

---

# **Using the ArcView<sup>®</sup> Dialog Designer<sup>™</sup>**

(Version 1.0)

---

Environmental Systems Research Institute, Inc.

Copyright © 1997 Environmental Systems Research Institute, Inc.

All Rights Reserved.

Printed in the United States of America.

ESRI and ArcView are registered trademarks; Avenue, the ESRI corporate logo, the ArcView logo, and Dialog Designer are trademarks; and www.esri.com is a service mark of Environmental Systems Research Institute, Inc.

The names of other companies and products herein are trademarks or registered trademarks of their respective trademark owners. Portions of ArcView GIS created through use of Neuron Data's Open Interface software program. Copyright © 1991–1997 Neuron Data. All Rights Reserved.

The information contained in this document is the exclusive property of Environmental Systems Research Institute, Inc. This work is protected under United States copyright law and the copyright laws of the given countries of origin and applicable international laws, treaties, and/or conventions. No part of this work may be reproduced or transmitted in any form or by any means, electronic or mechanical, including photocopying or recording, or by any information storage or retrieval system, except as expressly permitted in writing by Environmental Systems Research Institute, Inc. All requests should be sent to Attention: Contracts Manager, Environmental Systems Research Institute, Inc., 380 New York Street, Redlands, CA 92373-8100 USA.

The information contained in this document is subject to change without notice.

#### RESTRICTED/LIMITED RIGHTS LEGEND

Use, duplication, and disclosure by the U.S. Government are subject to restrictions as set forth in FAR §52.227-14 Alternate III (g)(3) (JUN 1987), FAR §52.227-19 (JUN 1987), and/or FAR §12.211/12.212 [Commercial Technical Data/Computer Software], and DFARS §252.227-7015 (NOV 1995) [Technical Data], and/or DFARS §227.7202 [Computer Software], as applicable. Contractor/Manufacturer is Environmental Systems Research Institute, Inc., 380 New York Street, Redlands, CA 92373-8100 USA.

---

# Contents

## **Chapter 1 Welcome to the Dialog Designer 1**

- What is a dialog? 2
- Why should you use the Dialog Designer? 3
- How does the Dialog Designer work? 4
- What controls are available to you? 5
- Getting technical support from ESRI 6
- Visit ESRI on the Web 6

## **Chapter 2 Quick start tutorial 7**

- Exercise 1: Creating a dialog and adding a control to it 8
- Exercise 2: Making controls work together 12
- Exercise 3: Connecting your dialog to data and documents 15
- Exercise 4: Adding controls directly to views and layouts 21
- What next? 24

## **Chapter 3 Delivering an application with dialogs 25**

- Incorporating dialogs in an extension 26
- Delivering dialogs in a project 33
- Creating a personal working environment 34
- Creating a system default environment 34

## **Appendix A Control properties descriptions 35**

- Control properties 36
- Property descriptions 43

## **Appendix B Object model diagram 65**

- Object model 66
- Class descriptions 67

## **Index 71**



---

## CHAPTER 1

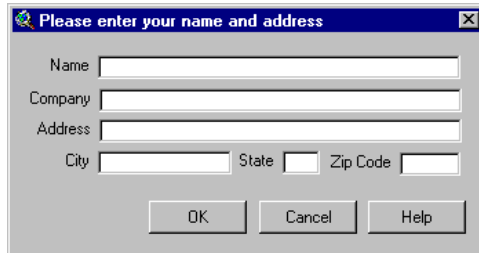
# Welcome to the Dialog Designer

Welcome to the ArcView® Dialog Designer™. The Dialog Designer provides Avenue™ developers with a new tool, a dialog, to customize ArcView's interface. You've seen dialogs that are part of ArcView's interface, you just couldn't build them yourself within ArcView's development environment—until now. No matter what kind of application you're developing, the Dialog Designer can help you create a more effective, user-friendly interface, tailored to your specific needs.

## What is a dialog?

A dialog is another user interface tool you can use to build applications in ArcView. A dialog lets you organize a single task or set of related tasks onto a separate window, much like you can organize related tasks under a particular menu item or on the button bar. For example, with the Dialog Designer, you can

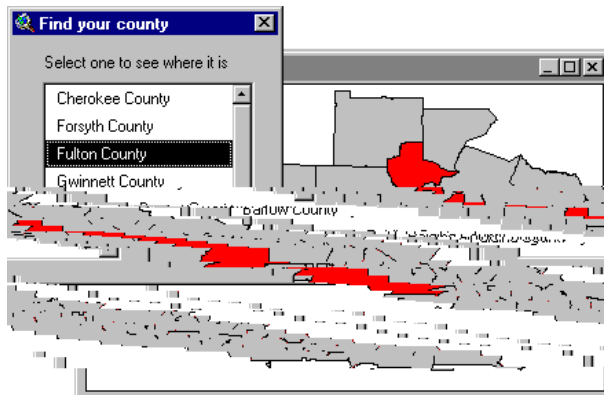
- Create your own input forms.



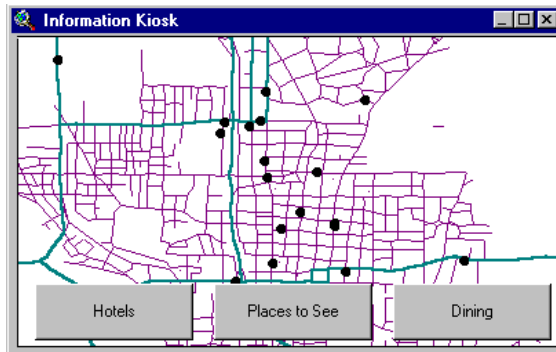
- Organize related tools in separate windows.



- Link what's displayed in a dialog directly to your data.



- Add buttons directly to a View or Layout.



The Dialog Designer only allows you to supplement ArcView's interface with dialogs that perform a specialized task. The Dialog Designer does not allow you to modify any of ArcView's existing dialogs or replace ArcView's main window and run your application completely from a dialog.

