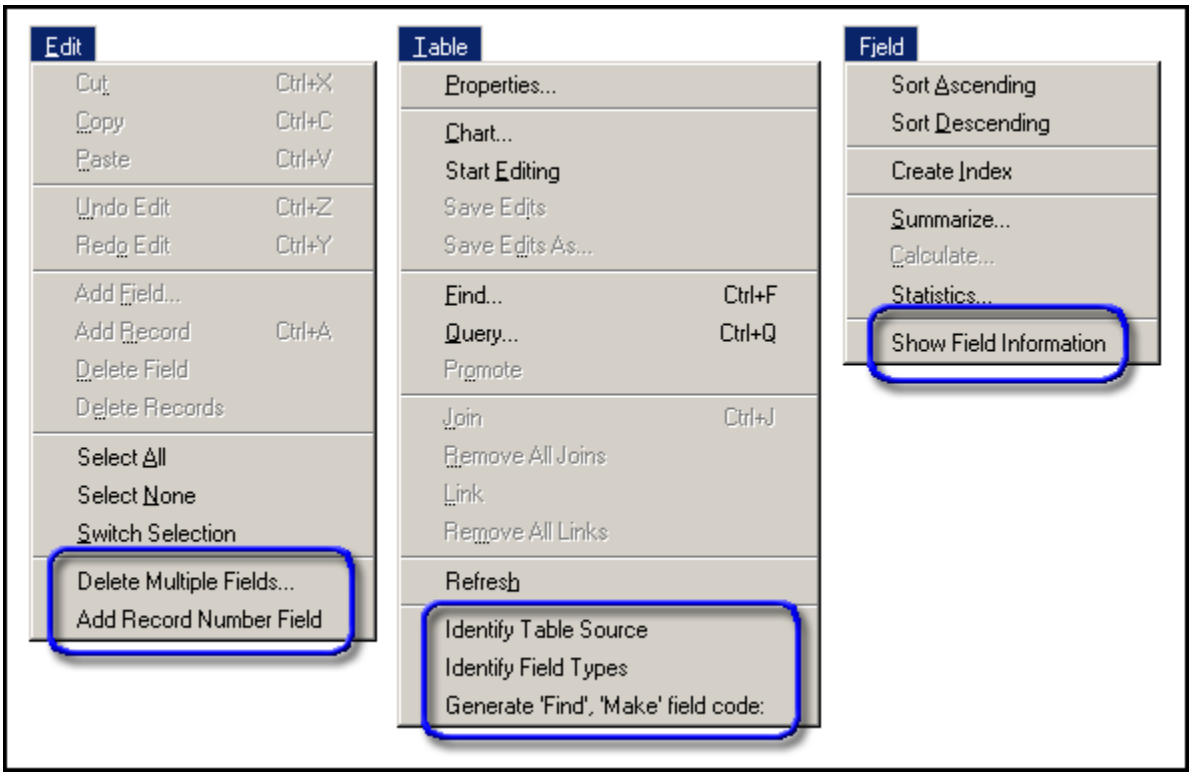
Table Buttons: 

The  button does an Unjoin on the current table so you don't have to search through the menu for it. It is only enabled if the active table has joins.

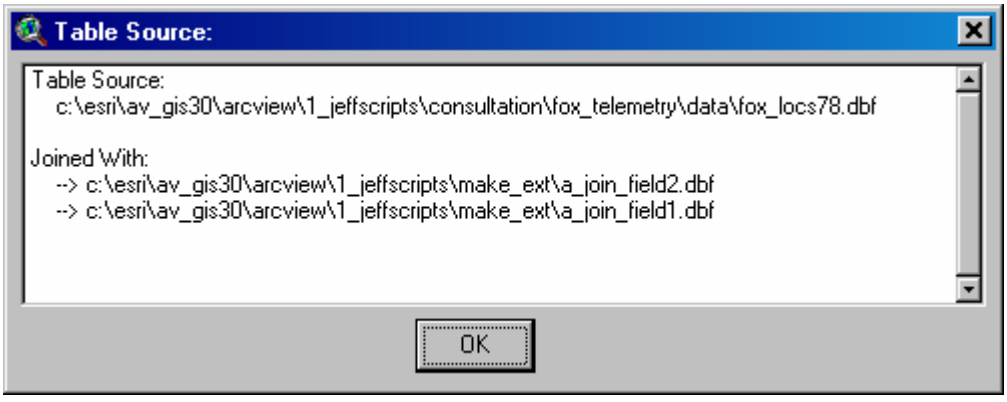
The  button exports a table to an Excel spreadsheet.

The  button imports selected records from an Excel spreadsheet into a dBASE table. This function requires that Excel be open and the records be selected.

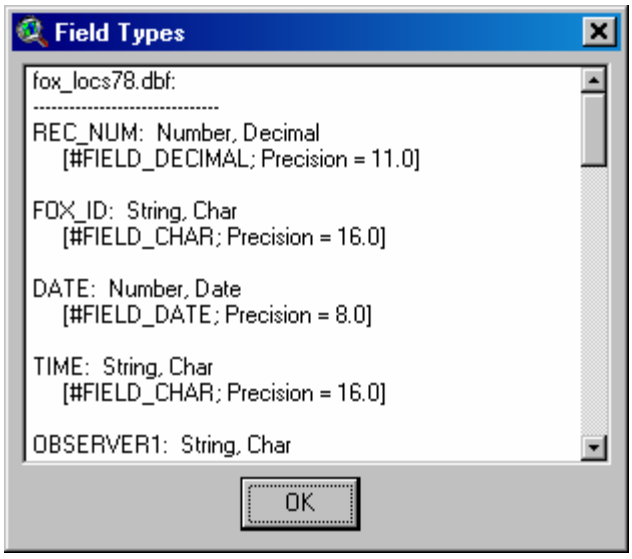
Table Menu Items:



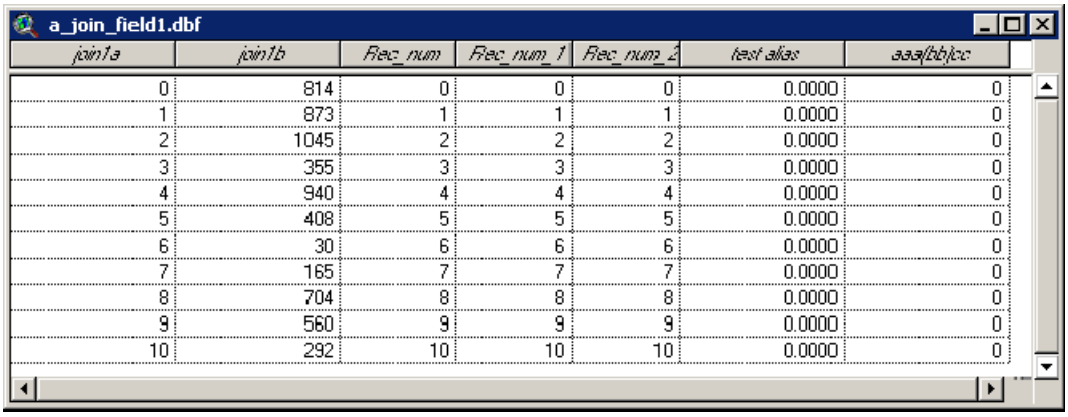
- *Delete Multiple Fields…:* prompts you to select a set of fields, then deletes them all. There is no UNDO for this operation!
- *Add Record Number Field:* adds a new numeric field to your table and populates that field with the record number for each record. This is an easy way to generate unique ID values.
- *Identify Table Source:* shows you the file source of your table, plus any joined files:



- *Identify Field Types:* generates a report listing details of all fields in the table:

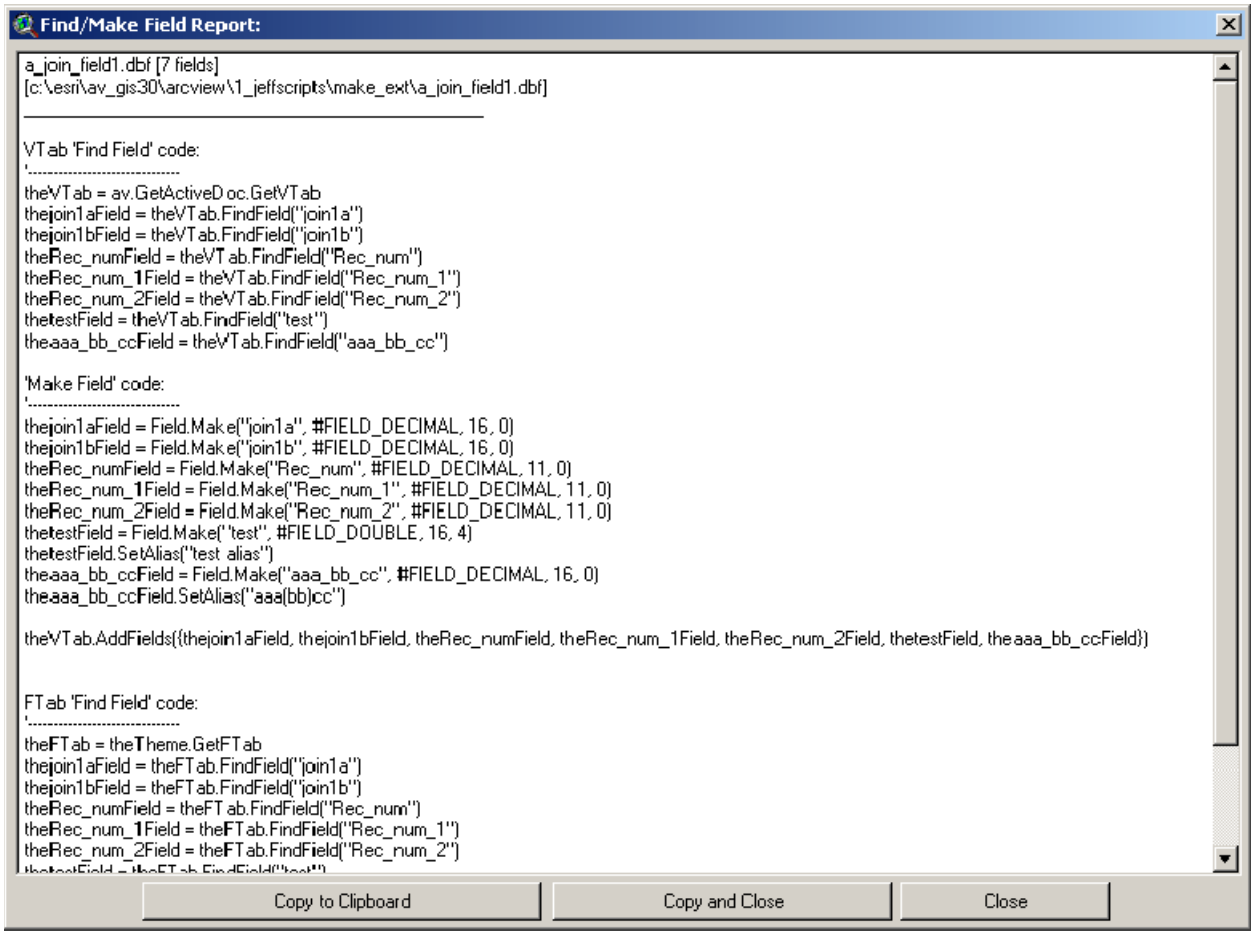


° *Generate 'Find', 'Make' field code:* gives you an easy way to write code to either find the fields in this particular table, or make new fields identical to these fields. Notice that ArcView usually identifies a numeric field type as Decimal no matter what you originally created it as. I don't know what this means, but you may need to redefine your field types afterwards.



a_join_field1.dbf

join1a	join1b	Rec_num	Rec_num_1	Rec_num_2	test alias	aaa(bb)cc
0	814	0	0	0	0.0000	0
1	873	1	1	1	0.0000	0
2	1045	2	2	2	0.0000	0
3	355	3	3	3	0.0000	0
4	940	4	4	4	0.0000	0
5	408	5	5	5	0.0000	0
6	30	6	6	6	0.0000	0
7	165	7	7	7	0.0000	0
8	704	8	8	8	0.0000	0
9	560	9	9	9	0.0000	0
10	292	10	10	10	0.0000	0



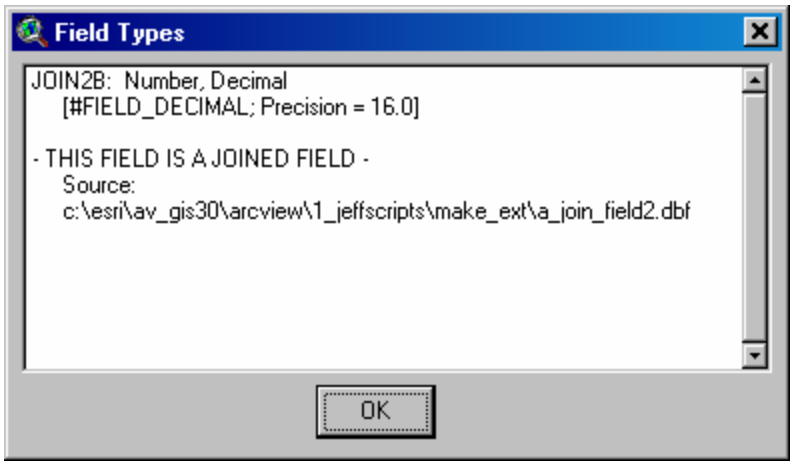
Find/Make Field Report:

```
a_join_field1.dbf [7 fields]
[c:\esri\av_gis30\arcview\1_jeffscripts\make_ext\a_join_field1.dbf]
_____

VTab 'Find Field' code:
'----------------------------
theVTab = av.GetActiveDoc.GetVTab
thejoin1aField = theVTab.FindField("join1a")
thejoin1bField = theVTab.FindField("join1b")
theRec_numField = theVTab.FindField("Rec_num")
theRec_num_1Field = theVTab.FindField("Rec_num_1")
theRec_num_2Field = theVTab.FindField("Rec_num_2")
thetestField = theVTab.FindField("test")
theaaa_bb_ccField = theVTab.FindField("aaa_bb_cc")

'Make Field' code:
'----------------------------
thejoin1aField = Field.Make("join1a", #FIELD_DECIMAL, 16, 0)
thejoin1bField = Field.Make("join1b", #FIELD_DECIMAL, 16, 0)
theRec_numField = Field.Make("Rec_num", #FIELD_DECIMAL, 11, 0)
theRec_num_1Field = Field.Make("Rec_num_1", #FIELD_DECIMAL, 11, 0)
theRec_num_2Field = Field.Make("Rec_num_2", #FIELD_DECIMAL, 11, 0)
thetestField = Field.Make("test", #FIELD_DOUBLE, 16, 4)
thetestField.SetAlias("test alias")
theaaa_bb_ccField = Field.Make("aaa_bb_cc", #FIELD_DECIMAL, 16, 0)
theaaa_bb_ccField.SetAlias("aaa(bb)cc")

theVTab.AddFields({thejoin1aField, thejoin1bField, theRec_numField, theRec_num_1Field, theRec_num_2Field, thetestField, theaaa_bb_ccField})


FTab 'Find Field' code:
'----------------------------
theFTab = theTheme.GetFTab
thejoin1aField = theFTab.FindField("join1a")
thejoin1bField = theFTab.FindField("join1b")
theRec_numField = theFTab.FindField("Rec_num")
theRec_num_1Field = theFTab.FindField("Rec_num_1")
theRec_num_2Field = theFTab.FindField("Rec_num_2")
```

Copy to Clipboard | Copy and Close | Close

The code illustrated above is duplicated for both VTab and FTab objects, so you can copy only the portion you want.

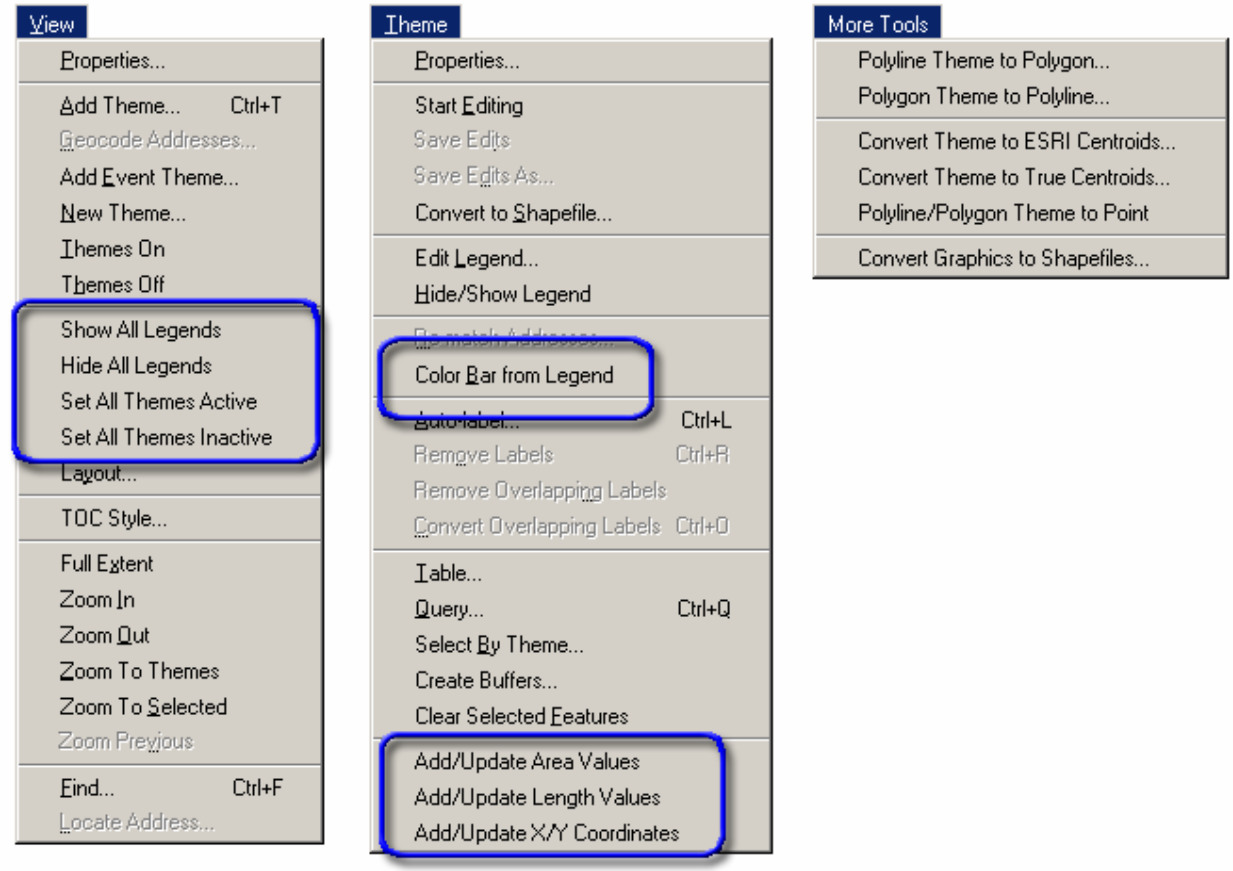
° *Show Field Information:* shows you the field data on that particular field:



View Buttons:

The button takes you forward through the list of Zoom Undoes. It's basically the opposite of the "Zoom Undo" button and becomes disabled any time you manually zoom to a new location. It becomes enabled as soon as you use the "Zoom Undo" function.

View Menu Items:



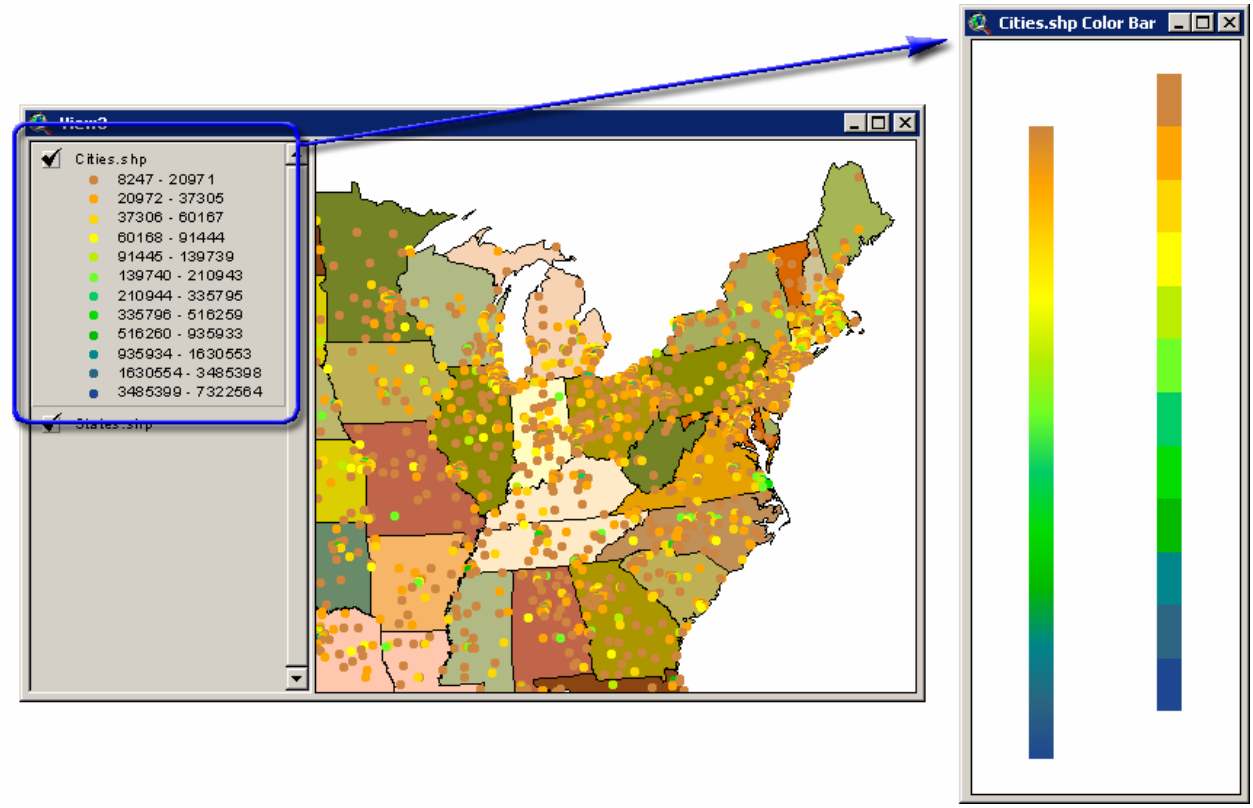
View Menu:

- *Show All Legends:* This opens all the legends for all the themes in the view.
- *Hide All Legends:* This closes all the legends for all the themes in the view, allowing you to see more of the theme names if you have several themes included.
- *Set All Themes Active:* Sets all themes as active. This is useful if you want to delete all or most of the themes in your view and you don't want to click on each one separately to make it active.
- *Set All Themes Inactive:* Sets your table of contents so that none of the themes are active. This is useful if you want to test your "update" scripts for tools that work on active themes.

