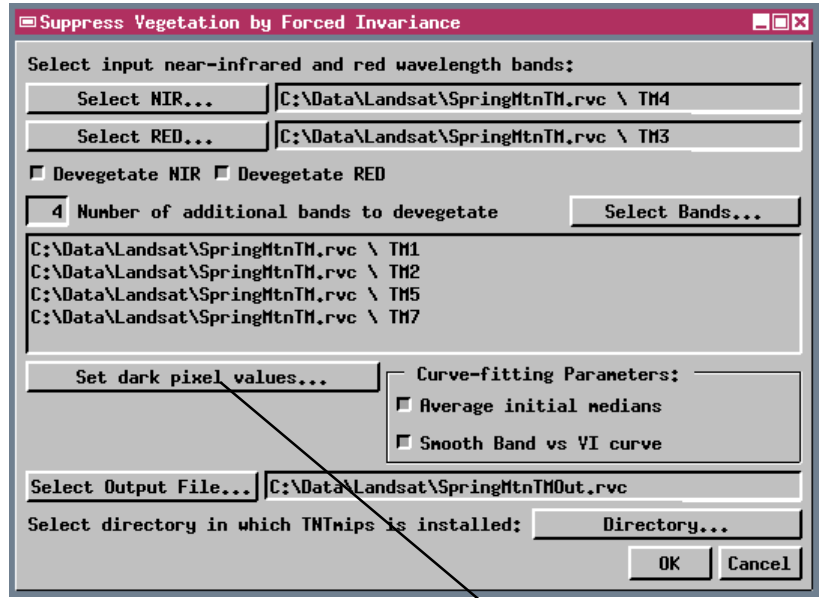# Nested SML Dialogs using XML

You can use an SML script to automate complex processing sequences involving multiple input and output objects of different types and varied processing parameters to be set by the user. Although SML provides standard pop-up dialogs for various types of user inputs, you can make the script easier to use by creating one or more custom dialog windows that bring together the various user input tasks.

The sample script devegX.sml (available for free download at www.microimages.com/freestuf/smlscripts.htm) provides a good example of a script that uses several custom dialog windows that are constructed using XML dialog specifications embedded in the SML script. The script is designed to process a multispectral image to suppress the expression of vegetation and enhance the contrast of background rock and soil (see the color plate entitled *Suppressing Vegetation in Multispectral Images*).

The main dialog window opened by the script is used to select input and output objects and to set most process parameters. It is a modal dialog, meaning that the window must be closed with its OK or Cancel button before the remainder of the script can be executed. The dialog window includes pushbuttons, toggle buttons, labels, edittext and editnumber controls, and a listbox, all of which are completely defined by XML code embedded in and interpreted by the SML script. Many of the controls are initially disabled, but are activated in sequence by callback procedures when required parameters higher in the dialog are set. This design leads the user through the controls and ensures that all required input objects and parameters are set before the OK button is activated.
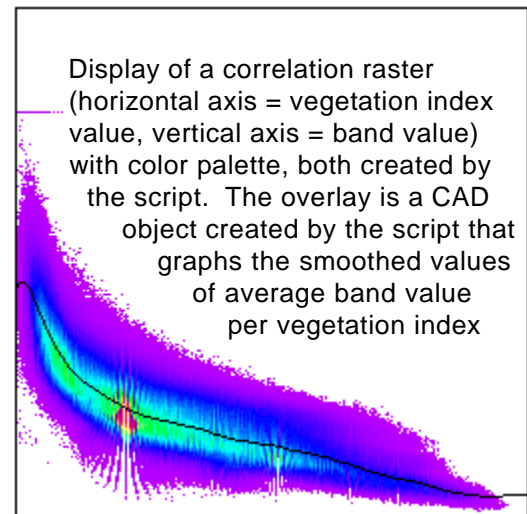
The Set Dark Pixel Values window is opened by pressing a push-button on the main window. This window presents a list of the selected input objects (as a sequence of label elements) and an accompanying numeric field to be edited by the user. Only the most general elements of this dialog can be specified by static XML code in the script; the list of labels and the numeric fields must be constructed dynamically from the input objects that the user has previously selected. The procedure called by the Set dark pixel values... pushbutton on the main window (shown on the reverse side of this plate) first reads the static XML code for the new window and parses it into memory. Additional XML elements corresponding to the object list and editnumber fields are then added to the XML structure before the dialog is opened.
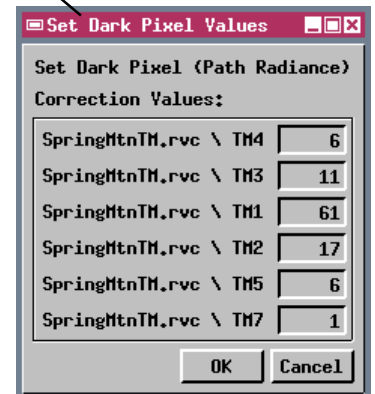
This script also provides examples of copying and modifying a color palette and creating and adding elements to a CAD object.