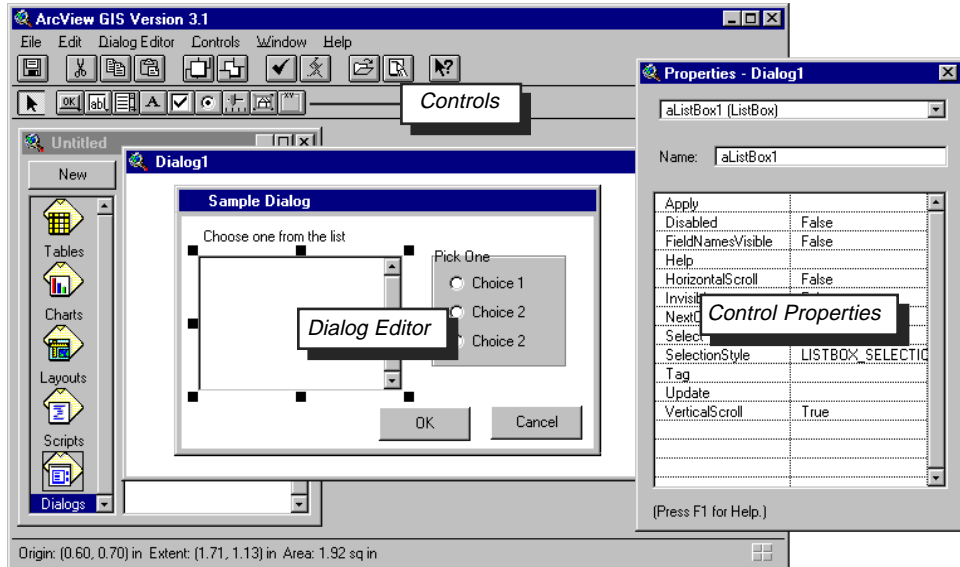
## Why should you use the Dialog Designer?

Before the Dialog Designer, you couldn't design and build custom dialogs using ArcView. Thus, you may have turned to other applications that run outside of ArcView, such as Microsoft® Visual Basic®, to provide the functionality you needed. While this works to a degree, getting the external dialog to communicate with ArcView is clumsy and inefficient. The Dialog Designer extension provides

- A cross-platform development environment that works on the PC, UNIX® workstation, and Macintosh®.
- Integrated dialogs that don't need to communicate with ArcView through Dynamic Data Exchange (DDE) or Remote Procedure Calls (RPC).
- Direct access to and manipulation of ArcView components such as tables, views, and themes.
- An easier way to distribute an application; everything can be included in an ArcView project or incorporated into an extension.

## How does the Dialog Designer work?

To use the Dialog Designer extension, you first load it into ArcView—just like you load any other ArcView extension. Once you do, you'll notice a new document in the project window called Dialogs. Double-click the icon and you'll create a new dialog editor document that's ready to accept the interface components you want on your dialog—buttons, sliders, and check boxes.



All of the interface components, or *controls*, you can add to your dialogs are located on the tool bar. You simply choose the ones you want and arrange them on the dialog. Each control has a set of properties that define it, listed on the Control Properties dialog.














Once you've added a set of controls to your dialog, you'll write the Avenue scripts that make it work. For instance, you'll write a script that defines what happens when someone clicks a button on your dialog. Then, you'll attach your dialog to ArcView's interface so people can use it. The next chapter, 'Quick start tutorial', contains step-by-step exercises that describe this process in detail. Last, if you want people to use your dialog outside of the current project, you'll need to package it in an extension as described in Chapter 3, 'Delivering an application with dialogs'.



## What controls are available to you?

The Dialog Designer contains a number of different controls you can include on your custom dialogs. Here's a description of each one.

---

Control	Description
	A <i>label button</i> performs a task when you click it. The text on the label button describes its action.
	A <i>button</i> likewise performs a task when you click it, but with an icon to illustrate its action. Buttons on a dialog work the same way as buttons on the button bar.
	A <i>tool</i> performs an action when you interact with a document such as a view. Tools on a dialog work the same way as tools on the tool bar.
	A <i>text line</i> accepts one line of information from you or displays information provided by the application.
	A <i>text box</i> accepts multiple lines of information from you or displays multiple lines of information provided by the application.
	A <i>text label</i> displays static text such as captions and instructions for using the dialog.
	A <i>list box</i> displays a single or multicolumn list of choices. The columns of the list box cannot be edited.
	A <i>combo box</i> displays a single column list of choices that appears to drop down when you click it.
	A <i>check box</i> allows a user to turn on (check) or turn off (uncheck) an option on the dialog.
	A <i>radio button</i> can be combined with other radio buttons to provide an exclusive set of choices.
	A <i>slider</i> allows you to set a discrete value within a finite range of values. A slider can be oriented horizontally or vertically.
	An <i>icon box</i> displays a static picture on the dialog, such as a company logo. Acceptable file formats include TIFF, GIF, and bitmap files.
	A <i>control panel</i> groups related controls on a dialog. You don't interact with a control panel.

---

## Getting technical support from ESRI

Please see the product registration and support card that came with ArcView, or look at the 'Obtaining technical support' section of ArcView's on-line help.

## Visit ESRI on the Web

Find out everything you want to know about ESRI products and services. Visit ESRI's Web home page at [www.esri.com](http://www.esri.com).

---

## CHAPTER 2

# Quick start tutorial

This chapter shows you how to begin building your own dialogs and provides you with step-by-step tutorials that illustrate common tasks and describe the fundamental concepts. This chapter assumes that you are already familiar with ArcView's development environment and have used it before to produce customized ArcView applications. If you can write an Avenue script, then you'll be able to make a dialog do the things you want it to do in no time at all.

Read this chapter to learn about how to:

- Create a dialog and add controls to it.
- Make controls interact with each other.
- Connect your dialog to your data.
- Add controls on top of your views and layouts.

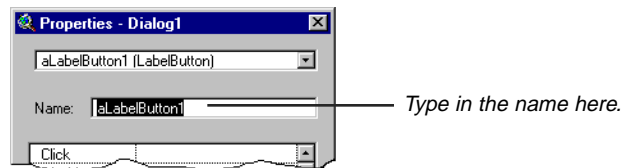
## Exercise 1: Creating a dialog and adding a control to it

A dialog is simply a container for a set of *controls*—such as a button, a slider, and a check box—arranged in an intuitive manner to collect information and perform some action. You’ve already worked with buttons and tools while modifying ArcView’s interface. When you add a button to the interface, you place it on the button bar and you

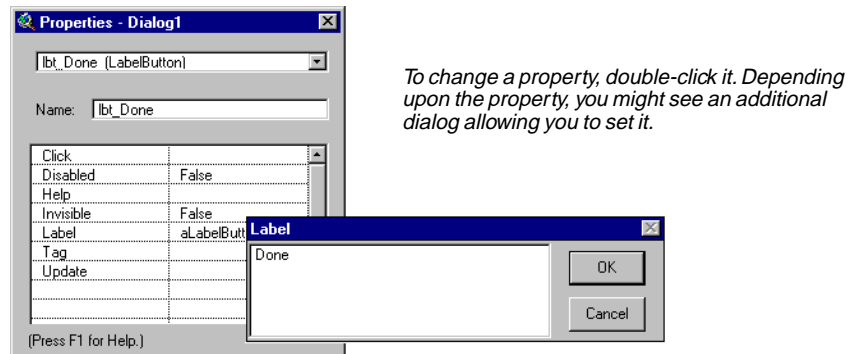
**Tip** Some tools on the tool bar are drop-down tool menus. For example, click and hold the mouse over the Label button tool to reveal other controls you can add to your dialogs.