Theme Menu:

- *Color Bar from Legend:* This extension can create graphic color bars for any grid or feature theme in your view, based on the current legend you have set for that theme. These color bars can help make nice legends for final maps.

  This function will automatically work on all active grid or feature themes in your view, so begin by clicking on those themes to make them active. Next, simply click the "Color Bar from Legend" menu item in the "Theme" menu and specify whether you want the color bars to be vertical or horizontal.



The function makes two color bars; one of which shows only the colors that are actually used in the legends (12 in the example above), while the other smoothly blends between colors.

These color bars are actually in View documents, so they can be added to layouts using the "Add View Frame" function. The color bars can also be copied from the Color Bar window and pasted directly into your views, in which case they will likely need to be resized and repositioned in order to look aesthetically correct. They can be resized and repositioned in the same manner as any other graphic shape by selecting it with the  button and then moving the shape or dragging one of the handles.

You can also take a screenshot of the color bar to paste it into graphic-editing software (i.e. Photoshop, Photopaint, etc.). Click [Alt]-Print Screen to copy the image to the clipboard, then [Control]-V to paste the image into your graphics software.

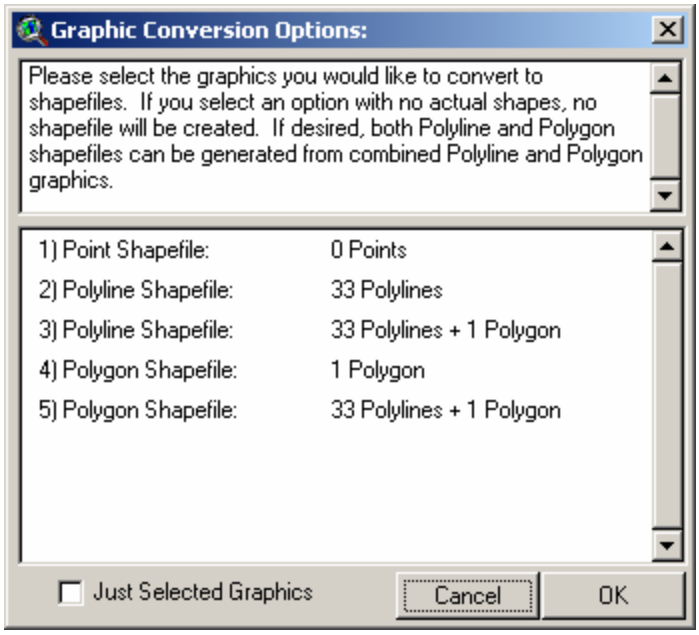
- *Add/Update Area Values:* This function will only be active if you have a single polygon theme active in your view. It will add polygon area values for all polygons in the theme, to either a new field or an existing field. If your view is projected, you have the option to calculate projected area values.
- *Add/Update Length Values:* This function will only be active if you have a single polyline theme active in your view. It will add polyline length values for all polylines in the theme, to either a new field or an existing field. If your view is projected, you have the option to calculate projected length values.
- *Add/Update X/Y Coordinates:* This function will only be active if you have a single point theme active in your view. It will add X- and Y-coordinate values for all points in the theme to either new fields or existing fields. If your view is projected, you have the option to calculate coordinates.

More Tools Menu:

- *Polyline Theme to Polygon:* This function will only be active if you have at least one polyline theme available in your view. It will convert polylines to polygons by connecting the beginning and end of the polylines. It will add area and perimeter fields, plus all the original fields from the original polyline theme.
- *Polygon Theme to Polyline:* This function will only be active if you have at least one polygon theme available in your view. It will convert polygons to polylines, adding a length field plus all the original fields from the original polygon theme.
- *Convert Theme to ESRI Centroids:* This function will only be active if you have at least one feature theme (point, polyline or polygon) available in your view. It will convert the original shapes to centroids and add them to a new point theme. It will not add coordinate fields, but these can be added using the "*Add/Update X/Y Coordinates:* " in the "*Theme*" menu above.
- *Convert Theme to True Centroids:* This function will only be active if you have at least one feature theme (point, polyline or polygon) available in your view. It will convert the original shapes to true centroids, defined as the center of mass of the shapes, and add them to a new point theme. The true center of mass is sometimes different than the

ESRI-derived centroid. It will not add coordinate fields, but these can be added using the "*Add/Update X/Y Coordinates:* " in the "*Theme*" menu above.

- *Polyline/Polygon to Point:* This function will only be active if you have at least one polyline or polygon theme in your view. It will break the original shape down into all the component vertices and add these vertices to a new point theme. The new point theme will contain all the original fields from the original theme.
- *Convert Graphics to Shapefiles:* This function will convert all point, polyline and polygon graphics in your view into shapefiles, and then add them as themes in your view. Click this item and you will be prompted to identify which graphics to convert:



Modifications:

July 30, 2002: Fixed a bug in the “Save All Scripts” button

Changed “Modified Dialog Saver” so that folder button opens to current working directory.

Changed “Modified Dialog Saver” so that the default file name is the same as the dialog, with any “.” or space symbols replaced by underscores.

August 1, 2002: Version 1.1

Modified the “Global Find” tool so it also searches uncompiled scripts. It indicates the fact the script was not compiled by adding “[- Not Compiled -]” to the script name in the report.

August 4, 2002: Version 1.12

Modified the “Global Find” tools so that it correctly searches uncompiled closed scripts.

August 2, 2002: Version 1.15

Added several table menu items:

- Edit: Delete Multiple Fields
- Table: Identify Field Types
- Table: Identify Table Source, including Join Tables
- Field: Show Field Information

November 2002: Version 1.19

Added tools to make dialogs and to produce lists of the components

December 2002: Version 1.20

Modified the “Save Objects” tool to also save spaces (i.e. in menus, button bars and tool bars), provided the custom spaces have something written to their “tags”. It saves the space if the tag <> nil.

Added View menu items to activate/deactivate all themes and to hide/show all legends.

January 2003: Version 1.25

Modified the “Make New Dialog” tool so that you can specify which scripts you want to make, and whether you want the new dialog to be modal and/or resizable.

February 6, 2003: Version 1.28

Added tools to shrink scripts to minimum open size, and added the “Add to ODB” and “Extract from ODB” tools to the Dialog and Project button bars.

February 13, 2003: Version 1.28

Added tools to add record number fields to tables and to add basic intro code to scripts.

February 26, 2003: Version 1.31

Expanded “Make Dialog” function to make 4 types of basic dialogs

March 1, 2003: Version 1.32

Corrected an error in the “Make Progress Meter” dialog in which it wasn’t importing the progress icon image correctly.

Added the option to select from Document Tables in the “Make Theme/ID” dialog.

April 27, 2003: Version 1.42

Added all Script, Project and Dialog menus. Added geometric function scripts, script recursion tools, dialog report tools, “Remove Joins” button and modified comment function.

May 13, 2003: Version 1.43

Corrected a bug in the “Extract Scripts from ODB” script in which it produced an error message upon extracting a tool or button separator.

May 19, 2003: Version 1.46

Added a button to go to a specified location in the script.

May 25, 2003: Version 1.47

Added shortcuts to “Select All”, “Comment” and “Uncomment” script actions.

May 26, 2003: Version 1.48

Added a listbox dialog to the list of standard dialogs.

June 3, 2003: Version 1.49

Added a Table menu function to generate code that finds and makes fields similar to those in the current table. Also added scripts to sort points based on X/Y coordinates or bearing from a central point.

June 17, 2003: Version 1.52

Added a geometry script to check if line segments intersect. Changed all report boxes to the customized report dialog with the “Copy to Clipboard” buttons. Added a script report function to describe the script and add line numbers to the code.

June 28, 2003: Version 1.56

Modified Table Field Report to show both VTab and FTab “FindField” and “AddFields” code.

July 1, 2003: Version 1.57

Added a tool to rearrange the tab order for controls on a dialog.

July 25, 2003: Version 1.60

Added a report of script length statistics to the script recursion reports.

August 14, 2003: Version 1.63

Added a random number generator script.

August 19, 2003: Version 1.65

Modified the “Copy Doc” request to fix a bug copying dialog editors.

October 1, 2003: Version 1.69

Added a function to search dialogs for a particular script.
Added a function to return forward through the “Undo Zoom” steps.

October 2, 2003: Version 1.70

Added a function to insert the cursor at a specified line number in a script.
Modified the “Reverse Undo Zoom” button to position itself next to the “Undo Zoom” button.

October 3, 2003: Version 1.71

Added a “search all scripts” button to the Dialog Editor button bar.
Added a button to open dialogs from *.ded files to the Project button bar.

October 7, 2003: Version 1.72

Added an option to produce a “Insert Commas in Number” script.

October 10, 2003: Version 1.73

Added the “Script/Dialog Tools” menu to the project GUI, with menu items for extracting scripts from files and making project files portable.

October 16, 2003: Version 1.76

Added the “Control Tools” menu to the Dialog Editor GUI, with menu items for shifting and resizing graphic controls.

October 24, 2003: Version 1.78

Fixed a bug that triggered an error when unloading the extension (bug says something like “Unable to find script ‘Jennessent.MakeCopyDoc’”)

November 2, 2003: Version 1.80

Added Pattern-style searching to the Global Search tool.

November 22, 2003: Version 1.81

Modified Script Recursion tools so that it doesn’t identify a script calling itself.

January 8, 2004: Version 1.83

Modified “Build List Dialog, Sortable with Add/Remove” option so the dialog has “Available” and “Selected” labels.

Added an option to generate a “Desired Projection for Calculations” dialog.

January 14, 2004: Version 1.84

Added support for Multipoint shapes to the FindNearestPoints script.

February 12, 2004: Version 1.85

Added a keystroke shortcut to put parentheses around selected script text.

February 15, 2004: Version 1.86

Added code to insert “Make/Print ODB” code, which will display a text representation of an object.

February 18, 2004: Version 1.88

Added code to generate Normally Distributed Random Numbers script.
Modified the “Make Select Projection Dialog” function to include an option for Great Circles in the dialog.

March 4, 2004: Version 1.90

Added code to make a quoted version of a script.

March 19, 2004: Version 1.91

Added functions to shift dialog controls to the vertical and horizontal center of the dialog.

November 30, 2004: Version 1.93

Added function to convert polylines/polygons to points.

Added functions to calculate area fields for polygon themes, length fields for polyline themes, and X/Y-Coordinate fields for Point themes.

December 1, 2004: Version 1.94

Added function to true center of mass.

Added function to create true center of mass script.

Added function to convert polygons to polylines.

January 11, 2005: Version 1.95

Added option to convert graphics to shapefiles.

January 30, 2005: Version 1.97

Add “Time Elapsed” code snippet.

Added corner resize bars to Report and Theme/ID Dialogs

February 8, 2005: Version 1.99

Added functions to search DocGUIs for script names

February 27, 2005: Version 1.992

Modified “Make Theme/ID Dialog” functions so you could specify that “No ID Field” popped up in the list of ID fields.

Added Error Checking code snipped, which locates errors in called scripts.

July 9, 2005: Version 1.995

Several minor changes

Added button to Dialog button bar which adds corner bars to dialogs.

Added option to create color bars from view themes.

July 21, 2005: Version 1.996

Added keyboard shortcut to search current script for text.

Added keyboard shortcut to open help files on a request or object in script.

October 7, 2005: Version 1.999

Added functions to identify problems in multiple scripts, and broke the “Script Tools” menu into 2 separate menus.

Removed all unusual characters, hoping to resolve a problem generated by the Japanese patch to ArcView 3.3.

October 26, 2005: Version 1.9992

Fixed two bugs in the 'Export to Excel' function in which it did not correctly export boolean values and it occasionally was unable to find the Excel.exe file.

Fixed a bug in the auto-generated listbox function in which the associated scripts were incorrectly named.

October 27, 2005: Version 1.9993

Modified "Add Commas in Number" script so that it allows you to send it both numbers and strings, and set a precision level for the return string.

January 2, 2006: Version 2.0

Modified Report Dialog creation tool so that the sample "MakeReport" script takes 3 parameters instead of two. The 3 parameters are now {report text, report title, modal}. The dialog itself is preset to modal or nonmodal based on the user's input.

Modified the "Search Dialogs for Script" and "Search DocGUIs for Script" functions so that they would automatically insert the current script name if they were called from a Script document. If called from a Dialog document, they will not insert any text into the input dialog.

Added a function to generate a string of random characters, which is useful in some cases when inserting text in an existing string. These random characters make useful placeholders.

Fixed a bug in the "Go to Character Number In Script" function in which it would crash if you entered a number larger than the script maximum character value.

Fixed a bug in the "Go to Line Number in Script" function in which it would drift from the true line number in large scripts.

Added a function to the "Dialog Tools" menu to search all scripts for references to a dialog.

Added functions to distribute controls on the dialog so that they maintain a constant horizontal or vertical spacing between the origin points. This differs from the standard distribution function in that this may force one of the end controls to shift slightly to maintain the constant distance, while the standard function keeps both end controls the same. The standard function usually has a few controls that are slightly closer to each other than the rest, though.

Added the number of Dialogs to the "Script Statistics" report.

Fixed a bug in the "Convert Graphics to Shapefiles" function in which it did not correctly convert polylines to polygons.

Added a first draft of functions to convert ArcView 3.x Dialogs to Visual Basic 6 Forms.

February 15, 2006: Version 2.0004

Modified the Dialog Export tool to add functions for resizing dialogs.

Added tools to generate resize class modules for VB 6 projects.

Added tools to generate resize text for existing VB Forms.

Added a tool to search all ArcView scripts for odd characters that might make it crash on Asian installations.

March 12, 2006: Version 2.0005

Modified the "Generate Script to Check Scripts" tool so that it would optionally generate an actual script and auto-install Help menu items.

Added Help menu items to Script/Dialog Tools Doc Types

May 24, 2006: Version 2.0006

Fixed a bug in the Convex Hull script, caused by a situation in which the left-most or right-most points lie in a column with identical X-values, and when there are more values in this column than there are unique X-values in the dataset.

March 2, 2007: Version 2.0008

Fixed a bug in the "Make project file portable" script, in which it would make duplicate datasets in the "portable" folder.

Added code to automatically load the "Script Decryptor" extension if that extension was available.

Modified the "Sort Clockwise" script so it would return the points sorted clockwise, and with the largest gap in compass bearing (going clockwise) occuring between the last point and the first point in the list. It now also returns the range of compass bearings between the first and last point. It also rounds compass bearings to the nearest 1000^{th} of a degree before sorting them.

March 11, 2007: Version 2.0009

Added a function to generate a "Select Folder" dialog.

Added a function to generate a “Select Color” dialog.

March 15, 2007: Version 2.0010

Modified the function to generate a “Select Folder” dialog to make it more flexible and aesthetic.

April 25, 2007: Version 2.0011

Further modified the function to generate a “Select Folder” dialog to make it more flexible and aesthetic and to correct a minor bug.

September 6, 2007: Version 2.0012

Added a function to generate descriptive statistics on existing VB6 projects.

October 25, 2007: Version 2.0014

Added a function to export VB6 projects.

November 3, 2007: Version 2.0015

Added a function to combine all form, class and module files from a VB6 project into a single rich text file document.

November 3, 2007: Version 2.0016

Added a function to quote text in such a way that VB6 can use it.

Appendix: Scripts Generated

Basic Dialog Scripts

"OPEN" script:

```
' jennessent.SampleDialogOpen

AVUpperLeft = av.ReturnOrigin
AVCenter = avUpperLeft + (av.ReturnExtent / (2@2))
halfDialogWidthHeight = Self.ReturnExtent.ReturnSize / (2@2)
MovePoint = AVCenter - halfDialogWidthHeight
Self.MoveTo(MovePoint.GetX, MovePoint.GetY)

theDialog = self
cmdOK = theDialog.FindByName("cmdOK")
cmdCancel = theDialog.FindByName("cmdCancel")
```

"ACTIVATE" script:

```
' jennessent.SampleDialogActivate

theDialog = self
cmdOK = theDialog.FindByName("cmdOK")
cmdCancel = theDialog.FindByName("cmdCancel")
```

"CLOSE" script:

```
' jennessent.SampleDialogClose

self.SetObjectTag(nil)
self.FindByName("cmdOK").SetObjectTag(nil)
self.FindByName("cmdCancel").SetObjectTag(nil)
```

"OK" script:

```
' jennessent.SampleDialogOK

self.GetDialog.Close
```

"CANCEL" script:

```
' jennessent.SampleDialogCancel

self.GetDialog.SetModalResult(nil)
self.GetDialog.Close
```

MultiChoice Scripts:

"MultiChoice" script:

```
' Jennessent.SampleMultiChoice

if ((self.Count) <> 4) then
  msgBox.Warning("Wrong number of parameters for MultiChoice Message Box; expected 4.",
     "Avenue Runtime Error:")
  return nil
end

theMessage = self.Get(0)
theTitle = self.Get(1)
theListOfLabels = self.Get(2)
theListOfLists = self.Get(3)

if (theMessage.Count > 55) then
  theWords = theMessage.AsList
  theCounter = 0
  theTempMessage = ""
  for each aWord in theWords
    theCounter = theCounter+aWord.Count+1
    if (theCounter > 55) then
      theTempMessage = theTempMessage+NL+aWord
      theCounter = 0
    else
      theTempMessage = theTempMessage++aWord
    end
  end
  theMessage = theTempMessage.Trim
end

if (theListOfLabels.Count <> theListOfLists.Count) then
  msgBox.Info("Number of labels is not equal to number of lists!  Bailing out...", "Problem:")
  return nil
end

' FOLLOWING JUST BECAUSE DIALOG BECOMES TOO BIG FOR MANY SCREEN RESOLUTIONS
if (theListOfLabels.Count > 20) then
  msgBox.Info("Too many lists!  Please limit number of lists to <= 15...", "Problem:")
  return nil
end

AllStrings = True
for each aLabel in theListOfLabels
  if (aLabel.Is(String).Not) then
    AllStrings = False
    break
  end
end

if (AllStrings.Not) then
  msgBox.Info("Labels (parameter 2) must all be strings!  Bailing out...", "Problem:")
  return nil
```

```
end

' IDENTIFY LOCATIONS FOR DIALOG COMPONENTS
theBasicHeight = 55
theAddedHeight = theListOfLabels.Count*30
FinalHeight = theBasicHeight+theAddedHeight
theWidth = 428
theSize = theWidth@FinalHeight
theRect = Rect.Make(0@0, theSize)

AVUpperLeft = av.ReturnOrigin
AVCenter = avUpperLeft + (av.ReturnExtent / (2@2))
halfDialogWidthHeight = theSize / (2@2)
MovePoint = AVCenter - halfDialogWidthHeight

theRect.SetOrigin(MovePoint.GetX@MovePoint.GetY)

' MAKE DIALOG
theDialog = Dialog.MakeSized (True, True, True, True, theRect)
theDialog.SetModal(True)
theDialog.SetTitle(theTitle)
theControlPanel = theDialog.GetControlPanel

' ADD COMBO BOXES
theYPos = 25
theCounter = 0
theListOfCbxNames = {}
for each anIndex in 0..(theListOfLabels.Count-1)
  theComboFasteners = {#CONTROL_FASTENER_TOP, #CONTROL_FASTENER_LEFT, #CONTROL_FASTENER_RIGHT,
      #CONTROL_FASTENER_HEIGHT}
  theCounter = theCounter+1
  theYPos = theYPos+30

  theComboBox = ComboBox.Make
  theComboBox.SetName("AComboBox"+anIndex.AsString)
  theListOfCbxNames.Add(theComboBox.GetName)
  theComboBox.SetLabel(theListOfLabels.Get(anIndex))
  theControlPanel.Add(theComboBox, Rect.Make(7@theYPos, 321@114))
  theComboBox.SetFasteners(theComboFasteners)
  theComboBox.DefineFromList(theListOfLists.Get(anIndex))

end

theButtonFasteners = {#CONTROL_FASTENER_TOP, #CONTROL_FASTENER_WIDTH, #CONTROL_FASTENER_RIGHT,
      #CONTROL_FASTENER_HEIGHT}

' ADD OK BUTTON
theOKScriptString = "theList = {}"+NL
for each aCbxName in theListOfCbxNames
  theOKScriptString = theOKScriptString+
      "theList.Add(self.GetDialog.FindByName("+aCbxName.Quote+").GetCurrentValue)"+NL
```

```
end
theOKScriptString = theOKScriptString+"self.GetDialog.SetModalResult(theList)"+NL+"self.GetDialog.Close"
theOKScriptName = "temp_cbx_ok"
theOKBaseName = "temp_cbx_ok"
theOKCounter = 0
while(av.FindDoc(theOKScriptName) <> nil)
  theOKCounter = theOKCounter+1
  theOKScriptName = theOKBaseName+theOKCounter.AsString
end
theOKSEd = SEd.MakeFromSource(theOKScriptString, theOKScriptName)

theOKButton = LabelButton.Make
theOKButton.SetName("cmdOK")
theOKButton.SetLabel("OK")
theControlPanel.Add(theOKButton, Rect.Make(344@14, 71@24))
theOKButton.SetFasteners(theButtonFasteners)
theOKButton.SetClick(theOKScriptName)

' ADD CANCEL BUTTON
theCancelScriptString = "self.GetDialog.SetModalResult(nil)"+NL+"self.GetDialog.Close"
theCancelScriptName = "temp_cbx_Cancel"
theCancelBaseName = "temp_cbx_Cancel"
theCancelCounter = 0
while(av.FindDoc(theCancelScriptName) <> nil)
  theCancelCounter = theCancelCounter+1
  theCancelScriptName = theCancelBaseName+theCancelCounter.AsString
end
theCancelSEd = SEd.MakeFromSource(theCancelScriptString, theCancelScriptName)
theCancelSEd.Compile

theCancelButton = LabelButton.Make
theCancelButton.SetName("cmdCancel")
theCancelButton.SetLabel("Cancel")
theCancelButton.SetClick(theCancelScriptName)
theControlPanel.Add(theCancelButton, Rect.Make(344@47, 71@24))
theCancelButton.SetFasteners(theButtonFasteners)

' ADD MESSAGE
theMessageLabel = TextLabel.Make
theMessageLabel.SetName("lblMessage")
theMessageLabel.SetLabel(theMessage)
theMessageLabel.SetFasteners({#CONTROL_FASTENER_TOP, #CONTROL_FASTENER_WIDTH, #CONTROL_FASTENER_LEFT,
      #CONTROL_FASTENER_HEIGHT})
theControlPanel.Add(theMessageLabel, Rect.Make(5@5, 328@45))

theOutput = theDialog.Open

theDialog = nil
av.GetProject.RemoveDoc(theCancelSEd)
av.GetProject.RemoveDoc(theOKSEd)
```

```
return theOutput
```