Display of a correlation raster (horizontal axis = vegetation index value, vertical axis = band value) with color palette, both created by the script. The overlay is a CAD object created by the script that graphs the smoothed values of average band value per vegetation index

Script for creating Set Dark Pixel Values window (devegX.sml)

Procedure to create dialog window to list input bands with fields to enter dark-pixel values. Called by pressing Set dark pixel values... pushbutton on main dialog window

```
proc SetDP () {

   xmldp$ = '<?xml version="1.0"?>
```

Create string variable with XML specification of dialog

```
   <root>
      <dialog id = "dpdlg" title = "Set Dark Pixel Values" OnOK = "dpOK()" >
         <label>Set Dark Pixel (Path Radiance)</label>
         <label>Correction Values:</label>
         <groupbox ExtraBorder = "3">
            <pane id = "dplist" Orientation = "Vertical"/>
         </groupbox>
      </dialog>
   </root>';
```

Create empty pane with id attribute to use when adding elements before the window is opened

```
   err = docdp.Parse(xmldp$);
   if ( err < 0 ) {
      PopupError( err );
      Exit( );
   }
```

Parse XML text for dialog into memory; return error if there are syntax errors

Modify the XML structure in memory before opening the dialog

```
   class XMLNODE dplist;
   dplist = docdp.GetElementByID( "dplist" );
```

Get the id for the parent pane that will contain the list of bands and the numeric fields for the correction values

Add horizontal pane for the first row of dialog elements (label and numeric field for NIR band)

```
   class XMLNODE paneNIR;
   paneNIR = dplist.NewChild( "pane" );
   paneNIR.SetAttribute( "Orientation", "Horizontal");

   class XMLNODE labelNIR;
   labelNIR = paneNIR.NewChild( "label" );
   nirprtext$ = sprintf("%s.%s \\ %s", FileNameGetName(nirfile$),
         FileNameGetExt(nirfile$), nirobjname$);
   labelNIR.SetText(nirprtext$);
```

Add label with name of NIR band to the NIR pane

```
   class XMLNODE prEditNIR;
   prEditNIR = paneNIR.NewChild( "editnumber" );
   prEditNIR.SetAttribute( "id", "prEditNIR" );
   prEditNIR.SetAttribute( "Width", "5" );
   prEditNIR.SetAttribute( "MinVal", "0" );
   prEditNIR.SetAttribute( "MaxVal", "255" );
   prEditNIR.SetAttribute( "Default", NumToStr(nirmin) );
   prEditNIR.SetAttribute( "Precision", "0" );
```

Add editnumber field to the NIR pane and set its attributes

Add horizontal pane for the second row of dialog elements (label and numeric field for RED band)

```
   class XMLNODE paneRED;
   paneRED = dplist.NewChild( "pane" );
   paneRED.SetAttribute( "Orientation", "Horizontal");

   class XMLNODE labelRED;
   labelRED = paneRED.NewChild( "label" );
   redprtext$ = sprintf("%s.%s \\ %s", FileNameGetName(redfile$),
         FileNameGetExt(redfile$), redobjname$);
   labelRED.SetText(redprtext$);
```

Add label with name of RED band to the RED pane

```
   class XMLNODE prEditRED;
   prEditRED = paneRED.NewChild( "editnumber" );
```

Add editnumber field to the RED pane and set its attributes

```
   prEditRED.SetAttribute( "id", "prEditRED" );
   prEditRED.SetAttribute( "Width", "5" );
   prEditRED.SetAttribute( "MinVal", "0" );
   prEditRED.SetAttribute( "MaxVal", "255" );
   prEditRED.SetAttribute( "Default", NumToStr(redmin) );
   prEditRED.SetAttribute( "Precision", "0" );
```

Loop to add rows for additional bands to be processed

```
   if ( numbands > 0 ) {

      class XMLNODE panepr;
      class XMLNODE labelpr;
      class XMLNODE editpr;
```

reusable XMLNODE class instances for new pane, label, and editnumber field for each additional band

```
      local string editid$;
```

string variable for id for editnumber element

```
      for num = 1 to numbands {
```

create horizontal pane for row

```
         panepr = dplist.NewChild( "pane" );
         panepr.SetAttribute( "Orientation", "Horizontal" );

         infile$ = inrastfilelist.GetString( num );
         inobjname$ = inrastobjlist.GetString( num );
```

Get file/object names for current band (stored in stringlists)

```
         labeltext$ = sprintf("%s.%s \\ %s", FileNameGetName(infile$),
               FileNameGetExt(infile$), inobjname$);

         labelpr = panepr.NewChild( "label" );
         labelpr.SetText( labeltext$ );
```

add label to the band pane

```
         editpr = panepr.NewChild( "editnumber" );
         editid$ = "editnum" + NumToStr( num );
         editpr.SetAttribute( "id", editid$ );
         editpr.SetAttribute( "Width", "5" );
         editpr.SetAttribute( "MinVal", "0" );
         editpr.SetAttribute( "MaxVal", "255" );
         editpr.SetAttribute( "Default", NumToStr( rastmins[ num ] ) );
         editpr.SetAttribute( "Precision", "0" );
```

Add editnumber field to the band pane and set its attributes

```
      }
   }
```

Get the dialog element from the parsed XML text and show error message if the dialog element can't be found

```
   nodedp = docdp.GetElementByID( "dpdlg" );

   if ( nodedp == 0 ) {
      PopupMessage( "Could not find dialog node in XML document" );
      Exit();
   }
```

Set the XMl dialog element as the source for the GUI_DLG class instance we are using for the dark-pixel-correction dialog window

```
   dlgdp.SetXMLNode( nodedp );


   ret = dlgdp.DoModal();
```

Open the dialog window as a modal dialog

```
}
```
end SetDP()