### Set the control properties

4. Display the label button's properties by double-clicking it on the dialog. ArcView displays the list of properties in the Control Properties dialog. Here you'll define the label button, such as giving it a name or setting a script to run when you press the button. Each property has equivalent Avenue requests that allow you to manipulate the control in your scripts at run time.
5. In the Name field, type in 'lbt\_Done'. Later in this chapter, you'll see how to use a control's name to identify it in scripts.



6. Double-click on the Label property and type in 'Done'.





### Attach a script to the control

7. Locate the Click property in the scrolling list and double-click it to display the Script Manager dialog. Press New to create a new script and name the script 'Dialog1.lbt\_Done.Click'. The Control Properties dialog will clear because the active document is no longer a dialog editor.
8. Type in the following into the new script and compile it. This script will dismiss the dialog when you press the label button.

```
self.GetDialog.Close
```

## Compile and run the dialog

9. Make the dialog the active document. Press the Compile button  then the Run button . A new dialog will appear that contains the button you added. Using the Run button to display the dialog allows you to check that it works as expected. Once you've finished building a dialog, you'll probably call it from a script attached to a menu or button in the interface, as you'll see how to do in the next exercise.
10. Press the Done button on your dialog to close it. You can also press the escape key to dismiss a dialog. This is a convenient way to dismiss a dialog if you forget to include a control on your dialog to do so.
11. Try changing some other properties to see the effect they have on the dialog. For example, display the properties for the dialog itself by double-clicking over an unoccupied area of the dialog. Toggle the HasTitleBar property, then compile and run the dialog again to see the result.
12. Save your project (optional). The next time you open this project, the Dialog Designer will load automatically.

### Keeping track of dialogs, scripts, and controls

When developing an application, the number of dialogs, controls on dialogs, and scripts associated with dialogs gets quite large. Keeping track of all these components can be difficult if you don't establish some method for managing them. One convention to set in place early is a naming convention that will make it easy to find and reference these components.

#### Naming dialogs

To ensure that your dialog names don't conflict with dialogs in ArcView or any extensions (either developed by ESRI or third party developers), we recommend that you prefix your dialog names with a unique identifier. For example, you might use the name of the application you're creating as the prefix,

<application name>.<dialog name>

or some other prefix such as the name of your organization.

#### Naming scripts

It's a good idea to give your scripts descriptive names so that you can easily reference them. Here's a naming convention that will keep the scripts associated with a dialog listed together in the project window and allow you to quickly find the one you want to edit.

Use the name of the dialog, followed by the control name and the property the script is attached to.

```
<dialog name>.<control name>.<property name>
```













For example, the name of the script attached to the Click property of a button named 'lbt\_Done' would be 'MyDialog.lbt\_Done.Click'.

Avoid common names that may conflict with other projects or system script names. Avoid, for instance, using the word 'Dialog', the first word of many of the system scripts associated with the Dialog Designer.

### Naming controls

If you use a standard naming convention for your controls, they'll be easy to reference in your scripts. One way to do this is to give each control a prefix that identifies its type, followed by some descriptive name, such as its visible label. For instance, an OK button on your dialog would be named 'lbt\_OK'. Control names on a particular dialog should be unique. However, you can use the same name for controls on different dialogs.

Here's a table of suggested prefixes.

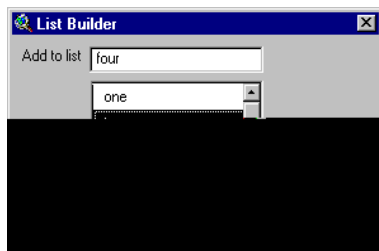
Prefix	Control Type	Prefix	Control Type
lbt	 - Label button	cbx	 - Combo box
btn	 - Button	chk	 - Check box
tol	 - Tool	rad	 - Radio button
txl	 - Text line	sld	 - Slider
tbx	 - Text box	ibx	
	- Text label	cpa	 - Control panel
lbx	 - List box		

## Exercise 2: Making controls work together

You won't often make dialogs with just one control on them. Most likely, your dialogs will contain many controls that you'll want to work together. For instance, you might create a dialog where entering some text—a name and address—enables an OK button, or checking a check box disables some other controls on the dialog. How do you make these controls interact with each other? By writing scripts that implement the desired behavior.

As you saw in the previous exercise, pushing a button on the dialog can execute a script. The same is true for typing into a text line, selecting an element in a list box, or moving the handle of a slider. Each control responds to certain types of interactions as defined by the Control Properties for the control. Through scripts attached to these properties, you control what the dialog does.

In this next exercise, you'll create a dialog that adds and deletes elements from a list. The dialog will look something like this:







*Text you type into the text line is added to the list. As long as there are elements in the list, the Delete button is available to delete the selected element.*

Once you finish creating the dialog, you'll learn how to make it accessible from ArcView's interface.

### Create a new dialog and add controls to it

1. Double-click the dialog icon in the project window.
2. Double-click on the new dialog editor to display the Control Properties dialog. Change the name of the dialog to 'ListBuilder'.
3. Add the following controls to the dialog and set their properties as specified in the table below. For properties associated with scripts, just create new scripts. The actual code you'll put in these scripts is included in the following steps.