"MultiChoice Sample Code" script:

```
' Jennessent.SampleMultiChoice_sample_code

' PASTE THE FOLLOWING LINE INTO THE TOP OF YOUR SCRIPT SOMEWHERE BEFORE THE
' MULTI-CHOICE BOX GETS CALLED:
MsgMultiChoice = av.FindScript("Jennessent.SampleMultiChoice")

' THIS MESSAGE BOX REQUIRES 4 PARAMETERS:
' 0) THE MESSAGE TO SHOW ABOVE THE DROP-DOWN BOXES:  MUST BE A STRING
' 1) THE DIALOG TITLE:                               MUST BE A STRING
' 2) A LIST OF LABELS FOR EACH DROP-DOWN BOX:        MUST CONTAIN ALL STRING LABELS
' 3) A LIST OF LISTS, TO FILL THE DROP-DOWN BOXES:   THESE LISTS CAN CONTAIN ANY OBJECTS

' USE THE FOLLOWING LINE TO OPEN A MULTI-CHOICE BOX AND RETURN A LIST OF SELECTED ITEMS:
' theChoices = MsgMultiChoice.DoIt({theMessage, theTitle, theListOfLabels, theListOfLists})

' FOR EXAMPLE, THE FOLLOWING CODE WILL GENERATE A MULTI-CHOICE MESSAGE BOX CONTAINING 6 DROP-DOWN LISTS
' AND RETURN THE 6 SELECTED VALUES IN A SINGLE LIST.  IT WILL THEN SHOW YOU A LIST OF THE SELECTED VALUES.

theMessage = "This is a sample message:"
theTitle = "This is a sample MultiChoice Title:"
theListOfLabels = {"Arizona", "New Mexico", "California", "Utah", "Colorado", "Nevada"}
theAZList = {5, 10, 15, 20, 25, 30, 35}
theNewMexicoList = theAZList.Clone
theCaliforniaList = theAZList.Clone
theUtahList = theAZList.Clone
theColoradoList = theAZList.Clone
theNevadaList = theAZList.Clone
theListOfLists = {theAZList, theNewMexicoList, theCaliforniaList, theUtahList,
    theColoradoList, theNevadaList}

' GET CHOICES
theChoices = MsgMultiChoice.DoIt({theMessage, theTitle, theListOfLabels, theListOfLists})
if (theChoices = nil) then return nil end

' MAKE REPORT OF CHOICES
theReport = ""
for each anIndex in 0..(theListOfLabels.Count-1)
  theReport = theReport+theListOfLabels.Get(anIndex)+": Choice = "+theChoices.Get(anIndex).AsString+NL
end
msgBox.Report(theReport, "Report of Choices:")
```

Progress Meter Scripts

"Open" script:

```
' Jennessent.SampleProgressMeter_Open

theProject = av.GetProject
icnProgressLine = self.FindByName("icnProgressLine")
panelProgressLine = self.FindByName("panelProgressLine")
lblTimeLeft = self.FindByName("lblTimeLeft")
```

```
lblCurrentTime = self.FindByName("lblCurrentTime")
lblBeginTime = self.FindByName("lblBeginTime")
lblIndex = self.FindByName("lblIndex")
lblPercentDone = self.FindByName("lblPercentDone")
lblRecordNumber = self.FindByName("lblRecordNumber")

AVUpperLeft = av.ReturnOrigin
AVCenter = avUpperLeft + (av.ReturnExtent / (2@2))
aDialog =  self
halfDialogWidthHeight = aDialog.ReturnExtent.ReturnSize / (2@2)
MovePoint = AVCenter - halfDialogWidthHeight
aDialog.MoveTo(MovePoint.GetX, MovePoint.GetY)

lblCurrentTime.SetLabel(date.now.setFormat("h:m:s AMPM").AsString)
lblTimeLeft.SetLabel ("Estimated time remaining: ---:---:---")
lblPercentDone.SetLabel ("(00.0%)")
lblBeginTime.SetLabel("---:---")
lblRecordNumber.SetLabel("-----")
lblIndex.SetLabel("-----")

self.SetTitle("Current Status...")
icnProgressLine.ResizeTo(0,13)              ' START GREEN PROGRESS BAR AT 0 PIXELS WIDE
```

"EstTimeLeft" script:

```
' Jennessent.SampleProgressMeter_EstTimeLeft

' ESTIMATED TIME LEFT CODE

BeginTime = self.Get(0)
theRecordNumber = self.Get(1)
theRecordCount = self.Get(2)
theProgressDialog = self.Get(3)
thePDTimeLeft = self.Get(4)
thePDPercentDone = self.Get(5)
thePDProgressBar = self.Get(6)
thePDCurrentTime = self.Get(7)
thePDDescription = self.Get(8)
thePDCurrentStep = self.Get(9)
theCurrentDescription = self.Get(10)

thePDDescription.SetLabel(theCurrentDescription)
thePDCurrentStep.SetLabel("Working on Step "+theRecordNumber.AsString+" of "+theRecordCount.AsString+"...")

thePercentDone = (theRecordNumber/theRecordCount)*100
thePDProgressBar.ResizeTo((thePercentDone*2.68), 10)  ' RESIZES PROGRESS BAR TO FULL SIZE OF 268 PIXELS
thePercentDone.SetFormat ("dd.d")
thePDPercentDone.SetLabel ("("+thePercentDone.AsString+"%)")
thePDCurrentTime.SetLabel(Date.Now.SetFormat("hhh:m:s").AsString)

theDuration = (Date.Now - BeginTime).AsSeconds
```

```
PredictedDuration = (theDuration * theRecordCount)/(theRecordNumber+1)
EstTimeLeft = (PredictedDuration-theDuration)+1

EstHoursLeft = (EstTimeLeft/3600).Truncate
EstMinutesLeft = ((EstTimeLeft - (EstHoursLeft*3600))/60).Truncate
EstSecondsLeft = (EstTimeLeft - (EstHoursLeft*3600) - (EstMinutesLeft*60)).Truncate
EstHoursStr = EstHoursLeft.AsString
If (EstMinutesLeft >= 10) then
  EstMinutesStr = EstMinutesLeft.AsString
else
  EstMinutesStr = "0"+EstMinutesLeft.AsString
end
If (EstSecondsLeft >= 10) then
  EstSecondsStr = EstSecondsLeft.AsString
else
  EstSecondsStr = "0"+EstSecondsLeft.AsString
end

EstTimeLeftStr = EstHoursStr +":"+EstMinutesStr+":"+EstSecondsStr
thePDTimeLeft.SetLabel ("Estimated time remaining:  "+EstTimeLeftStr)
theProgressDialog.Activate

return nil
```

"Sample Code" script:

```
' Jennessent.SampleProgressMeter_SampleCode

' THE PROGRESS METER CONTAINS 7 CONTROLS THAT YOU CAN MODIFY AS YOUR SCRIPT RUNS:
'    thePDBeginTime:  A TEXT LABEL INTENDED TO DISPLAY THE TIME THE SCRIPT STARTED
'  thePDCurrentTime:  A TEXT LABEL INTENDED TO DISPLAY THE CURRENT TIME
' thePDRecordNumber:  A TEXT LABEL INTENDED TO DISPLAY THE CURRENT STEP NUMBER, PLUS
'                     THE TOTAL NUMBER OF STEPS EXPECTED IN THE ANALYSIS
'        thePDIndex:  A TEXT LABEL INTENDED TO DISPLAY A BRIEF DESCRIPTION OF THE CURRENT TASK
'     thePDTimeLeft:  A TEXT LABEL INTENDED TO DISPLAY THE ESTIMATED TIME REMAINING
'  thePDPercentDone:  A TEXT LABEL INTENDED TO DISPLAY NUMERICALLY THE PERCENTAGE DONE
'  thePDProgressBar:  AN ICON INTENDED TO DISPLAY GRAPHICALLY THE AMOUNT DONE

' PASTE THE FOLLOWING CODE INTO THE TOP OF YOUR SCRIPT SOMEWHERE TO IDENTIFY DIALOG CONTROLS
' AND SCRIPTS AND PRESET DIALOG COMPONENTS.

' PROGRESS METER STUFF ---------------------------------------------
UpdateProgress = av.FindScript("Jennessent.SampleProgressMeter_EstTimeLeft")
theProgressDialog = av.FindDialog ("Jennessent.SampleProgressMeter")
thePDBeginTime = theProgressDialog.FindByName("lblBeginTime")
thePDCurrentTime = theProgressDialog.FindByName ("lblCurrentTime")
thePDCurrentStep = theProgressDialog.FindByName("lblRecordNumber")
thePDDescription = theProgressDialog.FindByName("lblIndex")
thePDTimeLeft = theProgressDialog.FindByName("lblTimeLeft")
thePDPercentDone = theProgressDialog.FindByName("lblPercentDone")
thePDProgressBar = theProgressDialog.FindByName("icnProgressLine")
```

```
thePDCurrentStep.SetLabel ("Current Record Number or Step...")
thePDPercentDone.SetLabel ("0%)")
thePDCurrentTime.SetLabel(date.now.setFormat("h:m:s AMPM").AsString)
theIndexLabel = "Description of Current Task..."
thePDDescription.SetLabel ("Preparing Data...")
thePDTimeLeft.SetLabel ("Estimated time remaining: ---:---:---")
thePDProgressBar.ResizeTo(0,13)              ' START GREEN PROGRESS BAR AT 0 PIXELS WIDE
' END PROGRESS METER STUFF ------------------------------------------

' PASTE THE FOLLOWING CODE IN AT THE POINT YOU WANT THE PROGRESS DIALOG TO OPEN AND START WORKING:
theProgressDialog.Open
theProgressDialog.Activate
BeginTime = Date.Now.SetFormat("MMMM d, h:m:s AMPM")
thePDBeginTime.SetLabel("Began Job:  "+BeginTime.AsString)
thePDCurrentTime.SetLabel(date.now.setFormat("h:m:s AMPM").AsString)

' AT ANY POINT YOU CAN SET ANY OF THE TEXT LABELS BY THE FOLLOWING CODE:
' theTextLabelName.SetLabel("your label")
' TO UPDATE ALL PORTIONS OF THE PROGRESS METER INCLUDING THE ESTIMATED TIME LEFT, YOU NEED TO
' IDENTIFY THE FOLLOWING VARIABLES:

'  theStepNumber = THE NUMBER OF THE CURRENT STEP OR CALCULATION.  FOR EXAMPLE, IF YOU WERE ON THE
'                  4TH OF 10 CALCULATIONS, THIS NUMBER WOULD BE 4.
'   theStepCount = THE TOTAL NUMBER OF CALCULATIONS.  FOR EXAMPLE, IF YOU WERE ON THE 4TH OF 10
'                  CALCULATIONS, THIS NUMBER WOULD BE 10.
' theDescription = A BRIEF MESSAGE DESCRIBING THE CURRENT CALCULATION.

' THEN PASTE THE FOLLOWING CODE TO UPDATE THE PROGRESS METER.  MOST OF THESE LIST ITEMS HAVE
' BEEN IDENTIFIED AT THE BEGINNING OF THE SCRIPT.  MAKE SURE TO REMOVE COMMENT TAGS

' UpdateProgress.DoIt({BeginTime, theStepNumber, theStepCount, theProgressDialog,
'       thePDTimeLeft, thePDPercentDone, thePDProgressBar, thePDCurrentTime,
'       thePDDescription, thePDCurrentStep, theDescription})

' FOR EXAMPLE:  ASSUMING YOU HAVE ALREADY PASTED THE INTRODUCTORY CODE ABOVE, THE FOLLOWING SCRIPTS
' ILLUSTRATE HOW THE PROGRESS DIALOG WORKS:

' THIS SCRIPT UPDATES THE PROGRESS DIALOG AT EVERY STEP
theProgressDialog.Open
theProgressDialog.Activate
BeginTime = Date.Now.SetFormat("MMMM d, h:m:s AMPM")
thePDBeginTime.SetLabel("Began Job:  "+BeginTime.AsString)
thePDCurrentTime.SetLabel(date.now.setFormat("h:m:s AMPM").AsString)

theTotalCount = 2000
theDescription = "Counting to 2,000"
for each aNumber in 0..theTotalCount
  UpdateProgress.DoIt({BeginTime, aNumber, theTotalCount, theProgressDialog,
        thePDTimeLeft, thePDPercentDone, thePDProgressBar, thePDCurrentTime,
        thePDDescription, thePDCurrentStep, theDescription})
end
```

```
theProgressDialog.Close

' THIS SCRIPT UPDATES THE PROGRESS DIALOG EVERY SECOND
theProgressDialog.Open
theProgressDialog.Activate
BeginTime = Date.Now.SetFormat("MMMM d, h:m:s AMPM")
thePDBeginTime.SetLabel("Began Job:  "+BeginTime.AsString)
thePDCurrentTime.SetLabel(date.now.setFormat("h:m:s AMPM").AsString)

theTotalCount = 40000
theDescription = "Counting to 40,000"
theTestTime = Date.Now
for each aNumber in 0..theTotalCount
  if ((Date.Now - theTestTime).AsSeconds >=1) then
    UpdateProgress.DoIt({BeginTime, aNumber, theTotalCount, theProgressDialog,
          thePDTimeLeft, thePDPercentDone, thePDProgressBar, thePDCurrentTime,
          thePDDescription, thePDCurrentStep, theDescription})
    theTestTime = Date.Now
  end
end
theProgressDialog.Close
```

Theme and ID Field Scripts:

"Cancel" script:

```
' Jennessent.SampleThemeID_Cancel

self.GetDialog.SetModalResult(nil)
self.GetDialog.Close
```

"CheckOK" script:

```
' Jennessent.SampleThemeID_CheckOK

theDialog = av.FindDialog("Jennessent.SampleThemeID")
cmdOK = theDialog.FindByName("cmdOK")
lbxField = theDialog.FindByName("lbxField")
lbxTheme = theDialog.FindByName("lbxTheme")

cmdOK.SetEnabled(lbxField.HasSelection and lbxTheme.HasSelection)
```

"Close" script:

```
' Jennessent.SampleThemeID_Close

Self.SetObjectTag(nil)
Self.FindByName("cmdCancel").SetObjectTag(nil)
Self.FindByName("cmdOK").SetObjectTag(nil)
```

```
Self.FindByName("lblSelectField").SetObjectTag(nil)
Self.FindByName("lblSelectTheme").SetObjectTag(nil)
Self.FindByName("lbxField").SetObjectTag(nil)
Self.FindByName("lbxTheme").SetObjectTag(nil)
```

"OK" script:

```
' Jennessent.SampleThemeID_OK

theDialog = av.FindDialog("Jennessent.SampleThemeID")
lbxField = theDialog.FindByName("lbxField")
lbxTheme = theDialog.FindByName("lbxTheme")

self.GetDialog.SetModalResult({lbxTheme.GetCurrentValue, lbxField.GetCurrentValue})
self.GetDialog.Close
```

"Open" script:

```
' Jennessent.SampleThemeID_Open

AVUpperLeft = av.ReturnOrigin
AVCenter = avUpperLeft + (av.ReturnExtent / (2@2))
halfDialogWidthHeight = Self.ReturnExtent.ReturnSize / (2@2)
MovePoint = AVCenter - halfDialogWidthHeight
Self.MoveTo(MovePoint.GetX, MovePoint.GetY)

theDialog = av.FindDialog("Jennessent.SampleThemeID")
cmdCancel = theDialog.FindByName("cmdCancel")
cmdOK = theDialog.FindByName("cmdOK")
lblSelectField = theDialog.FindByName("lblSelectField")
lblSelectTheme = theDialog.FindByName("lblSelectTheme")
lbxField = theDialog.FindByName("lbxField")
lbxTheme = theDialog.FindByName("lbxTheme")

theView = av.GetActiveDoc
theThemes = theView.GetThemes
theSelectThemes = {}
for each aTheme in theThemes
  if (aTheme.Is(FTheme)) then
    if (aTheme.GetFTab.GetShapeClass.GetClassName = "Polygon") then
      theSelectThemes.Add(aTheme)
    end
  end
end
lbxTheme.DefineFromList(theSelectThemes)
lbxField.Empty
cmdOK.SetEnabled(False)
```

"Select Theme" script:

```
' Jennessent.SampleThemeID_SelectTheme

theDialog = av.FindDialog("Jennessent.SampleThemeID")
lbxField = theDialog.FindByName("lbxField")

theTheme = Self.GetCurrentValue
if (theTheme.Is(FTheme)) then
  theVTab = theTheme.GetFTab
else
  theVTab = theTheme.GetVTab
end

theFieldList = {}
for each aField in theVTab.GetFields
  if (aField.IsTypeShape.Not) then theFieldList.Add(aField) end
end

lbxField.DefineFromList(theFieldList)

av.Run("Jennessent.SampleThemeID_CheckOK", nil)
```

"Sample Code" script:

```
' Jennessent.SampleThemeID_sample code

' THIS IS A SIMPLE EXTENSION TO OPERATE.  FIRST IDENTIFY THE DIALOG AND THEN USE THE 'OPEN'
' REQUEST TO RUN IT.  THE FOLLOWING CODE IDENTIFIES THE DIALOG, OPENS IT, THEN SHOWS YOU
' THE SELECTED THEME AND FIELD.  THE DEFAULT CODE ASSUMES IT IS BEING RUN FROM A VIEW.

theThemeIDDialog = av.FindDialog("Jennessent.SampleThemeID")
theThemeAndID = theThemeIDDialog.Open
if (theThemeAndID = nil) then return nil end
if (theThemeAndID.Get(0) = nil) then return nil end
theTheme = theThemeAndID.Get(0)
theIDField = theThemeAndID.Get(1)
msgBox.List(theThemeAndID, "Selected Theme and ID Field...", "Dialog Results:")
```

Report Dialog Scripts:

"Open" script:

```
' Jennessent.ReportDialog_Open
' Jenness Enterprises <www.jennessent.com>

AVUpperLeft = av.ReturnOrigin
AVCenter = avUpperLeft + (av.ReturnExtent / (2@2))
halfDialogWidthHeight = Self.ReturnExtent.ReturnSize / (2@2)
MovePoint = AVCenter - halfDialogWidthHeight
Self.MoveTo(MovePoint.GetX, MovePoint.GetY)
```

"OK" script:

```
' Jennessent.ReportDialog_OK
' Jenness Enterprises <www.jennessent.com>

self.GetDialog.Close
```

