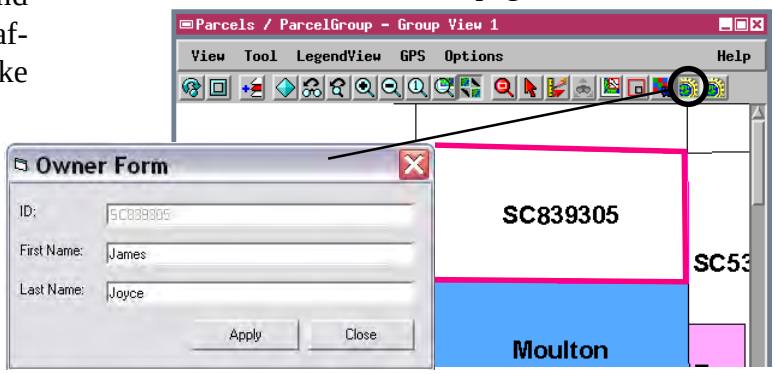
ActiveX Callbacks to SML

SML scripts can launch and communicate with ActiveX component software programs created in Visual Basic, C++, or Java. The method of communication between an SML script and an ActiveX dialog window that it launches depends on whether the dialog is set to operate as a *modal* or *modeless* dialog. A *modal* dialog in an ActiveX program takes sole control of SML script interactions, so that no further parts of the script are executed until the modal dialog closes. Using a *modeless* dialog in an ActiveX program allows the SML script to continue to execute even while the dialog is open.

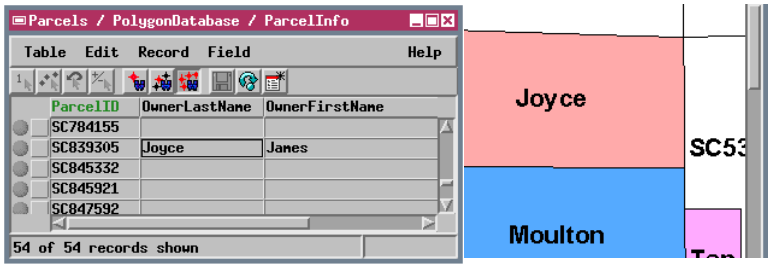
Communication between a modal ActiveX component dialog and an SML script is simple and indirect, since the script and dialog are not actually active at the same time. The SML script can pass data to "imported" component class members before the dialog opens and retrieve information from component class members after the dialog closes. The SML script can then take actions using or conditional upon the data retrieved.

Use of a modeless ActiveX component dialog requires ongoing communication between the dialog and the SML script. SML permits such communication, so that user interactions with a modeless component dialog can directly trigger actions by the SML script, such as the display or redisplay of a TNT View. When you create the ActiveX component class, you can define *events* associated with the dialog controls, such as the press of a particular button. Each event is provided with an associated class method that can be used in the SML script to register the name of a function elsewhere in the script that will be called and executed in response to that dialog event.

To provide an example of SML script interaction with a modeless ActiveX component dialog created in Visual Basic, MicroImages has modified the sample application described in the color plate entitled *Communicate with Visual Basic Programs using SML*. This application uses an SML tool script to provide interactive selection of vector polygons representing land parcels in a TNT View window, and a dialog opened by a Visual Basic program in which the user enters ownership information used to update the parcel database in TNTmips or some other relational database. A new ActiveX component class has been created to provide either modal or modeless versions of the dialog, depending on the class method called in the SML script. The sample tool script ParcelToolModeless.sml opens the modeless form of the dialog, which remains open as you select parcel polygons in the TNT View and update their ownership information in the Visual Basic dialog. This script includes callback functions registered with the Apply and Close buttons on the Visual Basic dialog. The sample tool script ParcelToolModal.sml opens the modal form of the dialog, which does not require SML callback functions. Excerpts of both of these scripts are shown on the other side of this page.



Modeless Visual Basic dialog "Owner Form" (in the imported class VBForm) that is launched by the ParcelToolModeless tool script. After the names are filled in, pressing the Apply button calls an `OnApply()` function in the tool script that reads the name fields from the VBForm class and updates the ParcelInfo database table and the TNT View window.



You can download the files required to run this demonstration from **microimages.com/freestuf/smlscripts.htm**. After downloading and unzipping the VBDEMO2 file, do the following in the VBDEMO2 directory:

1. Double-click on MicroImages_SML_OLE_Demo_EXE.exe to register the ActiveX component.
2. In Display, open the saved group ParcelGroup from the Parcels Project File. The sample tool scripts are pre-installed in this group.
3. Run the ParcelTool Modeless tool script.

An ActiveX component created in Visual Basic can be compiled as a simple executable file or as a dynamic link library (DLL). The latter method allows multiple instances of the component to run simultaneously. A modal ActiveX dialog can be activated from either form of component. However, if your component uses a modeless dialog, you must compile it as a simple executable file.