Control	Property	Setting
 Label button	Name	lbt_Done
	Click	ListBuilder.lbt_Done.Click
	Label	Done
 Label button	Name	lbt_Delete
	Click	ListBuilder.lbt_Delete.Click
	Label	Delete
 Text line	Name	txl_Add
	Apply	ListBuilder.txl_Add.Apply
	Label	Add to list
 List box	Name	lbx_Current
	HorizontalScroll	True
	Select	ListBuilder.lbx_Current.Select
Dialog	Title	List Builder
	Open	ListBuilder.Open

### Write scripts to make the dialog work

4. Write `ListBuilder.Open`. This script runs every time the dialog opens. An `Open` script typically contains statements to initialize controls on the dialog.

```
'ListBuilder.Open
'Clears out the list box and disables the delete button
theListBox = self.FindByName("lbt_Current")
theListBox.Empty
theListBox.GoColumn(0)
theListBox.SetColumnWidth(3)
self.FindByName("lbt_Delete").SetEnabled(false)
```

This `Open` script clears out any elements in the list box, sets the width of the list box, and disables the Delete button. The `self` object references the control or dialog the script is attached to. Here the `self` object is the dialog; in the `Click` script of a label button, the `self` object is the label button.

5. Write `ListBuilder.lbt_Done.Click`. This script closes the dialog.

```
'ListBuilder.lbt_Done.Click
'Attached to the Done button to close the dialog
self.GetDialog.Close
```

- Write `ListBuilder.lbt_Delete.Click`. This script deletes the highlighted element in the list. If there are no more elements in the list, the script also disables the Delete button.

```
'ListBuilder.lbt_Delete.Click
'Attached to the Delete button to remove selected elements from list
theListBox = self.GetDialog.FindByName("lbt_Current")
theListBox.DeleteRows(1)
theListBox.SelectCurrent(false)
if (theListBox.GetRowCount = 0) then
    self.SetEnabled(false)
end
```

- Write `ListBuilder.txtl_Add.Apply`. This script adds an element entered into the text line to the bottom of the list.

```
'ListBuilder.txtl_add.Apply
'Attached to the text line; adds what's typed in to the list.
aNewElement = self.GetText
theListBox = self.GetDialog.FindByName("lbt_Current")
theListBox.GoRow(theListBox.GetRowCount)
theListBox.InsertRows(1)
theListBox.SetCurrentValue (aNewElement)
'Uncomment the next line and the list will be sorted
'theListBox.SortAscending(false)
self.SetText(" ")
```

To manipulate the list box, you explicitly move to the specific row you want to operate on. In this case, the script inserts a new row after the last row. (Note: This script will accept the return key and enter a blank line in the list.)

- Write `ListBuilder.lbx_Current.Select`. This script executes when you select an element in the list box. Doing so enables the Delete button.

```
'ListBuilder.lbx_Current.Select
'Attached to the list box, this script enables the Delete button.
self.GetDialog.FindByName("lbt_Delete").SetEnabled(true)
```

### Attach the dialog to the interface and run it

At this point, you can run the dialog by pressing the Run button. But, you'll likely want to attach the dialogs you create to ArcView's interface. For example, you might want to run your dialog from a button on a view document. This next step shows you how to do this.

- Create the following script and attach it to a button on ArcView's button bar. (You can attach it to a button on a view document or any other document.)

```
'ListBuilder.Run
'Runs the dialog.
av.GetProject.FindDialog("ListBuilder").Open
```

Alternatively, you could have found the Dialog Editor document and gotten the Dialog object from that object like this:

```
av.GetProject.FindDoc("ListBuilder").GetDialog.Open
```

The above statements search the current project for dialogs. If your dialogs are loaded from an extension, you can open them using the following request:

```
av.FindDialog("ListBuilder").Open
```

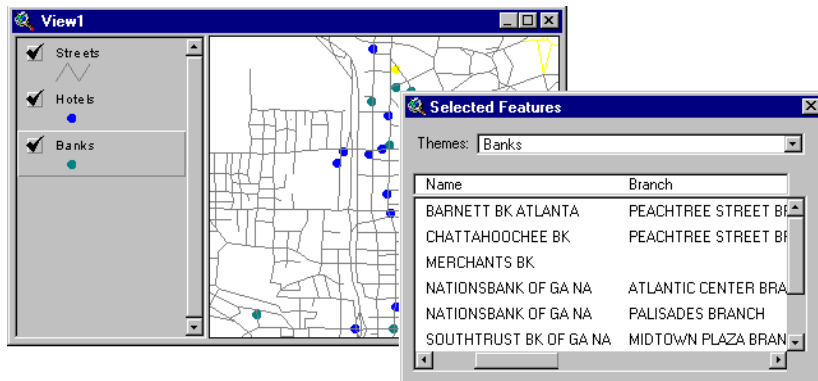
The next chapter discusses extensions and other ways to distribute your dialogs.

10. Now you can press the button you just added to run the dialog.

## Exercise 3: Connecting your dialog to data and documents

So far you've seen how to build some simple dialogs that run independently—they don't respond to or interact with any ArcView documents or the data displayed in the documents. Most likely, you'll want your dialogs to respond to something that's currently happening in ArcView.



In this exercise, you'll create a dialog that displays the attributes of the selected records in a theme. This dialog will monitor the selected features and update itself when the selected set changes. If a new document becomes active, the dialog will either disable itself if the new document is not a view, or update itself based upon the contents of the new view. Here's a preview of the dialog you'll create:



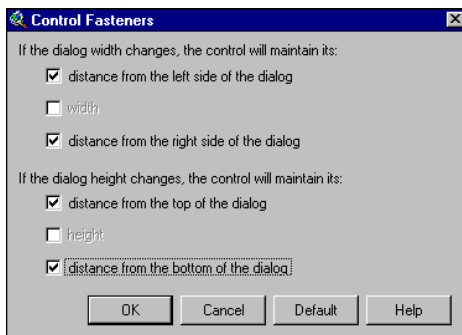
### Connect a dialog to a view

1. Add a view with a few themes in it to your project.
2. Double-click the dialog icon in the project window to create a new dialog. Name this dialog 'SelectedFeatures'.

3. Add the following controls to the dialog, arranged as in the figure above. Set the properties as specified in the table below. For those properties that are associated with scripts, just create new scripts. You'll fill in the code for these scripts later.

Control	Property	Setting
 Combo box	Name	cbx_Themes
	Label	Themes:
	Select	SelectedFeatures.cbx_Themes.Select
	Update	SelectedFeatures.cbx_Themes.Update
 List box	Name	lbox_Records
	FieldNamesVisible	True
	HorizontalScroll	True
	Update	SelectedFeatures.lbox_Records.Update
Dialog	Name	SelectedFeatures
	DocActivate	SelectedFeatures.DocActivate
	Open	SelectedFeatures.Open
	ServerSelectionChanged	SelectedFeatures.ServerSelChanged
	Title	Selected Features

4. Select the list box on the dialog editor. Then, from Controls, choose Control Fasteners. Uncheck the width and height options to enable the other check box options and then check the options to maintain the distance from the bottom and right side of the dialog as shown in the figure below.