"Close" script:

```
' Jennessent.ReportDialog_Close
' Jenness Enterprises <www.jennessent.com>

Self.SetObjectTag(nil)
Self.FindByName("cmdOK").SetObjectTag(nil)
Self.FindByName("txtReport").SetObjectTag(nil)
Self.FindByName("txtReport").SetText("")
```

"Copy" script:

```
' Jennessent.ReportDialog_Copy
' Jenness Enterprises <www.jennessent.com>

theText = self.GetDialog.FindByName("txtReport").GetText
theClipboard = Clipboard.The
theClipboard.Empty
theClipboard.Add(theText)
theClipboard.Update
```

"Copy and Close" script:

```
' Jennessent.ReportDialog_CopyClose
' Jenness Enterprises <www.jennessent.com>

theText = self.GetDialog.FindByName("txtReport").GetText
theClipboard = Clipboard.The
theClipboard.Empty
theClipboard.Add(theText)
theClipboard.Update

self.GetDialog.Close
```

"Run Dialog" script:

```
' Jennessent.ReportDialog_Run
' Jenness Enterprises <www.jennessent.com>

theText = self.Get(0)
theTitle = self.Get(1)
```

```
theReportDialog = av.FindDialog("Jennessent.ReportDialog")
theReportDialog.SetTitle(theTitle)
txtReport = theReportDialog.FindByName("txtReport")

txtReport.SetText(theText)
theReportDialog.Open
```

"Sample Code" script:

```
' Jennessent.ReportDialog_Sample_Code
' Jenness Enterprises <www.jennessent.com>

' First identify the script that runs the report dialog:
MakeReport = av.FindScript("Jennessent.ReportDialog_Run")

' Next generate a report and specify a title:
theText = "This is a sample report."
theTitle = "This is a sample title."

' Open your report dialog with the 'DoIt' request, with the text and
' title as the two parameters.

MakeReport.DoIt({theText, theTitle})
```

List Dialog Scripts:

"Open" script:

```
' zzz_Jennessent.SampleListBoxOpen
' Jenness Enterprises <http://www.jennessent.com>

AVUpperLeft = av.ReturnOrigin
AVCenter = avUpperLeft + (av.ReturnExtent / (2@2))
halfDialogWidthHeight = Self.ReturnExtent.ReturnSize / (2@2)
MovePoint = AVCenter - halfDialogWidthHeight
Self.MoveTo(MovePoint.GetX, MovePoint.GetY)

theDialog = self
cmdOK = theDialog.FindByName("cmdOK")
cmdCancel = theDialog.FindByName("cmdCancel")
```

"Activate" script:

```
' zzz_Jennessent.SampleListBoxActivate
' Jenness Enterprises <http://www.jennessent.com>
```

"Cancel" script:

```
' zzz_Jennessent.SampleListBoxCancel
' Jenness Enterprises <http://www.jennessent.com>

self.GetDialog.SetModalResult(nil)
self.GetDialog.Close
```

"OK" script:

```
' zzz_Jennessent.SampleListBoxOK
' Jenness Enterprises <http://www.jennessent.com>

theDialog = av.FindDialog("zzz_Jennessent.SampleListBox")
theDialog.SetModalResult(theDialog.FindByName("lbxList").GetSelection)
theDialog.Close
```

"Sample Code" script:

```
' zzz_Jennessent.SampleListBox_sample_code

' Jenness Enterprises <www.jennessent.com>

' IF THIS IS A MODAL DIALOG, YOU CAN USE IT COMPLETELY THROUGH THE zzz_Jennessent.SampleListBox_Run SCRIPT
' AS ILLUSTRATED BELOW.  OTHERWISE, LOOK AT zzz_Jennessent.SampleListBox_Run FOR A STRAIGHTFORWARD WAY TO
' PRE-LOAD THE MESSAGE AND LISTBOX.  NOTICE THAT YOU CAN INCLUDE OBJECTS OTHER THAN STRINGS AND NUMBERS

RunListDialog = av.FindScript("zzz_Jennessent.SampleListBox_Run")

theMessage = "This is a sample message to illustrate how to use this dialog.  You might want to "+
  "explain how to select multiple cells, rows or columns in the listbox below if you have set up "+
  "your list that way."

' NOTICE THE LIST CAN HAVE SUB-LISTS
theSymbol1 = RasterFill.Make
theSymbol1.SetColor(Color.GetRed)
theSymbol2 = RasterFill.Make
theSymbol2.SetColor(Color.GetBlue)
theSymbol3 = RasterFill.Make
theSymbol3.SetColor(Color.GetGreen)

theIcon = av.FindGUI("View").GetButtonBar.FindByScript("Project.Save").GetIcon.Clone

theList = {{theSymbol1,1,2,3},{4,theSymbol2,5}, {6,7,8,9,10},{12,11,theSymbol3,13},{14,15,16, theIcon}}

theTitle = "Sample Title"

theSelection = RunListDialog.DoIt({theMessage, theList, theTitle})

if (theSelection = nil) then return nil end
```

```
msgBox.Info(theSelection.Count.AsString+" objects selected. ", "Selection Report:")
```

"Run" script:

```
' zzz_Jennessent.SampleListBox_Run

' THIS SCRIPT WILL WORK IF THE LIST DIALOG IS A MODAL DIALOG

theText = self.Get(0)
theList = self.Get(1)
theTitle = self.Get(2)
theDialog = av.FindDialog("zzz_Jennessent.SampleListBox")
theDialog.FindByName("txtMessage").SetText(theText)
theDialog.FindByName("lbxList").DefineFromList(theList)
theSelection = theDialog.Open
return theSelection
```

"OK" script:

```
Jennessent.SelectProjectionDialog_cmdOKClick
' Jennessent.SelectProjectionDialog_cmdOKClick
' Jenness Enterprises <www.jennessent.com>

' DistanceByID.ProjectionDLGClose
' Activates when the 'OK' button on the Edges.SelectProjection Dialog is clicked.
' Just turns off the dialog.

theDialog = self.GetDialog

theDialog.Close
```

Sortable List Scripts:

"Close" script:

```
Jennessent.AddToListDialog_Close
' Jennessent.AddToListDialog_Close
' Jenness Enterprises <www.jennessent.com>

' Jennessent.SampleAddTolistClose

Self.SetObjectTag(nil)
Self.FindByName("cmdAdd").SetObjectTag(nil)
Self.FindByName("cmdCancel").SetObjectTag(nil)
Self.FindByName("cmdOK").SetObjectTag(nil)
Self.FindByName("cmdRemove").SetObjectTag(nil)
Self.FindByName("cmdShiftAllDown").SetObjectTag(nil)
Self.FindByName("cmdShiftAllUp").SetObjectTag(nil)
Self.FindByName("cmdShiftDown").SetObjectTag(nil)
```

```
Self.FindByName("cmdShiftUp").SetObjectTag(nil)
Self.FindByName("lbxAll").SetObjectTag(nil)
Self.FindByName("lbxSelected").SetObjectTag(nil)
```

"Add Button" script:

```
Jennessent.AddToListDialog_cmdAddClick
' Jennessent.AddToListDialog_cmdAddClick
' Jenness Enterprises <www.jennessent.com>

' Jennessent.SampleAddTolistAdd

theDialog = av.FindDialog("Jennessent.AddToListDialog")
cmdCancel = theDialog.FindByName("cmdCancel")
lbxAll = theDialog.FindByName("lbxAll")
lbxSelected = theDialog.FindByName("lbxSelected")

theAddList = lbxAll.GetSelection
theList = lbxSelected.GetList

for each aNewObject in theAddList
  FoundObject = False
  for each anExistingObject in theList
    if (anExistingObject = aNewObject) then
      FoundObject = True
      break
    end
  end
  if (FoundObject.Not) then theList.Add(aNewObject) end
end

'lbxSelected.DefineFromList(theList+theAddList)
lbxSelected.DefineFromList(theList)

lbxAll.SetSelection(Rect.MakeNull, False)
lbxAll.ShowCurrent

av.Run("Jennessent.AddToListDialog_ShuffleArrowUpdate", nil)
```

"Cancel Button" script:

```
Jennessent.AddToListDialog_cmdCancelClick
' Jennessent.AddToListDialog_cmdCancelClick
' Jenness Enterprises <www.jennessent.com>

' Jennessent.SampleAddTolistCancel

self.GetDialog.SetModalResult(nil)
self.GetDialog.Close
```

"OK Button" script:

```
Jennessent.AddToListDialog_cmdOKClick
' Jennessent.AddToListDialog_cmdOKClick
' Jenness Enterprises <www.jennessent.com>

' Jennessent.SampleAddTolistOK

theDialog = av.FindDialog("Jennessent.AddToListDialog")
lbxSelected = theDialog.FindByName("lbxSelected")

theDialog.SetModalResult(lbxSelected.GetList)
theDialog.Close
```

"Remove Button" script:

```
Jennessent.AddToListDialog_cmdRemoveClick
' Jennessent.AddToListDialog_cmdRemoveClick
' Jenness Enterprises <www.jennessent.com>

' Jennessent.SampleAddToListRemove

theDialog = av.FindDialog("Jennessent.AddToListDialog")
cmdCancel = theDialog.FindByName("cmdCancel")
lbxSelected = theDialog.FindByName("lbxSelected")

theList = lbxSelected.GetList
theCurrentRow = lbxSelected.GetCurrentRow

theList.Remove(theCurrentRow)
lbxSelected.DefineFromList(theList)

av.Run("Jennessent.AddToListDialog_ShuffleArrowUpdate", nil)
```

"Shuffle to Bottom Button" script:

```
Jennessent.AddToListDialog_cmdShiftAllDownClick
' Jennessent.AddToListDialog_cmdShiftAllDownClick
' Jenness Enterprises <www.jennessent.com>

' Jennessent.SampleAddToListShuffleDownAll

theDialog = av.FindDialog("Jennessent.AddToListDialog")
cmdCancel = theDialog.FindByName("cmdCancel")
cmdOK = theDialog.FindByName("cmdOK")
cmdShiftAllDown = theDialog.FindByName("cmdShiftAllDown")
cmdShiftAllUp = theDialog.FindByName("cmdShiftAllUp")
cmdShiftDown = theDialog.FindByName("cmdShiftDown")
```

```
cmdShiftUp = theDialog.FindByName("cmdShiftUp")
lbxSelected = theDialog.FindByName("lbxSelected")

theList = lbxSelected.GetList
theCount = theList.Count-1
theCurrentValue = lbxSelected.GetCurrentValue
theCurrentRow = lbxSelected.GetCurrentRow

theList.Shuffle(theList.Get(theCurrentRow), theCount+1)

lbxSelected.DefineFromList(theList)
lbxSelected.GoRow(theCount)
lbxSelected.SelectCurrent(False)
lbxSelected.ShowCurrent

av.Run("Jennessent.AddToListDialog_ShuffleArrowUpdate", nil)
```

"Shuffle to Top Button" script:

```
Jennessent.AddToListDialog_cmdShiftAllUpClick
' Jennessent.AddToListDialog_cmdShiftAllUpClick
' Jenness Enterprises <www.jennessent.com>

' Jennessent.SampleAddToListShuffleUpAll

theDialog = av.FindDialog("Jennessent.AddToListDialog")
cmdCancel = theDialog.FindByName("cmdCancel")
cmdOK = theDialog.FindByName("cmdOK")
cmdShiftAllDown = theDialog.FindByName("cmdShiftAllDown")
cmdShiftAllUp = theDialog.FindByName("cmdShiftAllUp")
cmdShiftDown = theDialog.FindByName("cmdShiftDown")
cmdShiftUp = theDialog.FindByName("cmdShiftUp")
lbxSelected = theDialog.FindByName("lbxSelected")

theList = lbxSelected.GetList
theCurrentValue = lbxSelected.GetCurrentValue
theCurrentRow = lbxSelected.GetCurrentRow
theList.Shuffle(theList.Get(theCurrentRow), 0)

lbxSelected.DefineFromList(theList)
lbxSelected.GoRow(0)
lbxSelected.SelectCurrent(False)
lbxSelected.ShowCurrent

av.Run("Jennessent.AddToListDialog_ShuffleArrowUpdate", nil)
```

"Shuffle Down Button" script:

```
Jennessent.AddToListDialog_cmdShiftDownClick
' Jennessent.AddToListDialog_cmdShiftDownClick
```

```
' Jenness Enterprises <www.jennessent.com>

' Jennessent.SampleAddToListShuffleDown

theDialog = av.FindDialog("Jennessent.AddToListDialog")
cmdCancel = theDialog.FindByName("cmdCancel")
cmdOK = theDialog.FindByName("cmdOK")
cmdShiftAllDown = theDialog.FindByName("cmdShiftAllDown")
cmdShiftAllUp = theDialog.FindByName("cmdShiftAllUp")
cmdShiftDown = theDialog.FindByName("cmdShiftDown")
cmdShiftUp = theDialog.FindByName("cmdShiftUp")
lbxSelected = theDialog.FindByName("lbxSelected")

theList = lbxSelected.GetList
theCurrentValue = lbxSelected.GetCurrentValue
theCurrentRow = lbxSelected.GetCurrentRow

theList.Shuffle(theList.Get(theCurrentRow), theCurrentRow +2)

lbxSelected.DefineFromList(theList)
lbxSelected.GoRow(theCurrentRow+1)
lbxSelected.SelectCurrent(False)
lbxSelected.ShowCurrent

av.Run("Jennessent.AddToListDialog_ShuffleArrowUpdate", nil)
```

"Shuffle Up Button" script:

```
Jennessent.AddToListDialog_cmdShiftUpClick
' Jennessent.AddToListDialog_cmdShiftUpClick
' Jenness Enterprises <www.jennessent.com>

' Jennessent.SampleAddToListShuffleUp

theDialog = av.FindDialog("Jennessent.AddToListDialog")
cmdCancel = theDialog.FindByName("cmdCancel")
cmdOK = theDialog.FindByName("cmdOK")
cmdShiftAllDown = theDialog.FindByName("cmdShiftAllDown")
cmdShiftAllUp = theDialog.FindByName("cmdShiftAllUp")
cmdShiftDown = theDialog.FindByName("cmdShiftDown")
cmdShiftUp = theDialog.FindByName("cmdShiftUp")
lbxSelected = theDialog.FindByName("lbxSelected")

theList = lbxSelected.GetList
theCurrentValue = lbxSelected.GetCurrentValue
theCurrentRow = lbxSelected.GetCurrentRow

theList.Shuffle(theList.Get(theCurrentRow), theCurrentRow - 1)

lbxSelected.DefineFromList(theList)
lbxSelected.GoRow(theCurrentRow-1)
```

```
lbxSelected.SelectCurrent(False)
lbxSelected.ShowCurrent

av.Run("Jennessent.AddToListDialog_ShuffleArrowUpdate", nil)
```

"Listbox Apply" script:

```
Jennessent.AddToListDialog_lbxAllApply
' Jennessent.AddToListDialog_lbxAllApply
' Jenness Enterprises <www.jennessent.com>

' Jennessent.SampleAddTolistAdd

theDialog = av.FindDialog("Jennessent.AddToListDialog")
cmdCancel = theDialog.FindByName("cmdCancel")
lbxAll = theDialog.FindByName("lbxAll")
lbxSelected = theDialog.FindByName("lbxSelected")

theAddList = lbxAll.GetSelection
theList = lbxSelected.GetList

for each aNewObject in theAddList
  FoundObject = False
  for each anExistingObject in theList
    if (anExistingObject = aNewObject) then
      FoundObject = True
      break
    end
  end
  if (FoundObject.Not) then theList.Add(aNewObject) end
end

'lbxSelected.DefineFromList(theList+theAddList)
lbxSelected.DefineFromList(theList)

lbxAll.SetSelection(Rect.MakeNull, False)
lbxAll.ShowCurrent

av.Run("Jennessent.AddToListDialog_ShuffleArrowUpdate", nil)
```

"Listbox Select" script:

```
Jennessent.AddToListDialog_lbxAllSelect
' Jennessent.AddToListDialog_lbxAllSelect
' Jenness Enterprises <www.jennessent.com>

' Jennessent.SampleAddToListSelFromAll

theDialog = av.FindDialog("Jennessent.AddToListDialog")
```

```
cmdAdd = theDialog.FindByName("cmdAdd")
cmdRemove = theDialog.FindByName("cmdRemove")
lbxAll = theDialog.FindByName("lbxAll")
lbxSelected = theDialog.FindByName("lbxSelected")

cmdAdd.SetEnabled(lbxAll.HasSelection)
cmdRemove.SetEnabled(lbxSelected.HasSelection)
```

"Listbox (Selected) Select" script:

```
Jennessent.AddToListDialog_lbxSelectedSelect
' Jennessent.AddToListDialog_lbxSelectedSelect
' Jenness Enterprises <www.jennessent.com>

' Jennessent.SampleAddTolistSelFromSel

theDialog = av.FindDialog("Jennessent.AddToListDialog")
cmdAdd = theDialog.FindByName("cmdAdd")
cmdRemove = theDialog.FindByName("cmdRemove")
lbxAll = theDialog.FindByName("lbxAll")
lbxSelected = theDialog.FindByName("lbxSelected")

cmdAdd.SetEnabled(lbxAll.HasSelection)
cmdRemove.SetEnabled(lbxSelected.HasSelection)

av.Run("Jennessent.AddToListDialog_ShuffleArrowUpdate", nil)
```

"Dialog Open" script:

```
Jennessent.AddToListDialog_Open
' Jennessent.AddToListDialog_Open
' Jenness Enterprises <www.jennessent.com>

' Jennessent.SampleAddTolistOpen

AVUpperLeft = av.ReturnOrigin
AVCenter = avUpperLeft + (av.ReturnExtent / (2@2))
halfDialogWidthHeight = Self.ReturnExtent.ReturnSize / (2@2)
MovePoint = AVCenter - halfDialogWidthHeight
Self.MoveTo(MovePoint.GetX, MovePoint.GetY)

theDialog = self
cmdOK = theDialog.FindByName("cmdOK")
cmdCancel = theDialog.FindByName("cmdCancel")
lbxAll = theDialog.FindByName("lbxAll")
cmdAdd = theDialog.FindByName("cmdAdd")
cmdRemove = theDialog.FindByName("cmdRemove")
lbxSelected = theDialog.FindByName("lbxSelected")
```

```
theList = theDialog.GetObjectTag
lbxAll.DefineFromList(theList)

cmdAdd.SetEnabled(False)
cmdRemove.SetEnabled(False)
lbxSelected.Empty

av.Run("Jennessent.AddToListDialog_ShuffleArrowUpdate", nil)
```

"Sample Code" script:

```
Jennessent.AddToListDialog_SampleCode
' Jennessent.AddToListDialog_SampleCode
' Jenness Enterprises <www.jennessent.com>


theDialog = av.FindDialog("Jennessent.AddToListDialog")
theList = {1,2,3,4,5,6,7,8,9,0}

theDialog.SetObjectTag(thelist)
theOutput = theDialog.Open
if (theOutput = nil) then return nil end
msgBox.ListAsString(theOutput, theOutput.Count.AsString+" objects chosen...", "Test")
```

"Shuffle Update" script:

```
Jennessent.AddToListDialog_ShuffleArrowUpdate
' Jennessent.AddToListDialog_ShuffleArrowUpdate
' Jenness Enterprises <www.jennessent.com>

' Jennessent.AddToListDialog_ShuffleArrowUpdate

theDialog = av.FindDialog("Jennessent.AddToListDialog")
cmdCancel = theDialog.FindByName("cmdCancel")
cmdOK = theDialog.FindByName("cmdOK")
cmdShiftAllDown = theDialog.FindByName("cmdShiftAllDown")
cmdShiftAllUp = theDialog.FindByName("cmdShiftAllUp")
cmdShiftDown = theDialog.FindByName("cmdShiftDown")
cmdShiftUp = theDialog.FindByName("cmdShiftUp")
lbxSelected = theDialog.FindByName("lbxSelected")

theRow = lbxSelected.GetCurrentRow
theCount = lbxSelected.GetList.Count-1

NotAtEnd = (theRow = theCount).Not
NotAtBeg = (theRow = 0).Not

ShouldEnable = lbxSelected.HasSelection
cmdShiftAllDown.SetEnabled(ShouldEnable and NotAtEnd)
cmdShiftAllUp.SetEnabled(ShouldEnable and NotAtBeg)
```

```
cmdShiftDown.SetEnabled(ShouldEnable and NotAtEnd)
cmdShiftUp.SetEnabled(ShouldEnable and NotAtBeg)

cmdOK.SetEnabled(lbxSelected.GetList.Count > 0)
```

Select Projection Scripts:

"Close" script:

```
Jennessent.SelectProjectionDialog_Close
' Jennessent.SelectProjectionDialog_Close
' Jenness Enterprises <www.jennessent.com>

' Jennessent.SelectProjectionDialogClose

Self.SetObjectTag(nil)
Self.FindByName("cmdOK").SetObjectTag(nil)
Self.FindByName("lblProjection").SetObjectTag(nil)
Self.FindByName("lblResultsProjection").SetObjectTag(nil)
Self.FindByName("optGeoCurve").SetObjectTag(nil)
Self.FindByName("optProjection").SetObjectTag(nil)
Self.FindByName("optUnprojected").SetObjectTag(nil)
```