Excerpt of Tool Script ParcelToolModeless.sml

```
func OnLeftButtonPress () {
   if (checkLayer()) {
      local class POINT2D point;
      local numeric num;
      local string parcelID$;

      point.x = PointerX;        point.y = PointerY;
      point = View.GetTransLayerToScreen(vectorLayer).ConvertPoint2DInv(point);

      num = FindClosestPoly(targetVector, point.x, point.y, GetLastUsedGeorefObject(targetVector));

      if (num > 0) {
         elementNum = num;
         vectorLayer.Poly.HighlightSingle(elementNum);
         parcelID$ = targetVector.Poly[elementNum].ParcelInfo.ParcelID$;

         form.Clear();
         form.ElemID = parcelID$;
         form.LastName = targetVector.Poly[elementNum].ParcelInfo.OwnerLastName$;
         form.FirstName = targetVector.Poly[elementNum].ParcelInfo.OwnerFirstName$;

         form.ShowDialog();
         }
      }
   }

func OnApply () {
   if (elementNum > 0) {
         targetVector.Poly[elementNum].ParcelInfo.OwnerLastName$ = form.LastName;
         targetVector.Poly[elementNum].ParcelInfo.OwnerFirstName$ = form.FirstName;
         TableTriggerRecordChangedCallback( parceltable );
         vectorLayer.UnhighlightAllElements( );
         }
   }

func OnClose () {
   View.SetDefaultTool();
   }

func OnInitialize () {
   if (Layout) {
      WidgetAddCallback( Layout.GroupSelectedCallback, cbGroup );
      activegroup = Layout.ActiveGroup;
      }
   else
      activegroup = Group;

      form.SetOnApply( OnApply );
      form.SetOnClose( OnClose );
   }

func OnActivate() {
   form.ShowDialog();
   }

func OnDeactivate() {
   form.HideDialog();
   }
```

called when user presses 'left' pointer/mouse button

if the selected layer is valid, proceed

set local variables

get screen coordinates from cursor and transform to layer coordinates

get the element number of the enclosing polygon

highlight polygon; get the parcel ID for the polygon from the database

clear fields in VB Dialog window; pass the parcel ID and other parcel attributes to the *VBForm* class to show in dialog

open the Owner Form window as a modeless dialog; SML script remains active as long as tool is active

function called when the Apply button on the Visual Basic dialog is pressed

set values in parcel table from info entered in Visual Basic dialog

notify RVC that table has changed and unhighlight all elements

function called when the Close button on the Visual Basic dialog is pressed

reset tool icon buttons on View when VB dialog closes

function called the first time the tool is activated

register callback functions in the script with the Visual Basic dialog

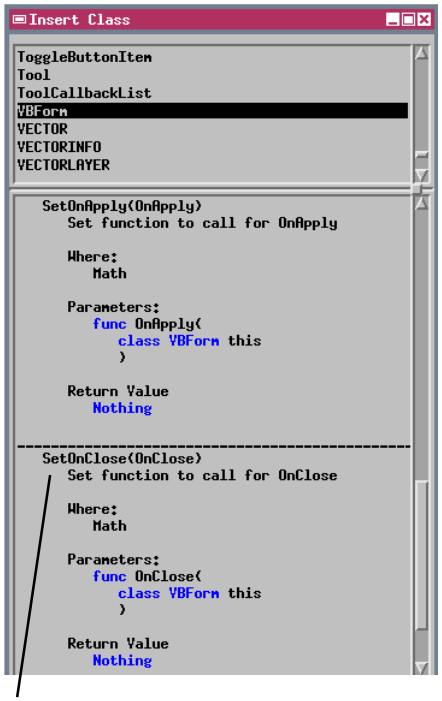
called each time the tool is activated

show the VB dialog

called each time the tool is deactivated

hide the VB dialog

This tool script opens the modeless version of the Owner Form Visual Basic dialog window. Functions in the tool script are called by the Apply and Close buttons on the dialog.



Each type of event defined for the Visual Basic dialog is provided with a method in the imported class that you can use to assign a function in the SML script that is called and executed when that dialog event occurs.

Excerpt of Tool Script ParcelToolModal.sml

```
func OnLeftButtonPress () {
   if (checkLayer()) {
         ... [same as
         above excerpt]...
      if (elementNum > 0)
         vectorLayer.Poly.HighlightSingle(elementNum);
         parcelID$ = targetVector.Poly[elementNum].ParcelInfo.ParcelID$;

         form.Clear();
         form.ElemID = parcelID$;
         form.LastName = targetVector.Poly[elementNum].ParcelInfo.OwnerLastName$;
         form.FirstName = targetVector.Poly[elementNum].ParcelInfo.OwnerFirstName$;

         form.DoModal();
                  <no script activity until the modal VB dialog closes>
         targetVector.Poly[elementNum].ParcelInfo.OwnerLastName$ = form.LastName;
         targetVector.Poly[elementNum].ParcelInfo.OwnerFirstName$ = form.FirstName;

         TableTriggerRecordChangedCallback( parceltable );
         vectorLayer.UnhighlightAllElements( );
      }
   }
```

No callbacks are required with a modal Visual Basic dialog because the VB dialog and SML script are not active simultaneously. After the VB dialog closes, the SML script merely reads information from the VBForm class.

highlight polygon

get the parcel ID and other attributes for the polygon from database

clear fields in VB Dialog window

pass the parcel ID and other parcel attributes to the *VBForm* class to show in dialog

open the Owner Form window as a modal dialog

set values in parcel table from info entered in Visual Basic dialog (now closed)

notify RVC that table has changed and unhighlight all elements