The list box will now change size whenever you resize the dialog (the default is to stay the same size). For more information, press the Help button on the Control Fasteners dialog.

## Programming each control

In the preceding exercise, controls were enabled and disabled by whichever script was executing at the time. Selecting an element in the list box, for instance, enabled the Delete button because the script attached to the Select property of the list box searched for it and enabled it. However, as the number of controls increases, it becomes more difficult to maintain and update your dialog if several different scripts manipulate a given control. A better approach is to program your dialog so that each control is responsible for the actions it needs to perform.

As you've seen, a control executes an action by running a script; for instance, the Click script of a button. Each control also has an Update script that manages the "state" of the control—for instance, whether or not it's enabled. In this exercise, you'll write Update scripts to manage the state of each control on the dialog.

5. Write SelectedFeatures.Open. This Open script initializes the dialog. It runs the Update scripts of the combo and list boxes by sending the Update request to them.

```
'SelectedFeatures.Open
cbx = self.FindByName("cbx_Themes")
cbx.Update
lbox = self.FindByName("lbox_Records")
lbox.Update
cbx.SetListeners({lbox})
```

As you'll see below, the Update scripts are responsible for initializing the controls and enabling and disabling them.

The last line of this script allows the combo box to execute the list boxes' Update script. The Update script is not actually run here; the code simply establishes a *broadcaster–listener* relationship between the combo box and the list box. Thus, whenever the combo box broadcasts a message, all the controls that are listening (in this example there's only one) will run their Update scripts. The advantage of establishing this kind of relationship is that it is dynamic; controls can be added to or removed from this relationship any time. For more information on how Update scripts execute, search the on-line Help for 'dialogs, updating controls on'.

6. Write SelectedFeatures.cbx\_Themes.Select. The primary control on this dialog is the combo box. It displays the current themes in the active view. As you select different themes, the list box updates to show the attributes of the selected features of the theme.

```
'SelectedFeatures.cbx_Themes.Select
aTheme = self.GetCurrentValue
aVTab = aTheme.GetFTab
self.GetDialog.SetServer(aVTab)
self.BroadcastUpdate
```

Selecting a theme listed in the combo box runs the above script. The script first gets the selected theme from the combo box, then gets that theme's VTab (FTab). Then, the script links the dialog in a client/server relationship with the VTab. Once done, the dialog (client) can respond when certain things happen to the VTab (server) such as when the selection changes, when records are added or deleted, or when the definition changes. In this example, the dialog will respond to a change in selection on the VTab.

The last line of this script broadcasts the Update event. Recall that the Open script established the list box as a listener to the combo box. When the combo box broadcasts its Update event, all controls that are listening run their Update scripts—in this case only the list box is listening. The list box Update script populates the list box with the attributes of the current selected features of the newly selected theme. To see how this works, look at this next script.

7. Write SelectedFeatures.lbx\_Records.Update. This script performs all the functions related to the list box. It determines whether or not the list box should be enabled (when the active document is a view that has themes in it) and it also ensures the list box displays the attributes of the current selected features.

```
'SelectedFeatures.lbx_Records.Update
aDoc = self.GetDialog.GetActiveDoc
if (aDoc.Is(View) and aDoc.GetThemes.IsEmpty.Not) then
  self.SetEnabled(true)
  aVTab = self.GetDialog.GetServer
  flist = aVTab.GetFields
  numFields = flist.Count
  self.DefineFromVTab(aVTab, flist, true)
  self.FitColumns(0..(numFields - 1),false)
else
  self.SetEnabled(false)
end
```

This script runs when you first open the dialog, when you select a new theme in the combo box, when you change the selected set of features in the current theme, or when you make a new document active. Whenever any one of these things happens, the script redefines the list box based upon the new parameters (i.e., a new theme or selected set). The list box defines its contents from the server VTab set up in the previous step. You can get the server VTab object from the dialog and use it to define the contents of the list box. In this case, the list box displays the attributes of all the fields in the VTab. You can easily build your own list of selected fields.

8. Write SelectedFeatures.cbx\_Themes.Update. This script makes sure that the combo box displays the correct information. It runs when the dialog is first opened (called from the Open script) and also, as you'll see below, when a new document is made active (DocActivate script).

```
'SelectedFeatures.cbx_Themes.Update
aDoc = self.GetDialog.GetActiveDoc
if (aDoc.Is(View) and aDoc.GetThemes.IsEmpty.Not) then
```

```

self.SetEnabled(true)
theThemes = aDoc.GetThemes
self.DefineFromList(theThemes)
self.GetDialog.SetServer(theThemes.Get(0).GetFTab)
else
self.SetEnabled(false)
end

```

Just like the list box Update script, this script disables the combo box when the active document is not a view or when there aren't any themes in the view. Notice that the script also sets the first theme in the view as the server VTab. That's because this script runs when there is no current theme—when the dialog is first opening, or when a new view with a new set of themes is activated.

9. Write SelectedFeatures.ServerSelChanged. Even though you've linked the dialog to the VTab, you still need to define what happens when the selection changes. You do this by assigning a script to the ServerSelectionChanged property of the dialog. ArcView runs this script whenever the selection in the server VTab changes.

```

'SelectedFeatures.ServerSelChanged
self.FindByName("lbx_Records").Update

```

Because the Avenue code that manipulates the list box is placed in the Update script of the list box, this script just calls it. The list box then updates itself to reflect the new selected features. You could put the code that updates the list box directly in this script, but by confining the code to the scripts associated with the list box you don't duplicate code or manipulate the control from more than one script.

10. Write SelectedFeatures.DocActivate. This script runs whenever you make a new document active.

```

'SelectedFeatures.DocActivate
self.FindByName("cbx_Themes").Update
self.FindByName("lbx_Records").Update

```

The script calls the Update scripts of the list box and combo box. Both these scripts make sure the information displayed in the dialog reflects the active view and the themes in that view.

### Attach the dialog to the interface and run it

11. Attach the dialog to a button on the View document.

```

av.FindDialog("SelectedFeatures").Open

```

12. Open the view you created in step 1. As you select features in a theme, the dialog should update to reflect the selection (as long as the theme is the one being displayed in the dialog). If you make another view active, the dialog should also update and display the themes in that view.