"Cancel" script:

```
Jennessent.SelectProjectionDialog_cmdCancelClick
' Jennessent.SelectProjectionDialog_cmdCancelClick
' Jenness Enterprises <www.jennessent.com>

' Jennessent.SelectProjectionDialog_Cancel

self.GetDialog.SetModalResult(nil)
self.GetDialog.Close
```

"OK" script:

```
Jennessent.SelectProjectionDialog_cmdOKClick
' Jennessent.SelectProjectionDialog_cmdOKClick
' Jenness Enterprises <www.jennessent.com>

' Jennessent.SelectProjectionDialog_cmdOKClick
' Jenness Enterprises <www.jennessent.com>

' Just turns off the dialog.

theDialog = self.GetDialog

theDialog.Close
```

"Open" script:

```
Jennessent.SelectProjectionDialog_Open
' Jennessent.SelectProjectionDialog_Open
' Jenness Enterprises <www.jennessent.com>

' Jennessent.SelectProjectionDialog_Open
' Jenness Enterprises <www.jennessent.com>

theProject = av.GetProject
theView = av.GetActiveDoc

AVUpperLeft = av.ReturnOrigin
AVCenter = avUpperLeft + (av.ReturnExtent / (2@2))
aDialog =  self
halfDialogWidthHeight = aDialog.ReturnExtent.ReturnSize / (2@2)
MovePoint = AVCenter - halfDialogWidthHeight
aDialog.MoveTo(MovePoint.GetX, MovePoint.GetY)

' Output Data will be either TRUE or FALSE depending on whether the user wants the output
' data to be in Projected (TRUE) or Geographic (FALSE) coordinates.  TRUE is the default.

theProjectionName = aDialog.GetObjectTag.AsString

optProjection = aDialog.FindByName ("optProjection")
optProjection.Select
optProjection.SetLabel (theProjectionName + " Projection")

lblProjection = aDialog.FindByName ("lblProjection")

lblText = "Your original data are unprojected, but your View has been" + NL +
  "projected into the " + theProjectionName + " projection." + NL + " " + NL +
  "Do you wish to calculate your RESULTS data based on this " + NL +
  "projection?  If your themes are both Point themes, you may"+NL+
  "calculate Great Circle distances (most accurate)."

lblProjection.Setlabel (lblText)

optGeoCurve = aDialog.FindByName("optGeoCurve")
optGeoCurve.SetEnabled(optGeoCurve.GetObjectTag = True)

aDialog.SetModalResult(TRUE)

cmdOK = aDialog.FindByName ("cmdOK")
aDialog.SetDefaultButton (cmdOK)
```

"Select Great Circle" script:

```
Jennessent.SelectProjectionDialog_optGeoCurveClick
```

```
' Jennessent.SelectProjectionDialog_optGeoCurveClick
' Jenness Enterprises <www.jennessent.com>

' Jennessent.SelectProjectionDialog_optGreatCircleClick
' Jenness Enterprises <www.jennessent.com>

' Runs when the Great Circle option is selected on the
' DistanceByID.SelectProjection dialog box is clicked.  Sets
' the Modal Result to TRUE, meaning the user wants the
' results data projected.

aDialog = Self.GetDialog
aDialog.SetModalResult("Great Circle")
```

"Select Projected" script:

```
Jennessent.SelectProjectionDialog_optProjectionClick
' Jennessent.SelectProjectionDialog_optProjectionClick
' Jenness Enterprises <www.jennessent.com>

' Jennessent.SelectProjectionDialog_optProjectionClick
' Jenness Enterprises <www.jennessent.com>

' Runs when the PROJECTED option is selected on the
' DistanceByID.SelectProjection dialog box is clicked.  Sets
' the Modal Result to TRUE, meaning the user wants the
' results data projected.

aDialog = Self.GetDialog
aDialog.SetModalResult(TRUE)
```

"Select Unprojected" script:

```
Jennessent.SelectProjectionDialog_optUnprojectedClick
' Jennessent.SelectProjectionDialog_optUnprojectedClick
' Jenness Enterprises <www.jennessent.com>

' Jennessent.SelectProjectionDialog_optUnprojectedClick
' Jenness Enterprises <www.jennessent.com>

' Runs when the PROJECTED option is selected on the
' DistanceByID.SelectProjection dialog box is clicked.  Sets
' the Modal Result to FALSE, meaning the user wants the
' results data unprojected.

aDialog = Self.GetDialog
aDialog.SetModalResult(FALSE)
```

"Sample Code" script:

```
Jennessent.SelectProjectionDialog_SampleCode
' Jennessent.SelectProjectionDialog_SampleCode
' Jenness Enterprises <www.jennessent.com>

theSelPrjDialog = av.FindDialog("Jennessent.SelectProjectionDialog")
IncludeGreatCircle = True        ' SET TO FALSE TO DISABLE GREAT CIRCLE OPTION
theSelPrjDialog.FindByName("optGeoCurve").SetObjectTag(IncludeGreatCircle)
ShouldProject = theSelPrjDialog.Open    ' TRUE, FALSE OR "GREAT CIRCLE"
msgBox.Info(ShouldProject.AsString, "")
```

Create VTab and FTab scripts:

"Create VTab" script:

```
'-----------------------------------------------------------------------
'<<<<<<| CREATE EMPTY TABLE |>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>
'-----------------------------------------------------------------------

' CREATE TABLE:  I CHECK FOR OPERATING SYSTEM BECAUSE THE FILE DIALOG BOX IN
' WINDOWS WORKS BETTER WHEN USING THE '\' SYMBOL RATHER THAN THE '/' SYMBOL
' IN THIS OPERATION.

if (theOS = #SYSTEM_OS_MSW) then
  tempFileString = "\results.dbf"
else
  tempFileString = "/results.dbf"
end
tempFileName = FileName.Make(theWorkDirStr+tempFileString)
tempFNCounter = 1

' SUGGEST FILE NAME AND DIRECTORY, BUT NOT ONE THAT ALREADY EXISTS

While (File.Exists(tempFileName))
  tempFNCounter = tempFNCounter + 1
  if (theOS = #SYSTEM_OS_MSW) then
    tempFileString = "\results"+tempFNCounter.AsString+".dbf"
  else
    tempFileString = "/results"+tempFNCounter.AsString+".dbf"
  end
  tempFileName = FileName.Make(theWorkDirStr+tempFileString)
  if (File.Exists(tempFileName).Not) then
    break
  end
end

theFilename  = FileDialog.Put(tempFilename, "*.dbf", "Please specify a name for your table:")

if (theFilename = nil) then
  msgBox.info ("No Table created:  Exiting routine...", "Problem:")
```

```
  return nil
end

theFilename.SetExtension("dbf")
theNewTable = VTab.MakeNew (theFilename, dBASE)
theIDField = Field.Make("ID", #FIELD_LONG, 6, 0)
theNewTable.AddFields({theIDField})
```

"Create FTab" script:

```
'-------------------------------------------------------------------------
'<<<<<<| CREATE EMPTY FTAB/SHAPEFILE |>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>
'-------------------------------------------------------------------------

' CREATE FTAB:  I CHECK FOR OPERATING SYSTEM BECAUSE THE FILE DIALOG BOX IN
' WINDOWS WORKS BETTER WHEN USING THE '\' SYMBOL RATHER THAN THE '/' SYMBOL
' IN THIS OPERATION.

if (theOS = #SYSTEM_OS_MSW) then
  tempFileString = "\random_points.shp"
else
  tempFileString = "/random_points.shp"
end
tempFileName = FileName.Make(theWorkDirStr+tempFileString)
tempFNCounter = 1

' SUGGEST FILE NAME AND DIRECTORY, BUT NOT ONE THAT ALREADY EXISTS

While (File.Exists(tempFileName))
  tempFNCounter = tempFNCounter + 1
  if (theOS = #SYSTEM_OS_MSW) then
    tempFileString = "\random_points"+tempFNCounter.AsString+".shp"
  else
    tempFileString = "/random_points"+tempFNCounter.AsString+".shp"
  end
  tempFileName = FileName.Make(theWorkDirStr+tempFileString)
  if (File.Exists(tempFileName).Not) then
    break
  end
end

theFilename  = FileDialog.Put(tempFilename, "*.shp", "Please specify a name for your shapefile:")

if (theFilename = nil) then
  msgBox.info ("No Shapefile created:  Exiting routine...", "Problem:")
  return nil
end

theFilename.SetExtension("shp")
theNewPointFTab = FTab.MakeNew (theFilename, Point)
theNewPointShapeField = theNewPointFTab.FindField("Shape")
```

```
theIDField = Field.Make("ID", #FIELD_LONG, 6, 0)
theNewPointFTab.AddFields({theIDField})
```

Generate Random Number scripts:

Generate Random Numbers:

```
' Jennessent.MakeRandomNum
' Jenness Enterprises <www.jennessent.com)>
' Adapted from code suggested by Bill Huber [Quantitative Decisions <whuber@quantdec.com]
' Given a range and a precision, this scripts returns a random number within that range
' with the specified number of decimal places.

theMin = self.Get(0)
theMax = self.Get(1)
thePrecision = self.Get(2)

theExponent = 10^thePrecision
nBig = 2^30     ' FROM BILL HUBER - Determines granularity, but cannot be larger than 2^31 - 1

theRandom = Number.MakeRandom(0, nBig)/nBig * (theMax - theMin) + theMin
Return ((theRandom * theExponent).Round)/theExponent
```

Generate Random Numbers Sample Code:

```
' CougarRandom.MakeRandomNum_sample
' Jenness Enterprises <www.jennessent.com)>

' To generate a random integer between 1 and 100, first identify the randomize script and
' run it as follows:

Jennessent = av.FindScript("CougarRandom.MakeRandomNum")
theMin = 1
theMax = 100
thePrecision = 0
theRandomNumber = CalcRandom.DoIt({theMin, theMax, thePrecision})

msgBox.Info("RandomNumber = "+theRandomNumber.SetFormatPrecision(thePrecision).AsString, "Random Number Generation Successful:")
```

Generate Normally-Distributed Random Number scripts:

Generate Normally Distributed Random Numbers:

```
' Jennessent.MakeNormRandomNum
' Jenness Enterprises <www.jennessent.com)>
' Randomization Code Adapted from code suggested by Bill Huber [Quantitative Decisions <whuber@quantdec.com]
```

```
' Given a mean and standard deviation, this script returns two random numbers within that distribution.
' Based on the Box-Muller Transformation:.

'          y1 = sqrt( - 2 ln(x1) ) cos( 2 pi x2 )
'          y2 = sqrt( - 2 ln(x1) ) sin( 2 pi x2 )
'        where:
'          x1 = first uniform random number (between 0 and 1)
'          x2 = second uniform random number (between 0 and 1)
'          y1 = first normally distributed random number
'          y2 = second normally distributed random number.

theMean = self.Get(0)
theSD = self.Get(1)
thePi = Number.GetPi
nBig = 2^30     ' FROM BILL HUBER - Determines granularity, but cannot be larger than 2^31 - 1

theRandNum1 = (Number.MakeRandom(0, nBig)/nBig)
theRandNum2 = (Number.MakeRandom(0, nBig)/nBig)

theNorm1 = (theSD*(((-2)*(theRandNum1.ln)).sqrt)*((2*thePi*theRandNum2).Cos))+theMean
theNorm2 = (theSD*(((-2)*(theRandNum1.ln)).sqrt)*((2*thePi*theRandNum2).Sin))+theMean

return {theNorm1, theNorm2}
```

Generate Normally Distributed Random Numbers Sample Code:

```
' Jennessent.MakeNormRandomNum_sample
' Jenness Enterprises <www.jennessent.com)>

' To generate a random integer between 1 and 100, first identify the randomize script and
' run it as follows:

CalcNormRandom = av.FindScript("Jennessent.MakeNormRandomNum")
theMean = 100
theSD = 10
theRandomNumbers = CalcNormRandom.DoIt({theMean, theSD})

msgBox.ListAsString(theRandomNumbers, "Two Random Numbers:"+NL+
    "Normal Distribution:  Mean = 100, SD = 10", "Random Number Generation Successful:")
```

Generate ‘Insert Commas in Number’ scripts:

Insert Commas in Number:

```
' Jennessent.InsertCommas
' Jenness Enterprises <www.jennessent.com)>
' Takes a string version of a number and inserts commas into it.
```

```
theAreaString = self.Get(0)
thePrecision = self.Get(1)

if (theAreaString.Is(Number)) then
  theAreaString = theAreaString.Clone.SetFormatPrecision(thePrecision).AsString
else
  theAreaString = theAreaString.AsNumber.SetFormatPrecision(thePrecision).AsString
end

theTokens = theAreaString.AsTokens(".")
theBaseNumber = theTokens.Get(0)
theCount = theBaseNumber.Count
theCommaString = ""

if (theCount > 3) then
  for each anIndex in (theCount-3)..0 by -3
    theCommaString = theBaseNumber.Middle(anIndex, 3)+","+theCommaString
    if (anIndex < 3) then theCommaString = theBaseNumber.Left(anIndex)+","+theCommaString end
  end
  theCommaString = theCommaString.BasicTrim(",",",")
else
  theCommaString = theBaseNumber
end

if (theTokens.Count > 1) then
  theCommaString = theCommaString+"."+theTokens.Get(1)
end

if (theCommaString.Contains(".")) then
  theCommaString = theCommaString.BasicTrim("", "0")
  theCommaString = theCommaString.BasicTrim("", ".")
end

return theCommaString
```

Insert Commas in Number Sample Code:

```
' saguaro.InsertCommas_sample
' Jenness Enterprises <www.jennessent.com)>

' To insert commas into the number 123456789.123, first identify the script and
' run it as follows:

AddCommas = av.FindScript("saguaro.InsertCommas")
theNumAsString = "123456789.123"
theNumWithCommas = AddCommas.DoIt(theNumAsString)

msgBox.Info("Original Number = "+theNumAsString+NL+"the Number with commas = "+theNumWithCommas, "Comma Insertion Successful:")
```

Make Measurement Unit Dictionaries script:

Make measurement unit dictionaries:

```
' Jennessent.MakeMeasureUnits
' Jenness Enterprises <www.jennessent.com>

theUnitsDictionary = Dictionary.Make(11)
theUnitsDictionary.Set(#UNITS_LINEAR_UNKNOWN, "Unknown")
theUnitsDictionary.Set(#UNITS_LINEAR_INCHES, "Inches")
theUnitsDictionary.Set(#UNITS_LINEAR_FEET, "US Survey Feet")
theUnitsDictionary.Set(#UNITS_LINEAR_YARDS, "Yards")
theUnitsDictionary.Set(#UNITS_LINEAR_MILES, "Miles")
theUnitsDictionary.Set(#UNITS_LINEAR_MILLIMETERS, "Millimeters")
theUnitsDictionary.Set(#UNITS_LINEAR_CENTIMETERS, "Centimeters")
theUnitsDictionary.Set(#UNITS_LINEAR_METERS, "Meters")
theUnitsDictionary.Set(#UNITS_LINEAR_KILOMETERS, "Kilometers")
theUnitsDictionary.Set(#UNITS_LINEAR_NAUTICALMILES, "Nautical miles")
theUnitsDictionary.Set(#UNITS_LINEAR_DEGREES, "Decimal degrees")

theNamesDictionary = Dictionary.Make(11)
for each aKey in theUnitsDictionary.ReturnKeys
  theNamesDictionary.Set(theUnitsDictionary.Get(aKey), aKey)
end

return {theUnitsDictionary, theNamesDictionary}
```

Geometric Function scripts:

Sort points according to X or Y value:

```
' xxxCalcSortByXorY
'Jenness Enterprises <www.jennessent.com>
theList = Self

theList.RemoveDuplicates

av.ShowMsg("Sorting points...")

theXList = {}

for each aPoint in theList
  theXList.Add(aPoint.GetX)
end

theXList.RemoveDuplicates

theXList.Sort(True)
```

```
' ADD X-VALUES TO DICTIONARY AS KEYS
theCounter = 0
theCount = theList.Count

theDictionary = Dictionary.Make(theXList.Count)
for each anX in theXList
  theCounter = theCounter+1
  av.SetStatus((theCounter/theCount)*100)
  theDictionary.Set(anX, {})
end

' ADD POINTS TO DICTIONARY AS ELEMENTS
theCounter = 0
theCount = theList.Count

for each aPoint in theList
  theCounter = theCounter+1
  av.SetStatus((theCounter/theCount)*100)
  theShortList = theDictionary.Get(aPoint.GetX)
  theShortList.Add(aPoint)

  ' SHUFFLE POINT TO APPROPRIATE PLACE IN LIST (SORTED LOW TO HIGH)
  theIndex = theShortList.Count-1
  while ((theIndex > 0) and (theShortList.Get(theIndex).GetY < theShortList.Get(theIndex-1).GetY))
    theShortList.Shuffle(aPoint, theIndex-1)
    theIndex = theIndex -1
  end
end

return {theXList, theDictionary}
```

Sort points according to bearing from a point:

```
' Jennessent.GeometrySortClockwise
' Jenness Enterprises <www.jennessent.com>

PointA = self.Get(0)
theList = self.Get(1)

theList.RemoveDuplicates

av.ShowMsg("Sorting points...")

theXList = {}

theDictionary = Dictionary.Make(theList.Count)

theCounter = 0
theCount = theList.Count
```

```
for each PointB in theList
  theCounter = theCounter+1
  av.SetStatus((theCounter/theCount)*100)

  if (PointA <> PointB) then

    xdist = (PointA.GetX - PointB.GetX)
    ydist = (PointA.GetY - PointB.GetY)

    xyTanDeg = (xdist/ydist).Atan.AsDegrees

    if (ydist >= 0) then
      theBearing = 180 + xytandeg
    else
      if (xdist <= 0) then
        theBearing = xytandeg
      else
        theBearing = 360+xytandeg
      end
    end ' END CALCULATING BEARING

    theBearing = theBearing.Abs
    theBearing = ((theBearing*1000).Round)/1000

    theShortList = theDictionary.Get(theBearing)

    if (theShortList = nil) then
      theXList.Add(theBearing)
      theDictionary.Set(theBearing, {PointB})
    else

      theShortList.Add(PointB)

      ' SHUFFLE POINT TO APPROPRIATE PLACE IN LIST (SORTED LOW TO HIGH)
      theIndex = theShortList.Count-1
      while ((theIndex > 0) and (theShortList.Get(theIndex).Distance(PointA) < theShortList.Get(theIndex-1).Distance(PointA)))
        theShortList.Shuffle(PointB, theIndex-1)
        theIndex = theIndex -1
      end
    end
  end
end

theXList.Sort(True)
' LOOK FOR LARGEST GAP

theLowBearing = theXList.Get(0)
theHighBearing = theXList.Get(theXList.Count-1)

theMaxGap = 360 - (theHighBearing - theLowBearing)
theMaxGapIndex = 0
```

```
for each anIndex in 0..(theXList.Count-2)
  theTestGap = theXList.Get(anIndex+1) - theXList.Get(anIndex)
  if (theTestGap > theMaxGap) then
    theMaxGapIndex = anIndex+1
    theMaxGap = theTestGap
  end
end

if (theMaxGapIndex <> 0) then
  theNewXList = {}
  for each anIndex in theMaxGapIndex..(theXList.Count-1)
    theNewXList.Add(theXList.Get(anIndex))
  end
  for each anIndex in 0..(theMaxGapIndex-1)
    theNewXList.Add(theXList.Get(anIndex))
  end
else
  theNewXList = theXList
end

theLowEnd = theNewXList.Get(0)
theHighEnd = theNewXList.Get(theNewXList.Count-1)

if (theHighEnd > theLowEnd) then
  theRange = theHighEnd - theLowEnd
else
  theRange = (theHighEnd+360) - theLowEnd
end

return {theNewXList, theDictionary, theRange}
```

"Calculate Bearing" script:

```
' Sample.CalcBearing
' Jenness Enterprises <www.jennessent.com>
' Given 2 consecutive points, this scripts the bearing of the line extending from the first
' point to the second point.

PointA = self.Get(0)
PointB = self.Get(1)

' FOR DEBUGGING
'PointA = Point.Make(14.19, 5)
'PointB = Point.Make(9.9, 10)

xdist = (PointA.GetX - PointB.GetX)
ydist = (PointA.GetY - PointB.GetY)
xyTanDeg = (xdist/ydist).Atan.AsDegrees

if (ydist >= 0) then
  theBearing = 180 + xytandeg
```

```
else
  if (xdist <= 0) then
    theBearing = xytandeg
  else
    theBearing = 360+xytandeg
  end
end ' END CALCULATING BEARING

theBearing = theBearing.Abs

' FOR DEBUGGING
'msgBox.Report("Bearing = "+theBearing.AsString+NL+"X-dist = "+(-xDist).AsString
'  +NL+"Y-dist = "+(-ydist).AsString+NL+"ArcTan = "+xyTanDeg.AsString, "")

return theBearing
```

"Check Clockwise" script:

```
' Sample.CalcCheckClockwise
' Jenness Enterprises <www.jennessent.com)>
' Given 3 consecutive points, this scripts calculates whether the third point lies to the right
' (clockwise) or to the left (counter-clockwise) of the line connecting the first point to
' the second point.

thePX = self.Get(0).GetX
thePY = self.Get(0).GetY
theQX = self.Get(1).GetX
theQY = self.Get(1).GetY
theRX = self.Get(2).GetX
theRY = self.Get(2).GetY

' CLOCKWISE IF TRUE
return ((theQX * (theRY - thePY)) + (theQY * (thePX - theRX)) - ((thePX)*(theRY)) + ((thePY)*(theRX)) < 0)
```