## Making your dialog react to changes in ArcView

The exercises in this chapter illustrate some of the ways a dialog can react when certain events happen in ArcView such as when a document becomes active. For each event, you can write a script to make your dialog respond appropriately. Here's a summary of the various ways you can make your dialog respond.

### Respond when the dialog becomes active

If you need to run a script whenever the dialog becomes active, you set the dialog's Activate property to the particular script you want to run. In the script, you perform whatever actions are necessary. You can set the Activate property either from the Control Properties dialog or with the SetActivate request on a Dialog.

### Respond when the active document changes

Suppose you create a dialog that contains tools that work with views and layouts. As long as a view or layout is active, the tools on your dialog should be enabled; otherwise, they should be disabled. You can write a script to enable or disable the controls and attach it to the dialog's DocActivate property. The DocActivate script runs whenever the active document changes. For example, you might write a DocActivate script like this:

```
'Dim control when active doc is not view or layout
'the self object is the dialog
theDoc = self.GetActiveDoc
enabled = theDoc.Is(View) or theDoc.Is(Layout)
self.GetControlPanel.SetEnabled(enabled)
```

### Respond to a change in a particular document

Suppose you've created a dialog that provides extra information about an individual view document. You want the dialog to appear when the view is open and disappear when closed. To do this, you need to link the dialog directly to the view document so that it can respond to changes such as closing the view. This involves establishing a client/server relationship between the dialog and the document, where the dialog is the client to the server document.

A dialog can respond to a limited number of server events and run a script attached to the appropriate dialog property when that event occurs. These server events are

- ServerActivated - The associated document is activated.
- ServerDeactivated - The associated document is deactivated.
- ServerOpened - The associated document is opened.
- ServerClosed - The associated document is closed.
- ServerSelectionChanged - The selection on the VTab is changed.
- ServerDefinitionChanged - The definition of the VTab is changed.



- ServerRecordsAdded - A record is added to the VTab or table document.
- ServerRecordsDeleted - A record is deleted from the VTab or table document.

To set up a server, you use the SetServer request on a Dialog. The script you actually put this request in depends upon what you're trying to accomplish. For example, you might put this request in a dialog's Open script:

```
'Server set in Open script  
self.SetServer(av.GetProject.FindDoc("View1"))
```

### Respond to other changes

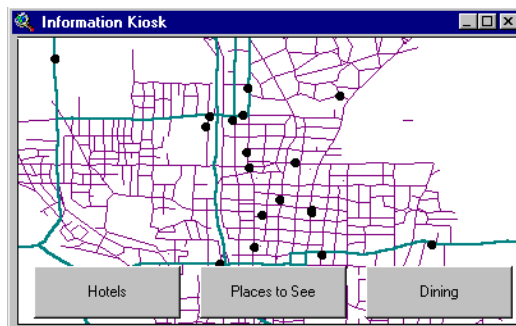
Your dialog can respond to almost anything that can happen during a session. However, not every event that can happen in ArcView has a dialog property you can set. Sometimes you'll need to do a bit of programming to make your dialog respond accordingly.

For example, you can attach an Update script to a document's button bar (ControlSet) and directly call the Update scripts of the controls in your dialog. The button bar's Update script will run as a result of an Update event sent by ArcView, just as the individual controls on the button bar automatically run their Update scripts. The Update scripts of each control on your dialog would then determine what sort of change occurred and respond accordingly.

For more information on any of the above, search the on-line Help for 'dialogs, responding to documents'.

## Exercise 4: Adding controls directly to views and layouts

So far, you've seen how to add controls to a dialog and make them work with other controls and with your data. A dialog, however, isn't the only place you can put controls. If you want to, you can add them directly on top of your views and layouts, making the document act as its own dialog. For instance, you might make an application that contains only a few buttons added directly on top of a view:

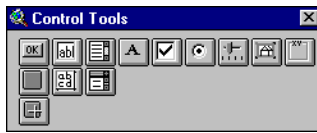


Adding controls to views and layouts works well when the application you're creating is designed to work with a particular data set. If your application requires more than just a few controls, or they need to work with any data set, you're probably better off putting the controls on a dialog.

In this exercise, you'll add a text line control on top of a view, then use that control to enter X,Y coordinate pairs. As you enter coordinates, the script associated with the text line will draw a line connecting the coordinates.


### Add a control to a view

1. Create a new, empty view in your project.
2. From Window, choose Show Control Tools. You should see the palette of controls you can add on top of your view. The controls on this palette are the same ones you add to a dialog, except that the check box and radio button will be disabled, as they can't be added to views and layouts.



3. Select the Text line tool.
4. Add a text line control to the bottom of the view window. Click and drag the mouse to position and size the control over the view. Alternatively, you can click once to add the control with a default size.
5. Double-click the control to display the Control Properties dialog and set the following properties:

---

Control	Property	Setting
 Text line	Name	txl_Input
	Apply	View1.txl_Input.Apply
	Label	Enter X,Y:
	When the map pans/ zooms the control will:	Pan with the map

---

## Write scripts that make the control work

6. Write the following script and name it View1.Initialize. This script sets the object tag of the control to be its graphic control. You'll need to substitute the name of your view in this script.

```
aGControl = av.FindDoc("View1").GetGraphics.FindByName("txl_Input")
aGControl.GetControl.SetObjectTag(aGControl)
```

Controls you add to views (and layouts and dialog editors) have a graphic component (GraphicControl class) that you manipulate much like any other graphic object you might add on top of your view. To find the graphic control, you can search in the view's graphic list. As you'll see in the next script, you'll use the text line's graphic control to move it across the view.

7. Write View1.txl\_Input.Apply.

```
theView = av.GetActiveDoc
txl_Input = self.GetObjectTag
'the first coordinate is the origin of the text line
fromP = txl_Input.GetOrigin
xyCoord = txl_Input.GetControl.GetText.AsTokens(",")
toP = Point.Make(xyCoord.Get(0).AsNumber,xyCoord.Get(1).AsNumber)
'draw the line
l = Line.Make(fromP, toP)
gl = GraphicShape.Make(l)
theView.GetGraphics.Add(gl)
'move the text line
txl_Input.Invalidate
txl_Input.SetOrigin(toP)
txl_Input.Invalidate
txl_Input.GetControl.SetText("")
```

This script draws a line between the current position of the text line control and a map coordinate entered into the text line. Then, it moves the text line to the end of the line, ready to accept the next coordinate.