"Find Closest Points" script:

```
Jennessent.GeometryFindClosestPoints
' Jennessent.GeometryFindClosestPoints
' Jenness Enterprises (www.jennessent.com)
' Given two shapes, this script return the line connecting the closest points on each shape.

theInputShape = self.Get(0)
CompShape = self.Get(1)

InputIsMultipoint = (theInputShape.Is(Multipoint)) and (theInputShape.Is(Polyline).Not) and (theInputShape.Is(Polygon).Not) and
  (theInputShape.Is(Rect).Not)

CompIsMultipoint = (CompShape.Is(Multipoint)) and (CompShape.Is(Polyline).Not) and (CompShape.Is(Polygon).Not) and
  (CompShape.Is(Rect).Not)
```

```
' INITIAL CHECK TO SEE IF SHAPES INTERSECT
if (theInputShape.Distance(CompShape) = 0) then
  theIntLine = Line.MakeNull
else

  ' CALCULATE MAXIMUM POSSIBLE DISTANCE TO SERVE AS STARTING POINT; THIS IS THE LENGTH
  ' OF THE DIAGONAL EXTENDING ACROSS THE EXTENT OF ALL THEMES USED IN THE ANALYSIS

  theFullExtent = (theInputShape.ReturnExtent.UnionWith(CompShape.ReturnExtent)).Scale(1.1)
  theLongestLength = theFullExtent.ReturnLength
  theCompMinDistance = theLongestLength

  ' IF THE INPUT THEME ISN'T A POINT, GET A LIST OF THE LINES THAT MAKE UP THE SHAPE

  if (theInputShape.Is(point).Not) then

    thePointX = theInputShape.ReturnCenter.GetX
    thePointY = theInputShape.ReturnCenter.GetY

    InputShapeLines = {}

    if (InputIsMultipoint) then
      theInputPointList = theInputShape.AsList
    else
      ' ---------------------- START:  MODIFED ON NOVEMBER 10, 2000 ----------------------------------------------------

      theInputPolyline = theInputShape.AsPolyline
      thePolyLineCount = theInputPolyLine.CountParts

      if (thePolyLineCount = 1) then      ' IF IT'S A SIMPLE POLYGON OR A SINGLE-PART POLYLINE
        theInputPointList = theInputShape.AsPolyline.AsMultiPoint.AsList
        for each theInputPointIndex in 0..(theInputPointList.Count -2)
          theInputPointA = theInputPointList.Get(theInputPointIndex)
          theInputPointB = theInputPointList.Get(theInputPointIndex+1)
          theInputLineSegment = Line.Make (theInputPointA, theInputPointB)
          InputShapeLines.Add (theInputLineSegment)
        end
      else                                ' IF IT'S A COMPLEX (MULTIPART OR HAS HOLES) POLYGON OR MULTIPART POLYLINE
        theInputPolyLineList = theInputPolyLine.AsList
        for each theInputLinePointsList in theInputPolyLineList
          for each theInputPointIndex in 0..(theInputLinePointsList.Count -2)
            theInputPointA = theInputLinePointsList.Get(theInputPointIndex)
            theInputPointB = theInputLinePointsList.Get(theInputPointIndex+1)
            theInputLineSegment = Line.Make (theInputPointA, theInputPointB)
            InputShapeLines.Add (theInputLineSegment)
          end
        end                               ' END WORKING THROUGH SEPARATE LINES IN POLYLINE
      end                                 ' END CHECKING TO SEE IF IT'S A COMPLEX POLYGON OR MULTIPART POLYLINE

      ' ---------------------- END:  MODIFED ON NOVEMBER 10, 2000 ------------------------------------------
```

```
  end

end                                  ' END IDENTIFYING INPUT SHAPE

if (CompShape.Is(Point).Not) then

' IF THE COMPARISON THEME IS A POLYGON, POLYLINE OR LINE, CREATE A LIST OF LINES THAT MAKE UP THE SHAPE

  if (CompIsMultipoint) then
    theCompPointList = CompShape.AsList
  else
    CompShapeLines = {}

    ' ---------------------- START:  MODIFED ON NOVEMBER 10, 2000 ------------------------------------------------

    theComparePolyline = CompShape.AsPolyline
    theComparePolyLineCount = theComparePolyLine.CountParts

    if (theComparePolyLineCount = 1) then      ' IF IT'S A SIMPLE POLYGON OR A SINGLE-PART POLYLINE

      thePointList = CompShape.AsPolyLine.AsMultiPoint.AsList
      for each thePointIndex in 0..(thePointList.Count -2)
        theCompPointA = thePointList.Get(thePointIndex)
        theCompPointB = thePointList.Get(thePointIndex+1)
        theCompLineSegment = Line.Make (theCompPointA, theCompPointB)
        CompShapeLines.Add (theCompLineSegment)
      end

    else                                 ' IF IT'S A COMPLEX (MULTIPART OR HAS HOLES) POLYGON OR MULTIPART POLYLINE
      theComparePolyLineList = theComparePolyLine.AsList
      for each thePointList in theComparePolyLineList

        for each thePointIndex in 0..(thePointList.Count -2)
          theCompPointA = thePointList.Get(thePointIndex)
          theCompPointB = thePointList.Get(thePointIndex+1)
          theCompLineSegment = Line.Make (theCompPointA, theCompPointB)
          CompShapeLines.Add (theCompLineSegment)
        end

      end                                ' END WORKING THROUGH SEPARATE LINES IN POLYLINE
    end                                  ' END CHECKING TO SEE IF IT'S A COMPLEX POLYGON OR MULTIPART POLYLINE

    ' ---------------------- END:  MODIFED ON NOVEMBER 10, 2000 --------------------------------------------
  end

end                                  ' END IDENTIFYING COMPARISON SHAPE

' IDENTIFY CLOSEST SEGMENTS OF INPUT AND COMPARISON SHAPES
if (CompShape.Is(Point).Not) then

  if (CompIsMultipoint.Not) then
```

```
  for each theCompLineTest in CompShapeLines

  ' IF THE INPUT SHAPE <IS NOT> A POINT, THEN COMPARE EACH OF THE LINES IN THE COMPARISON SHAPE WITH
  ' EACH OF THE LINES IN THE INPUT SHAPE.  THE RESULT WILL BE THE TWO LINES FROM THE TWO SHAPES THAT
  ' ARE CLOSEST TO EACH OTHER.

    if (theInputShape.Is(Point).Not) then
      for each theInputLineTest in InputShapeLines
        theCompDistance = theCompLineTest.Distance(theInputLineTest)

        if (theCompMinDistance > theCompDistance) then
          theCompMinDistance = theCompDistance
          theCompLine = theCompLineTest
          theInputLine = theInputLineTest

        end     ' NOW THE SCRIPT KNOWS WHICH LINES ARE CLOSEST TO EACH OTHER.

      end ' END WORKING THROUGH ALL THE LINES IN THE INPUT SHAPE

    else

    ' IF THE INPUT SHAPE <IS> A POINT:  RESULT WILL BE A LINE FROM THE COMPARISON SHAPE AND A POINT FROM
    ' THE INPUT SHAPE
      theInputLine = Line.MakeNull
      theCompDistance = theCompLineTest.Distance(theInputShape)
      if (theCompMinDistance > theCompDistance) then
        theCompMinDistance = theCompDistance
        theCompLine = theCompLineTest

      end       ' END ASSIGNING VARIABLES BASED ON CLOSEST LINE
    end         ' END CHECKING IF INPUT SHAPE IS A POINT OR NOT
  end           ' END WORKING THROUGH ALL THE LINES IN THE COMPARISON SHAPE
else
  for each theCompPointTest in theCompPointLIst

  ' IF THE INPUT SHAPE <IS NOT> A POINT, THEN COMPARE ALL OF THE POINTS IN THE COMPARISON SHAPE WITH
  ' EACH OF THE LINES IN THE INPUT SHAPE.  THE RESULT WILL BE THE POINT FROM THE MULTIPOINT AND A LINE
  ' FROM THE INPUT SHAPE THAT ARE CLOSEST TO EACH OTHER.

    if (theInputShape.Is(Point).Not) then
      if (InputIsMultipoint.Not) then
        for each theInputLineTest in InputShapeLines
          theCompDistance = theCompPointTest.Distance(theInputLineTest)

          if (theCompMinDistance > theCompDistance) then
            theCompMinDistance = theCompDistance
            CompShape = theCompPointTest
            theInputLine = theInputLineTest

          end     ' NOW THE SCRIPT KNOWS WHICH SHAPES ARE CLOSEST TO EACH OTHER.
```

```
        end ' END WORKING THROUGH ALL THE LINES IN THE INPUT SHAPE
      else
        for each theInputPointTest in theInputPointList
          theCompDistance = theCompPointTest.Distance(theInputPointTest)

          if (theCompMinDistance > theCompDistance) then
            theCompMinDistance = theCompDistance
            CompShape = theCompPointTest
            theInputShape = theInputPointTest
          end
        end
      end

    else

    ' IF THE INPUT SHAPE <IS> A POINT:  RESULT WILL BE A POINT FROM THE COMPARISON SHAPE AND A POINT FROM
    ' THE INPUT SHAPE
      theInputLine = Line.MakeNull
      theCompDistance = theCompPointTest.Distance(theInputShape)
      if (theCompMinDistance > theCompDistance) then
        theCompMinDistance = theCompDistance
        CompShape = theCompPointTest

      end       ' END ASSIGNING VARIABLES BASED ON CLOSEST LINE
    end         ' END CHECKING IF INPUT SHAPE IS A POINT OR NOT
  end           ' END WORKING THROUGH ALL THE LINES IN THE COMPARISON SHAPE


end
else            ' IF THE COMPARISON SHAPE IS A POINT AND THE INPUT SHAPE IS NOT

  theCompLine = Line.MakeNull
  if (theInputShape.Is(Point).Not) then
    if (InputIsMultipoint.Not) then
      for each theInputLineTest in InputShapeLines
        theCompDistance = theInputLineTest.Distance(CompShape)

        if (theCompMinDistance > theCompDistance) then
          theCompMinDistance = theCompDistance
          theInputLine = theInputLineTest

        end     ' END ASSIGNING VARIABLES BASED ON CLOSEST LINE.  NOW THE SCRIPT KNOWS WHICH LINES
                ' ARE CLOSEST TO EACH OTHER.
      end       ' END WORKING THROUGH ALL THE LINES IN THE INPUT SHAPE
    else
      for each theInputPointTest in theInputPointList
        theCompDistance = theInputPointTest.Distance(CompShape)

        if (theCompMinDistance > theCompDistance) then
          theCompMinDistance = theCompDistance
          theInputShape = theInputPointTest
```

```
        end     ' END ASSIGNING VARIABLES BASED ON CLOSEST LINE.  NOW THE SCRIPT KNOWS WHICH LINES
                ' ARE CLOSEST TO EACH OTHER.
      end       ' END WORKING THROUGH ALL THE LINES IN THE INPUT SHAPE
    end
  end         ' END CHECKING SHAPE TYPE OF INPUT SHAPE.
end           ' END CALCULATING CLOSEST EDGE POINTS AND LINES.  LAST OPTION IS THAT THEY ARE BOTH
              ' POINTS, AND CALCULATIONS ARE EASY FOR THAT CASE.

' IDENTIFY START/END POINTS OF CLOSEST INPUT AND COMPARISON SEGMENTS

if (theInputShape.Is(Point).Not) then
  theInputPointOne = theInputLine.ReturnStart
  theInputPointTwo = theInputLine.ReturnEnd
end
if (CompShape.Is(Point).Not) then
  theCompPointOne = theCompLine.ReturnStart
  theCompPointTwo = theCompLine.ReturnEnd
end

' CALCULATE THE SLOPE AND Y-INTERCEPT OF THAT CLOSEST LINE (LINE A), THEN THE SLOPE AND Y-INTERCEPT OF THE LINE
' PERPENDICULAR TO [LINE A] THAT PASSES THROUGH THE INPUT POINT (LINE B).  THEN SEE IF [LINE B] INTERSECTS
' [LINE A] BETWEEN THE TWO ENDPOINTS OF [LINE A].  IF IT DOES, THEN THE INTERSECTION OF [LINE A] AND [LINE B]
' IS THE CLOSEST POINT BETWEEN THE FEATURES.  IF IT DOESN'T, THEN ONE OF THE TWO ENDPOINTS OF [LINE A] IS THE
' CLOSEST POINT.
'
' NEED TO DO THIS BASED ON ONE OF FOUR SITUATIONS:
' 1) INPUT SHAPE = POINT        COMPARISON SHAPE <> POINT
' 2) INPUT SHAPE <> POINT       COMPARISON SHAPE <> POINT
' 3) INPUT SHAPE <> POINT       COMPARISON SHAPE = POINT
' 4) INPUT SHAPE = POINT        COMPARISON SHAPE = POINT

if ((theInputShape.Is(Point)) AND (CompShape.Is(Point).Not)) then

  theX1 = theCompPointOne.GetX      'POINT ONE ON THE CLOSEST LINE SEGMENT
  theY1 = theCompPointOne.GetY      'POINT ONE ON THE CLOSEST LINE SEGMENT
  theX2 = theCompPointTwo.GetX      'POINT TWO ON THE CLOSEST LINE SEGMENT
  theY2 = theCompPointTwo.GetY      'POINT TWO ON THE CLOSEST LINE SEGMENT
  theX3 = theInputShape.GetX        'INPUT POINT
  theY3 = theInputShape.GetY        'INPUT POINT

  theSlope = ((theY1 - theY2)/(theX1 - theX2))
  theInvSlope = (-1/theSlope)

  LineOneIntercept = (-1*(theSlope * theX1)) + theY1
  LineInvIntercept = (-1*(theInvSlope * theX3)) + theY3

  theIntX = (((LineOneIntercept/theInvSlope) - (LineInvIntercept/theInvSlope)) / (1 - (theSlope/theInvSlope)))
  theIntY = (theSlope*theIntX) + LineOneIntercept

  ' SPECIAL CASE:  IF SLOPE IS PERFECTLY HORIZONTAL OR PERFECTLY VERTICAL, ABOVE CALCULATIONS DON'T WORK.
```

```
  if (theSlope.IsInfinity) then                          ' SLOPE PERFECTLY VERTICAL, INVERSE SLOPE HORIZONTAL
    theIntY = theY3
    theIntPoint = Point.Make (theX1, theIntY)
  elseif (theSlope=0) then                               ' SLOPE PERFECTLY HORIZONTAL, INVERSE SLOPE VERTICAL
    theIntX = theX3
    theIntPoint = Point.Make (theIntX, theY1)
  else
    theIntPoint = Point.make (theIntX, theIntY)
  end                                                    ' END SPECIAL CASE FOR VERTICAL AND HORIZONTAL SLOPES

  if (theIntPoint.Intersects (theCompLine)) then
    theIntLine = Line.Make (theInputShape, theIntPoint)
  elseIf (theInputShape.Distance (theCompPointOne) <= theInputShape.Distance (theCompPointTwo)) then
    theIntLine = Line.Make (theInputShape, theCompPointOne)
  elseIf (theInputShape.Distance (theCompPointOne) > theInputShape.Distance (theCompPointTwo)) then
    theIntLine = Line.Make (theInputShape, theCompPointTwo)
  end

elseif ((theInputShape.Is(Point).Not) AND (CompShape.Is(Point).Not)) then

' WORKING WITH TWO LINES NOW, THE LINES FROM THE INPUT SHAPE AND THE COMPARISON SHAPE THAT WERE CLOSEST.

  theX1 = theCompPointOne.GetX      'POINT ONE ON THE CLOSEST LINE SEGMENT
  theY1 = theCompPointOne.GetY      'POINT ONE ON THE CLOSEST LINE SEGMENT
  theX2 = theCompPointTwo.GetX      'POINT TWO ON THE CLOSEST LINE SEGMENT
  theY2 = theCompPointTwo.GetY      'POINT TWO ON THE CLOSEST LINE SEGMENT
  theX3 = theInputPointOne.GetX     'POINT ONE ON THE INPUT LINE SEGMENT
  theY3 = theInputPointOne.GetY     'POINT ONE ON THE INPUT LINE SEGMENT
  theX4 = theInputPointTwo.GetX     'POINT TWO ON THE INPUT LINE SEGMENT
  theY4 = theInputPointTwo.GetY     'POINT TWO ON THE INPUT LINE SEGMENT

' CALCULATE SLOPES AND INVERSE SLOPES FOR BOTH LINES

  theCompSlope = ((theY1 - theY2)/(theX1 - theX2))
  theInvCompSlope = (-1/theCompSlope)
  theInputSlope = ((theY3 - theY4)/(theX3 - theX4))
  theInputInvSlope = (-1/theInputSlope)

' CALCULATE Y-INTERCEPTS FOR BOTH LINES, THEN RUN THE SLOPES THROUGH BOTH POINTS ON THE OPPOSING LINE AND
' CALCULATE Y-INTERCEPTS FOR BOTH OF THOSE LINES.  NOW HAVE TWO ORIGINAL LINES AND 4 POTENTIAL INTERSECTION LINES

  LineOneIntercept = (-1*(theCompSlope * theX1)) + theY1
  LineInvIntercept1 = (-1*(theInvCompSlope * theX3)) + theY3
  LineInvIntercept2 = (-1*(theInvCompSlope * theX4)) + theY4

  LineInputIntercept = (-1*(theInputSlope * theX3)) + theY3
  LineInputInvIntercept1 = (-1*(theInputInvSlope * theX1)) + theY1
  LineInputInvIntercept2 = (-1*(theInputInvSlope * theX2)) + theY2

' CALCULATE THE FOUR POINTS AT WHICH THE FOUR INTERSECTION LINES CROSS THE OPPOSITE LINE.
```

```
    ' COMPARISON LINE CROSSING THE FIRST INPUT INTERSECTION LINE
    theIntX1 = (((LineOneIntercept/theInvCompSlope) - (LineInvIntercept1/theInvCompSlope)) / (1 - (theCompSlope/theInvCompSlope)))
    theIntY1 = (theCompSlope*theIntX1) + LineOneIntercept

    ' COMPARISON LINE CROSSING THE SECOND INPUT INTERSECTION LINE
    theIntX2 = (((LineOneIntercept/theInvCompSlope) - (LineInvIntercept2/theInvCompSlope)) / (1 - (theCompSlope/theInvCompSlope)))
    theIntY2 = (theCompSlope*theIntX2) + LineOneIntercept


    ' INPUT LINE CROSSING THE FIRST COMPARISON INTERSECTION LINE
    theIntX3 = (((LineInputIntercept/theInputInvSlope) - (LineInputInvIntercept1/theInputInvSlope)) / (1 -
(theInputSlope/theInputInvSlope)))
    theIntY3 = (theInputSlope*theIntX3) + LineInputIntercept

    ' INPUT LINE CROSSING THE SECOND COMPARISON INTERSECTION LINE
    theIntX4 = (((LineInputIntercept/theInputInvSlope) - (LineInputInvIntercept2/theInputInvSlope)) / (1 -
(theInputSlope/theInputInvSlope)))
    theIntY4 = (theInputSlope*theIntX4) + LineInputIntercept

    ' SPECIAL CASE:  IF COMPARISON SLOPE IS PERFECTLY HORIZONTAL OR PERFECTLY VERTICAL, ABOVE CALCULATIONS DON'T WORK.

    if (theCompSlope.IsInfinity) then                       ' SLOPE PERFECTLY VERTICAL, INVERSE SLOPE HORIZONTAL
      theIntY1 = theY3
      theIntY2 = theY4
      theIntX1 = theX1
      theIntX2 = theX2
    elseif (theCompSlope=0) then                            ' SLOPE PERFECTLY HORIZONTAL, INVERSE SLOPE VERTICAL
      theIntX1 = theX3
      theIntX2 = theX4
      theIntY1 = theY1
      theIntY2 = theY2
    end                                                     ' END SPECIAL CASE FOR VERTICAL AND HORIZONTAL SLOPES

    ' SPECIAL CASE:  IF INPUT SLOPE IS PERFECTLY HORIZONTAL OR PERFECTLY VERTICAL, ABOVE CALCULATIONS DON'T WORK.

    if (theInputSlope.IsInfinity) then                      ' SLOPE PERFECTLY VERTICAL, INVERSE SLOPE HORIZONTAL
      theIntY3 = theY1
      theIntY4 = theY2
      theIntX3 = theX3
      theIntX4 = theX4
    elseif (theInputSlope=0) then                           ' SLOPE PERFECTLY HORIZONTAL, INVERSE SLOPE VERTICAL
      theIntX3 = theX1
      theIntX4 = theX2
      theIntY3 = theY3
      theIntY4 = theY4
    end                                                     ' END SPECIAL CASE FOR VERTICAL AND HORIZONTAL SLOPES

    theIntPoint1 = Point.make (theIntX1, theIntY1)
    theIntPoint2 = Point.make (theIntX2, theIntY2)
    theIntPoint3 = Point.make (theIntX3, theIntY3)
```

```
  theIntPoint4 = Point.make (theIntX4, theIntY4)

' NOW MAKE LINES CONNECTING THE FOUR ORIGINAL POINTS WITH THE CLOSEST POINT ON THE OPPOSITE FEATURE (THE CLOSEST
' POINT WILL EITHER BE ONE OF THE OTHER ORIGINAL POINTS OR ONE OF THE NEW INTERSECT POINTS).  EACH POINT HAS TO
' CHECK THE DISTANCE BETWEEN 3 FEATURES:  THE OPPOSING 2 POINTS AND THE INTERCEPT POINT.

  if (theIntPoint1.Intersects (theCompLine)) then
    theIntLine1 = Line.Make (theInputPointOne, theIntPoint1)
  elseIf (theInputPointOne.Distance (theCompPointOne) <= theInputPointOne.Distance (theCompPointTwo)) then
    theIntLine1 = Line.Make (theInputPointOne, theCompPointOne)
  elseIf (theInputPointOne.Distance (theCompPointOne) > theInputPointOne.Distance (theCompPointTwo)) then
    theIntLine1 = Line.Make (theInputPointOne, theCompPointTwo)
  end

  if (theIntPoint2.Intersects (theCompLine)) then
    theIntLine2 = Line.Make (theInputPointTwo, theIntPoint2)
  elseIf (theInputPointTwo.Distance (theCompPointOne) <= theInputPointTwo.Distance (theCompPointTwo)) then
    theIntLine2 = Line.Make (theInputPointTwo, theCompPointOne)
  elseIf (theInputPointTwo.Distance (theCompPointOne) > theInputPointTwo.Distance (theCompPointTwo)) then
    theIntLine2 = Line.Make (theInputPointTwo, theCompPointTwo)
  end

  if (theIntPoint3.Intersects (theInputLine)) then
    theIntLine3 = Line.Make (theIntPoint3, theCompPointOne)
  elseIf (theCompPointOne.Distance (theInputPointOne) <= theCompPointOne.Distance (theInputPointTwo)) then
    theIntLine3 = Line.Make (theInputPointOne, theCompPointOne)
  elseIf (theCompPointOne.Distance (theInputPointOne) > theCompPointOne.Distance (theInputPointTwo)) then
    theIntLine3 = Line.Make (theInputPointTwo, theCompPointOne)
  end

  if (theIntPoint4.Intersects (theInputLine)) then
    theIntLine4 = Line.Make (theIntPoint4, theCompPointTwo)
  elseIf (theCompPointTwo.Distance (theInputPointOne) <= theCompPointTwo.Distance (theInputPointTwo)) then
    theIntLine4 = Line.Make (theInputPointOne, theCompPointTwo)
  elseIf (theCompPointTwo.Distance (theInputPointOne) > theCompPointTwo.Distance (theInputPointTwo)) then
    theIntLine4 = Line.Make (theInputPointTwo, theCompPointTwo)
  end

' FIND THE SHORTEST OF THE FOUR LINES

  theIntLine = Line.Make ((theFullExtent.GetLeft)@(theFullExtent.GetBottom), (theFullExtent.GetRight)@(theFullExtent.GetTop))
  for each IntLineTest in {theIntLine1, theIntLine2, theIntLine3, theIntLine4}
    if (IntLineTest.ReturnLength < theIntLine.ReturnLength) then
      theIntLine = IntLineTest
    end
  end

elseif ((theInputShape.Is(Point).Not) AND (CompShape.Is(Point))) then

  theX1 = CompShape.GetX         'COMPARISON POINT
  theY1 = CompShape.GetY         'COMPARISON POINT
```

```
    theX2 = theInputPointOne.GetX  'POINT ONE ON THE INPUT LINE SEGMENT
    theY2 = theInputPointOne.GetY  'POINT ONE ON THE INPUT LINE SEGMENT
    theX3 = theInputPointTwo.GetX  'POINT TWO ON THE INPUT LINE SEGMENT
    theY3 = theInputPointTwo.GetY  'POINT TWO ON THE INPUT LINE SEGMENT

    theSlope = ((theY2 - theY3)/(theX2 - theX3))
    theInvSlope = (-1/theSlope)

    LineOneIntercept = (-1*(theSlope * theX2)) + theY2
    LineInvIntercept = (-1*(theInvSlope * theX1)) + theY1

    theIntX = (((LineOneIntercept/theInvSlope) - (LineInvIntercept/theInvSlope)) / (1 - (theSlope/theInvSlope)))
    theIntY = (theSlope*theIntX) + LineOneIntercept

    ' SPECIAL CASE:  IF SLOPE IS PERFECTLY HORIZONTAL OR PERFECTLY VERTICAL, ABOVE CALCULATIONS DON'T WORK.

    if (theSlope.IsInfinity) then                          ' SLOPE PERFECTLY VERTICAL, INVERSE SLOPE HORIZONTAL
      theIntY = theY1
      theIntPoint = Point.Make (theX2, theIntY)
    elseif (theSlope=0) then                               ' SLOPE PERFECTLY HORIZONTAL, INVERSE SLOPE VERTICAL
      theIntX = theX1
      theIntPoint = Point.Make (theIntX, theY2)
    else
      theIntPoint = Point.make (theIntX, theIntY)
    end                                                    ' END SPECIAL CASE FOR VERTICAL AND HORIZONTAL SLOPES

    if (theIntPoint.Intersects (theInputLine)) then
      theIntLine = Line.Make (theIntPoint, CompShape)
    elseIf (CompShape.Distance (theInputPointOne) <= CompShape.Distance (theInputPointTwo)) then
      theIntLine = Line.Make (theInputPointOne, CompShape)
    elseIf (CompShape.Distance (theInputPointOne) > CompShape.Distance (theInputPointTwo)) then
      theIntLine = Line.Make (theInputPointTwo, CompShape)
    end

  elseif ((theInputShape.Is(Point)) AND (CompShape.Is(Point))) then

    TheIntLine = Line.Make(theInputShape, CompShape)

  end ' CALCULATING CLOSEST POINTS AND CONNECTING LINES BASED ON FOUR SITUATIONS
end   ' END CHECKING TO SEE IF TWO SHAPES INTERSECT

return theIntLine
```