## Activate the control

8. Run View1.Initialize.
9. Activate the text line control. From the View menu bar, choose Graphics, then Run Controls. This toggles the control from design mode to run mode.
10. Type in an X,Y map coordinate pair into the text line. To find a valid coordinate, move the mouse over the view window. At the right side of the tool bar is a coordinate display. Using the display as your guide, enter in a coordinate into the text line. Make sure you use a comma to separate the X and Y coordinates.

11. Optionally, append this code to View1.txl\_Input.Apply. This code will add a text label, annotating the line with the coordinate pair you typed in.

```
theDisplay = theView.GetDisplay
txt_Label = TextLabel.Make
txt_Label.SetLabel(fromP.GetX.SetFormat("d.dd").AsString++
    fromP.GetY.SetFormat("d.dd").AsString)
aWidth = theDisplay.ReturnVisExtent.GetWidth
'The next line sizes the text label based on the view's
'display. You may need to change it if the text gets clipped.
aSize = (aWidth * 0.3)@(aWidth * 0.04)
r = rect.Make(fromP,aSize)
gt = GraphicControl.Make(txt_Label, r)
theView.GetGraphics.Add(gt)
gt.SetEditable(false)
gt.SetConstraint(#GRAPHICCONTROL_CONSTRAINT_POSITION)
```

To create a text label through a script, you use the `TextLabel.Make` request. Then, you create a graphic control from the control and add it to the view's graphic list. Although a text label is a static control, you can activate it with the `SetEditable` request, thus preventing it from being edited accidentally. The last request allows the label to pan with the map.

## What next?

We hope you've enjoyed working through the exercises in this tutorial and found them to be a useful introduction to the Dialog Designer. To learn about the various ways to distribute dialogs, read the next chapter. As you begin building your own dialogs, refer to the on-line Help to answer questions you have while working at your computer. The on-line Help contains descriptions of common tasks—with Avenue code examples—and a complete reference for all classes and requests.

## CHAPTER 3

# Delivering an application with dialogs

Depending on your application, there are several ways to distribute your custom dialogs. If you'll be using your dialogs with different projects or you plan to distribute them to others either within or outside your organization, you could create an ArcView extension. If you've designed your dialogs for a specific project, you could store and deliver them in that ArcView project. If the functionality in your dialogs is something that only you'll use, you could create a personal working environment. If everyone in your organization needs your dialogs for their work, you could create a system default environment.

In this chapter you'll learn how to:

- Create an extension containing dialogs.
- Deliver dialogs in a project.
- Create a personal working environment that includes your dialogs.
- Create a systemwide environment for everyone.

## Incorporating dialogs in an extension

Extensions provide a powerful and flexible mechanism for delivering customized functionality to end users. Because an extension can contain any type of ArcView object, you can easily incorporate your dialogs and associated scripts into them. The objects in an extension are independent of those stored within a project; thus, they are not replicated in or saved to the project. The advantage of this is if you need to modify or update your custom dialogs, you can update your extension and deliver a new version instead of updating each project that uses it. A user simply loads the new version of your extension to get the updated dialogs.

Extensions start out as projects. The first step in building an extension is to create a project that contains the customizations—including dialogs—you want to deliver. This project is called the extension's source project. The extension you'll build here uses the Selected Features dialog created in Exercise 3 of the previous chapter. If you don't have access to this dialog, you can substitute one of your own.

The discussion below assumes you already know a little about building extensions. If you don't, refer to Chapter 5, 'Storing and delivering customizations' in *Using Avenue* or search the on-line Help for 'Extensions'.

### Open the project containing your customizations

1. Open the project that contains the Selected Features dialog. This project will become the source project for the extension. It should contain the dialog, a new button on the view's interface to launch the dialog, and all the associated scripts. If you don't have this project, you can substitute another project that contains a dialog. However, the scripts below assume that you have these components; thus, you'll need to make the appropriate modifications to them.

### Write the script that makes the extension

In the source project, you'll write a script that builds the extension. This script, the Make script, retrieves the objects you want to deliver from the source project and writes those objects out to the extension file (.avx).

2. Write SelectedFeaturesExt.Make and compile it. You'll run it later in this chapter.

```
'SelectedFeaturesExt.Make - Makes the extension object
SelectedFeaturesExt = Extension.Make(
    "$USEREXT/SelectedFeatures.avx".AsFileName,
    "Selected Features",
    av.FindScript("SelectedFeaturesExt.Install"),
    av.FindScript("SelectedFeaturesExt.Uninstall"),
    {"$AVBIN/avdlog.dll".AsFileName})

SelectedFeaturesExt.SetExtVersion(1)
SelectedFeaturesExt.SetAbout("Selected Features dialog (v1)")
```

```

SelectedFeaturesExt.SetUnloadScript(av.FindScript("SelectedFeaturesExt.Unload"))
SelectedFeaturesExt.SetCanUnloadScript(av.FindScript("SelectedFeaturesExt.CanUnload"))

'Retrieve Selected Features dialog from project, remove server and VTab
'object reference, and add it to extension
aDialog = av.FindDialog("SelectedFeatures")
aDialog.SetServer(nil)
aDialog.FindByName("lbx_Records").Empty
aDialog.FindByName("cbx_Themes").Empty
SelectedFeaturesExt.Add(aDialog)

'Retrieve View button that launches dialog and add it to the extension
SelectedFeaturesExt.Add(av.FindGUI("View").GetButtonBar.GetControls.Get(1))

'Retrieve all scripts for the dialog, make sure they are compiled, then
'add them to the extension. Scripts prefix by "SelectedFeatures"
for each aDoc in av.GetProject.GetDocs
  if (aDoc.Is(SEd)) then
    if (aDoc.GetName.Contains("SelectedFeatures.")) then
      if (aDoc.IsCompiled.Not) then
        aDoc.Compile
      end
      SelectedFeaturesExt.Add(av.FindScript(aDoc.GetName))
    end
  end
end

'Write the extension to file
SelectedFeaturesExt.Commit

```

This script first makes the extension object (SelectedFeaturesExt) using the Extension.Make request. The parameters for this request are

---

Parameter	Setting
Extension's file name	SelectedFeatures.avx
Extension name	Selected Features
Install script name	SelectedFeaturesExt.Install
Uninstall script name	SelectedFeaturesExt.Uninstall
Dependencies list	{"\$AVBIN/avdlog.dll".AsFileName}

---

Then the Make script sets some properties for the extension such as the version number, about string, Unload script name, and CanUnload script name. The script then adds the dialog, the button, and the necessary scripts to the extension object and commits them to the extension file.