"Make Point and Line" script:

```
' Sample.CalcPointLine
' Jenness Enterprises <www.jennessent.com>
' Given an origin point, distance and bearing, this script will return a new point at that distance and bearing, and a line
' connecting that new point to the origin point
```

```
theOrigin = self.Get(0)
theLength = self.Get(1)
theAzimuth = self.Get(2)

' MAKE SURE AZIMUTH IS BETWEEN 0 AND 360
while (theAzimuth < 0)
  theAzimuth = theAzimuth+360
end
theAzimuth = theAzimuth.Mod(360)

' NEW SEGMENT AND POINT DISTANCE NORTH/SOUTH AND EAST/WEST BASED ON DISTANCE AND BEARING FROM ORIGIN.
' THERE ARE EIGHT DIFFERENT POSSIBILITIES:  THE BEARING COULD BE ONE OF THE FOUR CARDINAL DIRECTIONS OR IT
' COULD BE IN ONE OF THE FOUR QUADRANTS.  THE BEARING IS TREATED DIFFERENTLY IN EACH OF THESE CIRCUMSTANCES.
' THE CALCULATION TO DETERMINE THE NEW POINT LOCATION IS ESSENTIALLY A MATTER OF TAKING THE SINE OR THE
' COSINE OF THE ANGLE (AFTER CONVERTING IT TO RADIANS), AND MULTIPLYING THAT SINE OR COSINE BY THE MEASURED
' DISTANCE.  PLEASE CONTACT THE AUTHOR IF THIS DOESN'T MAKE SENSE, OR IF YOU WOULD LIKE FURTHER EXPLANATION.

if ((theAzimuth = 0) or (theAzimuth = 360)) then
  NorthSouthDistance = theLength
  NorthSouth = 1
  EastWestDistance = 0
  EastWest = 1
elseif (theAzimuth = 180) then
  NorthSouthDistance = theLength
  NorthSouth = -1
  EastWestDistance = 0
  EastWest = 1
elseif (theAzimuth = 90) then
  NorthSouthDistance = 0
  NorthSouth = 1
  EastWestDistance = theLength
  EastWest = 1
elseif (theAzimuth = 270) then
  NorthSouthDistance = 0
  NorthSouth = 1
  EastWestDistance = theLength
  EastWest = -1
elseif ((theAzimuth > 0) and (theAzimuth < 90)) then
  NorthSouthDistance = ((theAzimuth.AsRadians.Cos)*theLength)
  NorthSouth = 1
  EastWestDistance = ((theAzimuth.AsRadians.Sin)*theLength)
  EastWest = 1
elseif ((theAzimuth > 90) and (theAzimuth < 180)) then
  NorthSouthDistance = (((theAzimuth - 90).AsRadians.Sin)*theLength)
  NorthSouth = -1
  EastWestDistance = (((theAzimuth - 90).AsRadians.Cos)*theLength)
  EastWest = 1
elseif ((theAzimuth > 180) and (theAzimuth < 270)) then
  NorthSouthDistance = (((theAzimuth - 180).AsRadians.Cos)*theLength)
  NorthSouth = -1
  EastWestDistance = (((theAzimuth - 180).AsRadians.Sin)*theLength)
```

```
  EastWest = -1
elseif ((theAzimuth > 270) and (theAzimuth < 360)) then
  NorthSouthDistance = (((theAzimuth - 270).AsRadians.Sin)*theLength)
  NorthSouth = 1
  EastWestDistance = (((theAzimuth - 270).AsRadians.Cos)*theLength)
  EastWest = -1
else
  msgBox.Info ("Problem:  "+theAzimuth.AsString+" doesn't lie within 0-360 degrees!", "Problem:")
end

theMovementNorth = NorthSouthDistance*NorthSouth
theMovementWest = EastWestDistance*EastWest

theEndPoint = Point.Make (theOrigin.GetX + theMovementWest, theOrigin.GetY + theMovementNorth)
thePolyline = PolyLine.Make({{theOrigin, theEndPoint}})

return {theEndPoint, thePolyline}
```

"Triangle Area from Points" script:

```
' Sample.CalcTrianglePoints
' Jenness Enterprises <www.jennessent.com)>
' Given 3 points, this scripts calculates the area of the triangle defined
' by those points.

thePX = self.Get(0).GetX
thePY = self.Get(0).GetY
theQX = self.Get(1).GetX
theQY = self.Get(1).GetY
theRX = self.Get(2).GetX
theRY = self.Get(2).GetY

theArea = ((((theQX - thePX)*(theRY - thePY)) - ((theRX - thePX)*(theQY - thePY)))/2).abs

return theArea
```

"Triangle Area from 3D Points" script:

```
' Sample.CalcTriangle3DPoints
' Jenness Enterprises <www.jennessent.com)>
' Given 3 three-dimensional points, this scripts calculates the area of the triangle defined
' by those points.

thePX = self.Get(0).GetX
```

```
thePY = self.Get(0).GetY
thePZ = self.Get(0).GetZ
theQX = self.Get(1).GetX
theQY = self.Get(1).GetY
theQZ = self.Get(1).GetZ
theRX = self.Get(2).GetX
theRY = self.Get(2).GetY
theRZ = self.Get(2).GetZ

thePQVec = {theQX - thePX, theQY - thePY, theQZ - thePZ}
thePRVec = {theRX - thePX, theRY - thePY, theRZ - thePZ}

theI = (((theQY - thePY)*(theRZ - thePZ)) - ((theRY - thePY)*(theQZ - thePZ)))^2
theJ = (((theQX - thePX)*(theRZ - thePZ)) - ((theRX - thePX)*(theQZ - thePZ)))^2
theK = (((theQX - thePX)*(theRY - thePY)) - ((theRX - thePX)*(theQY - thePY)))^2

theArea = ((theI + theJ + theK).Sqrt)/2

return theArea
```

"Triangle Area from Sides" script:

```
' Sample.CalcTriangleSides
' Jenness Enterprises <www.jennessent.com)>
' Given 3 triangle side lengths, this scripts calculates the area of the triangle defined
' by those sides.  Returns "Number Null" if not a true triangle

sideA = self.Get(0)
sideB = self.Get(1)
sideC = self.Get(2)

theS = (sideA + sideB + sideC)/2

theArea = (theS * (theS - sideA) * (theS - sideB) * (theS - sideC)).sqrt

return theArea
```

"Generate Center of Mass and Area" script:

```
' Sample.CalcCenterOfMass
' Jenness Enterprises <www.jennessent.com)>

' LOOSELY ADAPTED FROM ALGORITHMS IN JOSEPH O'ROURKE (1998):  COMPUTATIONAL GEOMETRY IN C, 2ND EDITION,
'   CAMBRIDGE UNIVERSITY PRESS.  P. 21
```

```
if (self.IsNull) then
  return {Point.MakeNull, Number.MakeNull}
else

  theESRICentroid = self.ReturnCenter
  thePolygonList = self.Explode

  theAreas = {}
  theCenters = {}
  for each aPoly in thePolygonList
    theVertices = aPoly.AsMultipoint.AsList
    for each anIndex in 0..(theVertices.Count-2)
      theP = theVertices.Get(anIndex)
      theQ = theVertices.Get(anIndex+1)
      theCentX = theESRICentroid.GetX
      theCentY = theESRICentroid.GetY

      theArea = -((((theQ.GetX - theP.GetX)*(theCentY - theP.GetY)) -
                 ((theCentX - theP.GetX)*(theQ.GetY - theP.GetY)))/2)
      theCenter = (theP + theQ + theESRICentroid)/(3@3)

      theAreas.Add(theArea)
      theCenters.Add(theCenter)
    end
  end

  theCount = theAreas.Count

  theArea = 0
  theCentroid = 0@0

  for each anIndex in 0..(theCount-1)
    theSubArea = theAreas.Get(anIndex)
    theArea = theArea+theSubArea
    theCentroid = theCentroid+(theCenters.Get(anIndex) * (theSubArea@theSubArea))
  end

  theCentroid = theCentroid / (theArea@theArea)

  return {theCentroid, theArea}
end
```

“Calculate Internal Angle” script:

```
' Sample.CalcInternalAngle
' Jenness Enterprises <www.jennessent.com>
' Given 3 consecutive points, this scripts calculates how many degrees the bearing of the second segment
' deviates from the bearing of the first segment.
```

```
PointA = self.Get(0)
PointB = self.Get(1)
PointC = self.Get(2)

' FOR DEBUGGING
'PointA = Point.Make(4.9, 5)
'PointB = Point.Make(10, 10)
'PointC = Point.Make(5, 5)

'PointA = Point.Make(449400, 3704000)
'PointB = Point.Make(449550, 3704150)
'PointC = Point.Make(450000, 3704600)
'PointC = Point.Make(449400, 3704000)

' INTERNAL ANGLE WITH LAW OF COSINES;
'       c^2 = a^2 + b^2 - (2ab * Cos C), OR
'       Cos C = (a^2 +b^2 - c^2)/(2ab)

lenA = Line.Make(PointA, PointB).returnLength
lenB = Line.Make(PointC, PointB).returnLength
lenC = Line.Make(PointC, PointA).returnLength

InternalAngle = (((lenA^2) + (lenB^2) - (lenC^2))/(2*lenA*lenB)).ACos.AsDegrees
theAngleDeviation = 180-InternalAngle

' IF EITHER IS NULL, CHECK ANGLES INDIVIDUALLY
if ((InternalAngle.IsNull) or (theAngleDeviation.IsNull)) then
  GetBearing = av.FindScript("Sample.CalcBearing")
  theFirstAngle = GetBearing.DoIt({PointA, PointB})
  theSecondAngle = GetBearing.DoIt({PointB, PointC})
  if (theFirstAngle = theSecondAngle) then
    InternalAngle = 180
    theAngleDeviation = 0
  else
    InternalAngle = 0
    theAngleDeviation = 180
  end
end

'msgBox.Info(InternalAngle.AsString +" Internal"+NL+theAngleDeviation.AsString+" Deviation", "")

return {InternalAngle, theAngleDeviation}
```

“Generate Convex Hull” script:

```
' Sample.CalcConvexHull

theList = Self
```

```
CheckClockwise = av.FindScript("CalcCheckClockwise")

av.ShowMsg("Sorting points...")

theList.RemoveDuplicates

theXList = {}
for each aPoint in theList
  theXList.Add(aPoint.GetX)
end

theXList.RemoveDuplicates
theXList.Sort(True)

' ADD X-VALUES TO DICTIONARY AS KEYS
theCounter = 0
theCount = theList.Count

theDictionary = Dictionary.Make(theXList.Count)
for each anX in theXList
  theCounter = theCounter+1
  av.SetStatus((theCounter/theCount)*100)
  theDictionary.Set(anX, {})
end

' ADD POINTS TO DICTIONARY AS ELEMENTS
theCounter = 0
theCount = theList.Count

for each aPoint in theList
  theCounter = theCounter+1
  av.SetStatus((theCounter/theCount)*100)
  theShortList = theDictionary.Get(aPoint.GetX)
  theShortList.Add(aPoint)

  ' SHUFFLE POINT TO APPROPRIATE PLACE IN LIST (SORTED LOW TO HIGH)
  theIndex = theShortList.Count-1
  while ((theIndex > 0) and (theShortList.Get(theIndex).GetY < theShortList.Get(theIndex-1).GetY))
    theShortList.Shuffle(aPoint, theIndex-1)
    theIndex = theIndex -1
  end
end

' CHECK FOR SUFFICIENT NUMBER OF POINTS
theCount = 0
CountOK = False
theShowList = {}

for each aPoint in theXList
  theCheckList = {}
  theTestList = theDictionary.Get(aPoint).DeepClone
```

```
  for each aPoint in theTestList
    theCheckList.Add(aPoint.GetY)
  end
  theCheckList.RemoveDuplicates
  theCount = theCount+theCheckList.Count
  if (theCount >= 3) then
    CountOK = True
    break
  end
end

if ((CountOK) AND (theXList.Count = 1)) then
  return "All points have same X-coordinate"
end

if (CountOK) then

  ' UPPER HULL GOES FROM LOWER LEFT TO UPPER RIGHT
  theUpperHullPoints = theDictionary.Get(theXList.Get(0))            ' START WITH LEFT-MOST POINTS
  if (theUpperHullPoints.Count > 2) then
    theUpperHullPoints = {theUpperHullPoints.Get(0), theUpperHullPoints.Get(theUpperHullPoints.Count-1)}
  end

  ' HAVE TO CHECK FOR RIGHT-HAND TURN (CLOCKWISE).  THIS CAN BE COMPUTED BY CHECKING THE SIGN OF THE DETERMINANT OF
  '
  '           |                 |
  '           | 1   p(x)   p(y) |        WHERE THE THREE POINTS ARE ORDERED (p,q,r).
  '           |                 |
  '       D = | 1   q(x)   q(y) |                        |ABC|
  '           |                 |        DETERMINANT OF  |DEF| = (AEI)-(AFH)-(BDI)+(BFG)+(CDH)-(CEG)
  '           | 1   r(x)   r(y) |                        |GHI|
  '           |                 |
  '                                      BECAUSE OF OUR COLUMN OF 1 VALUES, THIS CAN BE REDUCED T0
  '                                                    (EI)-(FH)-(BI)+(BF)+(CH)-(CE)
  '
  '                          OR    (q(x)r(y))-(q(y)r(x))-(p(x)r(y))+(p(x)q(y))+(p(y)r(x))-(p(y)q(x))
  '
  ' SOURCE:  M. de Berg, M. van Dreveld, M. Overmars and O. Schwarzkopf.  1998.  Computational Geometry,
  '          Algorithms and Applications (2nd. Edition)  Springer, p. 16

  ' GET UPPER HULL

  av.ShowMsg("Generating Upper Convex Hull...")

  theCounter = theUpperHullPoints.Count + 1
  theCount = theXList.Count - 1

  for each anIndex in 1..(theXList.Count-1)
    theCounter = theCounter+1
    av.SetStatus((theCounter/theCount)*100)
```

```
    theShortList = theDictionary.Get(theXList.Get(anIndex))
    theRPoint = theShortList.Get(theShortList.Count-1)
    theUpperHullPoints.Add(theRPoint)

    while ((theUpperHullPoints.Count > 2) and
          (CheckClockwise.DoIt({theUpperHullPoints.Get(theUpperHullPoints.Count-3),
                  theUpperHullPoints.Get(theUpperHullPoints.Count-2), theRPoint}).Not))
      theUpperHullPoints.Remove(theUpperHullPoints.Count-2)
    end
  end

  ' GET LOWER HULL

  av.ShowMsg("Generating Lower Convex Hull...")

  theLowerHullPoints = {}
  theLowerHullStartList = theDictionary.Get(theXList.Get(theXList.Count-1))
  if (theLowerHullStartList.Count > 1) then
    theLowerHullPoints = {theLowerHullStartList.Get(theLowerHullStartList.Count-1), theLowerHullStartList.Get(0)}
  else
    theLowerHullPoints = theLowerHullStartList
  end

  theCounter = 2
  theCount = theXList.Count - 1

  for each anIndex in (theXList.Count-2)..0 by -1

    theCounter = theCounter+1
    av.SetStatus((theCounter/theCount)*100)

    theRPoint = theDictionary.Get(theXList.Get(anIndex)).Get(0)
    theLowerHullPoints.Add(theRPoint)

    while ((theLowerHullPoints.Count > 2) and
          (CheckClockwise.DoIt({theLowerHullPoints.Get(theLowerHullPoints.Count-3),
                  theLowerHullPoints.Get(theLowerHullPoints.Count-2), theRPoint}).Not))
      theLowerHullPoints.Remove(theLowerHullPoints.Count-2)
    end
  end

  theLowerHullPoints.Remove(0)
  theLowerHullPoints.Remove(theLowerHullPoints.Count-1)
  AllPoints = theUpperHullPoints+theLowerHullPoints
  thePoly = Polygon.Make({AllPoints})

else
  thePoly = "Insufficient Unique points available..."
end

return thePoly
```

"Check if line segments cross or touch" script:

```
' Jennessent.CalcLineIntersect
' Jenness Enterprises <www.jennessent.com>
'
' ADAPTED FROM ALGORITHMS IN Cormen, Thomas H.; Leiserson, Charles E.; Rivest,
' Ronald L.; and Stein, Clifford.  2001.  Introduction to Algorithms, 2nd. Ed.
' Massachusetts Institute of Technology Press.
'
' GIVEN 4 POINTS, REPRESENTING START AND END OF LINE 1, AND START AND END OF LINE 2
' RETURNS 0 IF LINES INTERSECT
'         1 IF POINT FROM ONE LINE SITS ON THE OTHER LINE
'         2 IF LINES DO NOT INTERSECT

Line1Point1 = self.Get(0)     ' P1
Line1Point2 = self.Get(1)     ' P2
Line2Point1 = self.Get(2)     ' P3
Line2Point2 = self.Get(3)     ' P4

the11X = Line1Point1.GetX
the11Y = Line1Point1.GetY
the12X = Line1Point2.GetX
the12Y = Line1Point2.GetY
the21X = Line2Point1.GetX
the21Y = Line2Point1.GetY
the22X = Line2Point2.GetX
the22Y = Line2Point2.GetY

theDir1 = (the22X * (the11Y - the21Y)) + (the22Y * (the21X - the11X)) - ((the21X)*(the11Y)) + ((the21Y)*(the11X))
theDir2 = (the22X * (the12Y - the21Y)) + (the22Y * (the21X - the12X)) - ((the21X)*(the12Y)) + ((the21Y)*(the12X))
theDir3 = (the12X * (the21Y - the11Y)) + (the12Y * (the11X - the21X)) - ((the11X)*(the21Y)) + ((the11Y)*(the21X))
theDir4 = (the12X * (the22Y - the11Y)) + (the12Y * (the11X - the22X)) - ((the11X)*(the22Y)) + ((the11Y)*(the22X))

if ((((theDir1 > 0) and (theDir2 < 0)) or ((theDir1 < 0) and (theDir2 > 0))) and
    (((theDir3 > 0) and (theDir4 < 0)) or ((theDir3 < 0) and (theDir4 > 0)))) then
  Return 0
elseif ((theDir1 = 0) and (the21X.Min(the22X) <= the11X) and (the11X <= the21X.Max(the22X)) and
                          (the21Y.Min(the22Y) <= the11Y) and (the11Y <= the21Y.Max(the22Y))) then
  Return 1
elseif ((theDir2 = 0) and (the21X.Min(the22X) <= the12X) and (the12X <= the21X.Max(the22X)) and
                          (the21Y.Min(the22Y) <= the12Y) and (the12Y <= the21Y.Max(the22Y))) then
  Return 1
elseif ((theDir3 = 0) and (the11X.Min(the12X) <= the21X) and (the21X <= the11X.Max(the12X)) and
                          (the11Y.Min(the12Y) <= the21Y) and (the21Y <= the11Y.Max(the12Y))) then
  Return 1
elseif ((theDir4 = 0) and (the11X.Min(the12X) <= the22X) and (the22X <= the11X.Max(the12X)) and
                          (the11Y.Min(the22Y) <= the22Y) and (the22Y <= the11Y.Max(the12Y))) then
  Return 1
```

```
else
  Return 2
end
```

VB Code Generated:

Sample Resize Anchor Code, to insert into Load subroutine:

```
' PUT IN GENERAL DECLARATIONS SECTION
Private Anchors As AnchorObjectList  ' Main anchor control object


   Set Anchors = New AnchorObjectList  ' Create new instance
   With Anchors
      With .Item(cmdCancel)
         .SetAnchors enumSizeEnd, enumStartSize
      End With
      With .Item(cmdOK)
         .SetAnchors enumSizeEnd, enumStartSize
      End With
      With .Item(cpIncludeInTable)
         .SetAnchors enumStartEnd, enumSizeEnd
      End With
      With .Item(lbxSumOptions)
         .SetAnchors enumStartEnd, enumSizeEnd
      End With
      With .Item(cmdAdd)
         .SetAnchors enumStartSize, enumSizeEnd
      End With
      With .Item(cmdDelete)
         .SetAnchors enumStartSize, enumSizeEnd
      End With
      With .Item(cbxFieldInclude)
         .SetAnchors enumStartSize, enumSizeEnd
      End With
      With .Item(lblField)
         .SetAnchors enumStartSize, enumSizeEnd
      End With
      With .Item(cbxFieldSummarizeBy)
         .SetAnchors enumStartSize, enumStartSize
      End With
      With .Item(lblSummarizeBy)
         .SetAnchors enumStartSize, enumSizeEnd
      End With
      With .Item(lblTheme)
         .SetAnchors enumNone, enumStartSize
      End With
      With .Item(lblSummaryField)
         .SetAnchors enumNone, enumStartSize
```

```
      End With
      With .Item(lbxThemes)
         .SetAnchors enumNone, enumStartEnd
      End With
      With .Item(lbxSummaryFields)
         .SetAnchors enumNone, enumStartEnd
      End With
      With .Item(optAllValues)
         .SetAnchors enumSizeEnd, enumStartSize
      End With
      With .Item(optSelValues)
         .SetAnchors enumSizeEnd, enumStartSize
      End With
      .Form = Me  ' Set form reference (suggested to be last step)
   End With
```

Anchor Class Module: Anchor.cls

```
' Anchors
' Class: Anchor
' By neophile (n_e_o_p_h_i_l_e@yahoo.com)
' 5/28/2002
' ------------------------------------------------
' MODIFIED JANUARY 2006 TO ALLOW FOR DIFFERENT ANCHOR TYPES
' JEFF JENNESS (jeffj@jennessent.com)
'

Option Explicit

Public Enum AnchorTypes
   enumNone        ' Avenue Fastener ( - , - , - )
   enumStart       ' Avenue Fastener (Left/Top, - , - )
   enumStartSize   ' Avenue Fastener (Left/Top, Width/Height, - )
   enumStartEnd    ' Avenue Fastener (Left/Top, - , Right/Bottom )
   enumSize        ' Avenue Fastener ( - , Width/Height, - )
   enumSizeEnd     ' Avenue Fastener ( - , Width/Height, Right/Bottom)
   enumEnd         ' Avenue Fastener ( - , - , Right/Bottom)

   'atPosition  ' Anchor position (Left/Top)
   'atSize      ' Anchor size (Width/Height)
End Enum


Public AnchorType As AnchorTypes  ' Anchor type

Public MinValue As Long  ' Minimum value

Public MaxValue As Long  ' Maximum value

Public Value As Single  ' Relative distance
```

```
Private Sub Class_Initialize()
   MinValue = -&H7FFFFFFF  ' Set to max lower limit
   MaxValue = &H7FFFFFFF  ' Set to max upper limit
End Sub
```

AnchorObject Class Module: AnchorObject.cls

```
' Anchors
' Class: AnchorObject
' By neophile (n_e_o_p_h_i_l_e@yahoo.com)
' 5/28/2002
' ------------------------------------------------
' MODIFIED JANUARY 2006 TO ALLOW FOR DIFFERENT ANCHOR TYPES
' JEFF JENNESS (jeffj@jennessent.com)
'

Option Explicit

Private mCtl As Control    ' Control reference
Private mX As Anchor       ' X anchor
Private mY As Anchor       ' Y anchor
Private mX2 As Anchor      ' X2 anchor
Private mY2 As Anchor      ' Y2 anchor
Private mWidth As Anchor   ' Width anchor
Private mHeight As Anchor  ' Height anchor

Public Property Let Control(vData As Control)
   Set mCtl = vData  ' Set reference
End Property
Public Property Get Control() As Control
   Set Control = mCtl  ' Return reference
End Property

Public Property Get X() As Anchor
   Set X = mX  ' Return X anchor
End Property

Public Property Get Y() As Anchor
   Set Y = mY  ' Return Y anchor
End Property

Public Property Get X2() As Anchor
   Set X2 = mX2  ' Return X anchor
End Property

Public Property Get Y2() As Anchor
   Set Y2 = mY2  ' Return Y anchor
End Property

Public Sub SetAnchors(Optional ByVal XType As AnchorTypes, Optional ByVal YType As AnchorTypes)
   X.AnchorType = XType  ' Set X anchor type
```

```
Select Case XType  ' X anchor
   Case enumNone         ' Avenue Fastener ( - , - , - )
     mX.Value = mCtl.Left / mCtl.Container.Width
     mWidth.Value = mCtl.Width / mCtl.Container.Width
   Case enumStart        ' Avenue Fastener (Left, - , - )   DON'T ADJUST X AT ALL!
     mX.Value = mCtl.Left
     mWidth.Value = (mCtl.Width / mCtl.Container.Width)
     Debug.Print mCtl.Width
     Debug.Print mCtl.Container.Width
     Debug.Print (mCtl.Width / mCtl.Container.Width)
     Debug.Print mWidth.Value
   Case enumStartSize    ' Avenue Fastener (Left, Width/, - )
     mX.Value = mCtl.Left
     mWidth.Value = mCtl.Width
   Case enumStartEnd     ' Avenue Fastener (Left, - , Right )
     mX.Value = mCtl.Left
     mWidth.Value = mCtl.Container.Width - mCtl.Width
   Case enumSize         ' Avenue Fastener ( - , Width, - )
     mX.Value = ((mCtl.Left + (mCtl.Width / 2)) / mCtl.Container.Width)
     mX2.Value = mCtl.Width / 2
     mWidth.Value = mCtl.Width
   Case enumSizeEnd      ' Avenue Fastener ( - , Width, Right)
     mX.Value = mCtl.Container.Width - mCtl.Left
     mWidth.Value = mCtl.Width
   Case enumEnd          ' Avenue Fastener ( - , - , Right)
     mX.Value = mCtl.Width / mCtl.Container.Width
     mX2.Value = mCtl.Container.Width - mCtl.Width - mCtl.Left
     mWidth.Value = mCtl.Width / mCtl.Container.Width
End Select
Y.AnchorType = YType  ' Set Y anchor type
Select Case YType  ' Y anchor
   Case enumNone         ' Avenue Fastener ( - , - , - )
     mY.Value = mCtl.Top / mCtl.Container.Height
     mHeight.Value = mCtl.Height / mCtl.Container.Height
   Case enumStart        ' Avenue Fastener (Top, - , - )   DON'T ADJUST Y AT ALL!
     mY.Value = mCtl.Top
     mHeight.Value = mCtl.Height / mCtl.Container.Height
   Case enumStartSize    ' Avenue Fastener (Top, Height, - )
     mY.Value = mCtl.Top
   Case enumStartEnd     ' Avenue Fastener (Top, - , Bottom )
     mY.Value = mCtl.Top
     mHeight.Value = mCtl.Container.Height - mCtl.Height
   Case enumSize         ' Avenue Fastener ( - , Height, - )
     mY.Value = ((mCtl.Top + (mCtl.Height / 2)) / mCtl.Container.Height)
     mY2.Value = mCtl.Height / 2
   Case enumSizeEnd      ' Avenue Fastener ( - , Height, Bottom)
     mY.Value = mCtl.Container.Height - mCtl.Top
   Case enumEnd          ' Avenue Fastener ( - , - , Bottom)
     mY.Value = mCtl.Height / mCtl.Container.Height
     mY2.Value = mCtl.Container.Height - mCtl.Height - mCtl.Top
     mHeight.Value = mCtl.Height / mCtl.Container.Height
```

```
   End Select
'   Select Case YType  ' Y anchor
'      Case atPosition  ' Get position
'         mY.Value = mCtl.Container.Height - mCtl.Top  ' Control's top relative to form's bottom
'      Case atSize  ' Get size
'         mY.Value = mCtl.Container.Height - mCtl.Height  ' Control's bottom relative to form's bottom
'   End Select
End Sub

Public Sub DoAnchors()
   On Error Resume Next  ' Ignore errors
   Select Case mX.AnchorType  ' X anchor
      Case enumNone        ' Avenue Fastener ( - , - , - )
        mCtl.Left = mCtl.Container.Width * mX.Value
        mCtl.Width = mCtl.Container.Width * mWidth.Value
'         If mCtl.Left < mX.MinValue Then mCtl.Left = mX.MinValue  ' Lower limit
'         If mCtl.Left > mX.MaxValue Then mCtl.Left = mX.MaxValue  ' Upper limit
      Case enumStart       ' Avenue Fastener (Left, - , - )   DON'T ADJUST Y AT ALL!
'        mCtl.Left = mX.Value
        mCtl.Width = mCtl.Container.Width * mWidth.Value
'         If mCtl.Left < mX.MinValue Then mCtl.Left = mX.MinValue  ' Lower limit
'         If mCtl.Left > mX.MaxValue Then mCtl.Left = mX.MaxValue  ' Upper limit
      Case enumStartSize   ' Avenue Fastener (Left, Width, - )
'        mCtl.Left = mX.Value
'         If mCtl.Left < mX.MinValue Then mCtl.Left = mX.MinValue  ' Lower limit
'         If mCtl.Left > mX.MaxValue Then mCtl.Left = mX.MaxValue  ' Upper limit
      Case enumStartEnd    ' Avenue Fastener (Left, - , Right )
'        mCtl.Left = mX.Value
        mCtl.Width = mCtl.Container.Width - mWidth.Value
'         If mCtl.Left < mX.MinValue Then mCtl.Left = mX.MinValue  ' Lower limit
'         If mCtl.Left > mX.MaxValue Then mCtl.Left = mX.MaxValue  ' Upper limit
      Case enumSize        ' Avenue Fastener ( - , Width, - )
        mCtl.Left = (mCtl.Container.Width * mX.Value) - mX2.Value
'         If mCtl.Left < mX.MinValue Then mCtl.Left = mX.MinValue  ' Lower limit
'         If mCtl.Left > mX.MaxValue Then mCtl.Left = mX.MaxValue  ' Upper limit
      Case enumSizeEnd     ' Avenue Fastener ( - , Width, Right)
        mCtl.Left = mCtl.Container.Width - mX.Value
'         If mCtl.Left < mX.MinValue Then mCtl.Left = mX.MinValue  ' Lower limit
'         If mCtl.Left > mX.MaxValue Then mCtl.Left = mX.MaxValue  ' Upper limit
      Case enumEnd         ' Avenue Fastener ( - , - , Right)
        mCtl.Left = mCtl.Container.Width - (mCtl.Container.Width * mX.Value) - mX2.Value
        mCtl.Width = mCtl.Container.Width * mWidth.Value
'         If mCtl.Left < mX.MinValue Then mCtl.Left = mX.MinValue  ' Lower limit
'         If mCtl.Left > mX.MaxValue Then mCtl.Left = mX.MaxValue  ' Upper limit
   End Select
   Select Case mY.AnchorType  ' Y anchor
      Case enumNone        ' Avenue Fastener ( - , - , - )
        mCtl.Top = mCtl.Container.Height * mY.Value
        mCtl.Height = mCtl.Container.Height * mHeight.Value
'         If mCtl.Top < mY.MinValue Then mCtl.Top = mY.MinValue  ' Lower limit
'         If mCtl.Top > mY.MaxValue Then mCtl.Top = mY.MaxValue  ' Upper limit
```

```
      Case enumStart       ' Avenue Fastener (Top, - , - )   DON'T ADJUST Y AT ALL!
'        mCtl.Top = mY.Value
        mCtl.Height = mCtl.Container.Height * mHeight.Value
'         If mCtl.Top < mY.MinValue Then mCtl.Top = mY.MinValue  ' Lower limit
'         If mCtl.Top > mY.MaxValue Then mCtl.Top = mY.MaxValue  ' Upper limit
      Case enumStartSize   ' Avenue Fastener (Top, Height, - )
'        mCtl.Top = mY.Value
'         If mCtl.Top < mY.MinValue Then mCtl.Top = mY.MinValue  ' Lower limit
'         If mCtl.Top > mY.MaxValue Then mCtl.Top = mY.MaxValue  ' Upper limit
      Case enumStartEnd    ' Avenue Fastener (Top, - , Bottom )
'        mCtl.Top = mY.Value
        mCtl.Height = mCtl.Container.Height - mHeight.Value
'         If mCtl.Top < mY.MinValue Then mCtl.Top = mY.MinValue  ' Lower limit
'         If mCtl.Top > mY.MaxValue Then mCtl.Top = mY.MaxValue  ' Upper limit
      Case enumSize        ' Avenue Fastener ( - , Height, - )
        mCtl.Top = (mCtl.Container.Height * mY.Value) - mY2.Value
'         If mCtl.Top < mY.MinValue Then mCtl.Top = mY.MinValue  ' Lower limit
'         If mCtl.Top > mY.MaxValue Then mCtl.Top = mY.MaxValue  ' Upper limit
      Case enumSizeEnd     ' Avenue Fastener ( - , Height, Bottom)
        mCtl.Top = mCtl.Container.Height - mY.Value
'         If mCtl.Top < mY.MinValue Then mCtl.Top = mY.MinValue  ' Lower limit
'         If mCtl.Top > mY.MaxValue Then mCtl.Top = mY.MaxValue  ' Upper limit
      Case enumEnd         ' Avenue Fastener ( - , - , Bottom)
        mCtl.Top = mCtl.Container.Height - (mCtl.Container.Height * mY.Value) - mY2.Value
        mCtl.Height = mCtl.Container.Height * mHeight.Value
'         If mCtl.Top < mY.MinValue Then mCtl.Top = mY.MinValue  ' Lower limit
'         If mCtl.Top > mY.MaxValue Then mCtl.Top = mY.MaxValue  ' Upper limit
   End Select
   On Error GoTo 0  ' Stop ignoring errors
End Sub


Private Sub Class_Initialize()
   Set mX = New Anchor       ' Create new anchor instance
   Set mY = New Anchor       ' Create new anchor instance
   Set mX2 = New Anchor      ' Create new anchor instance
   Set mY2 = New Anchor      ' Create new anchor instance
   Set mWidth = New Anchor   ' Create new anchor instance
   Set mHeight = New Anchor  ' Create new anchor instance
End Sub

Private Sub Class_Terminate()
   Set mX = Nothing       ' Discard anchor instance
   Set mY = Nothing       ' Discard anchor instance
   Set mX2 = Nothing      ' Discard anchor instance
   Set mY2 = Nothing      ' Discard anchor instance
   Set mWidth = Nothing   ' Discard anchor instance
   Set mHeight = Nothing  ' Discard anchor instance
End Sub
```

AnchorObjectList Class Module: AnchorObjectList.cls

```
' Anchors
' Class: AnchorObjectList
' By neophile (n_e_o_p_h_i_l_e@yahoo.com)
' 5/28/2002
' ------------------------------------------------
' MODIFIED JANUARY 2006 TO ALLOW FOR DIFFERENT ANCHOR TYPES
' JEFF JENNESS (jeffj@jennessent.com)
'

Option Explicit

Private Declare Function LockWindowUpdate Lib "user32" (ByVal hwndLock As Long) As Long

Private WithEvents mForm As Form  ' Form reference
Private mCol As Collection  ' Item collection


Public Property Let Form(vData As Form)
   Set mForm = vData  ' Set reference
End Property
Public Property Get Form() As Form
   Set Form = mForm   ' Return reference
End Property

Public Function Count() As Long
   Count = mCol.Count  ' Return anchor collection count
End Function

Public Function Item(Control As Control) As AnchorObject
   Dim lIdx As Long
   lIdx = IndexOf(Control)  ' Get position in item collection
   If lIdx = 0 Then  ' If no item was found...
      Set Item = New AnchorObject  ' ...create a new item
      Item.Control = Control  ' Set reference
      mCol.Add Item  ' Add item to collection
   Else
      Set Item = mCol(IndexOf(Control))  ' Return item from collection
   End If
End Function

Public Function IndexOf(Control As Control) As Long
   Dim l As Long
   If mCol.Count > 0 Then  ' If there are any items...
      For l = 1 To mCol.Count  ' ...loop through them
         If mCol(l) Is Control Then  ' If the refrences match...
            IndexOf = l  ' ...return its position
            Exit For  ' Stop looping
         End If
      Next
   End If
```

```
End Function

Public Sub Remove(Control As Control)
   mCol.Remove IndexOf(Control)  ' Remove item from collection
End Sub

Public Sub SetAnchors()
   Dim oAO As AnchorObject
   For Each oAO In mCol  ' Loop through items
      oAO.SetAnchors  ' Set both anchors
   Next
End Sub

Public Sub DoAnchors()
   Dim oAO As AnchorObject
   If Not (mForm Is Nothing) Then Call LockWindowUpdate(mForm.hWnd)  ' Lock repainting
   For Each oAO In mCol  ' Loop through items
      oAO.DoAnchors  ' Do both anchors
   Next
   Call LockWindowUpdate(0)  ' Unlock repainting
End Sub


Private Sub mForm_Resize()
   Me.DoAnchors  ' Do all anchors
End Sub


Private Sub Class_Initialize()
   Set mCol = New Collection  ' Create new collection
End Sub

Private Sub Class_Terminate()
   Set mCol = Nothing  ' Discard collection
End Sub
```