If you use your extension's source project to test your dialogs, you need to remove any unnecessary object references to prevent those objects from being written to

your extension file. For instance, in this example the Selected Features dialog sets up a server VTab and also defines the list box contents from the VTab and the combo box contents from a list of themes. Running the dialog will establish the object references. Thus, if you then write the dialog to your extension, these referenced objects would be written to the extension as well. Unless you're distributing the additional objects with the extension, you should remove references to them before you write the dialog to the extension. The following code fragment from the Make script sets the dialog's server to nil and empties the list and combo box:

```
aDialog = av.FindDialog("SelectedFeatures")
aDialog.SetServer(nil)
aDialog.FindByName("lbox_Records").Empty
aDialog.FindByName("cbx_Themes").Empty
```

At this point, don't run this Make script yet. You'll run it after you've written all the other scripts the extension needs—the Install, CanUnload, Uninstall, and Unload scripts. In general, it's a good idea to name all scripts associated with an extension with a common prefix. In this example, the scripts for the extension are prefixed with "SelectedFeaturesExt."

### What does your extension need in order to work?

An extension can be dependent on other extensions or certain ESRI shared libraries (dlls). When ArcView loads your extension it makes sure that everything the extension depends upon is already loaded. All extensions that contain dialogs must be dependent upon either the ESRI dialog library, avdlog.dll or the Dialog Designer extension, dialog.avx. If you want the users of your extension to have access to the dialog editor in addition to your dialogs, use \$AVEXT/dialog.avx in the dependencies list. If your extension delivers complete dialogs that users will not edit, use \$AVBIN/avdlog.dll in the dependencies list.

### Write the scripts to load and install the extension

As you create your extension, you need to consider how your customizations will get added to ArcView when a user loads it. When you open the Extensions dialog and check an extension, two scripts run—the Load script and the Install script.

The Load script executes once when an extension file loads into ArcView. Use this script to establish the environment your extension requires such as connecting to a database or setting some object tags. Nothing special needs to be done to load an extension containing dialogs. The Load script is optional and is not used in this example.

The Install script executes when you first load the extension and every time you open a project or create a new project during that ArcView session. The Install script defines how the extension objects are installed into the current project.



3. Write `SelectedFeaturesExt.Install` and compile it. This `Install` script adds the button to the View button bar to launch the dialog.

```
'SelectedFeaturesExt.Install
'Install user interface components if a project is open
if (av.GetProject = nil) then
    return nil
end

'Retrieve the button from the extension (self) and add it after the
'first button in the View button bar
myButton = self.Get(1)
viewGUI = av.GetProject.FindGUI("View")
viewGUI.GetButtonBar.Add(myButton, 0)
viewGUI.SetModified(TRUE)
```

The `Install` script adds extension objects to the current project; thus, if there isn't a current project, the script ends. If there is, the `Install` script adds the button to the View's `DocGUI`. The `Install` script retrieves objects from the extension in the same order that you added them. Recall that in the `Make` script, the button is the second object added to the extension; thus, you retrieve it using `self.Get(1)`, where the `self` object is the extension.

The `Install` script doesn't need to install any dialogs or scripts referenced by the extension into the current project. ArcView will automatically search through any loaded extensions to find dialogs and scripts.

### Write the script that determines whether the extension can be unloaded

4. Write `SelectedFeaturesExt.CanUnload` and compile it. The `CanUnload` script checks to see if your extension can be unloaded.

```
'SelectedFeaturesExt.CanUnload
return (System.CanUnloadLibrary(self.GetDependencies.Get(0)))
```

This script checks whether the dialog library, `avdlog.dll`, can be unloaded. From the `Make` script you know that `avdlog.dll` is the first item in the dependencies list of the extension (`self`). You can use `self.GetDependencies.Get(0)` to reference that library when you check to see if it can be unloaded.

In the extensions you create in the future, the `CanUnload` script is a place where you can query some state and determine whether it's OK to unload your extension. For instance, the `CanUnload` script for the Dialog Designer extension checks whether the current project has any dialogs in it. If it does, the `CanUnload` script prevents you from unloading the Dialog Designer extension because the dialogs in the project depend upon the functionality the Dialog Designer provides.

## Write the scripts to unload and uninstall the extension

To unload your extension, you must remove any objects from the current project that the extension installed. In this example, this means removing the button added to the View's DocGUI. When you unload an extension, two scripts run—the Uninstall script and the Unload script.

5. Write SelectedFeaturesExt.Uninstall and compile it. This Uninstall script removes extension objects from the project when you unload the extension. All objects that you add to the project in the Install script must be removed from the project in the Uninstall script.

```
'SelectedFeaturesExt.UnInstall
if (av.GetProject = nil) then
    return nil
end

if (av.GetProject.IsClosing) then
    return nil
end

'Remove the button from the button bar
theButtonBar = av.GetProject.FindGUI("View").GetButtonBar
theButtonBar.Remove(self.Get(1))
```

This script removes the new button from the View button bar. Again, you can use the extension (self) to directly reference the objects to remove from the current project. The Uninstall script is also the place where you remove any objects from the project that are created and depend upon the extension.

6. Write SelectedFeaturesExt.Unload and compile it. Unload scripts perform any final cleanup your extension requires.

```
'SelectedFeaturesExt.Unload
Dialog.DetachFromExtension(self)
```

Any extension containing dialogs must have the Dialog.DetachFromExtension request in its Unload script. This request allows the extension to unload properly. Your Unload scripts may contain additional statements required to unload your extension.

## Create the extension

7. Make sure the Install, Uninstall, CanUnload, and Unload scripts are compiled.
8. Run the Make script.
9. Save your source project and close it.

## Test the extension

10. Open a new project and load the Selected Features extension using the Extensions dialog. Test your dialog to make sure it works correctly.

If everything works correctly then you're done. You've successfully created an extension containing dialogs. If there are any problems, go back into your source project and fix them and run the Make script again.

## Other considerations

You've just learned how to create a simple extension. If your extension creates and uses object tags or delivers customized document user interfaces (DocGUIs), there are a few extra things you need to consider before distributing the extension.

### Do your dialogs or other objects in your extension use object tags?

You should avoid setting object tags on objects in your source project as they may get written out to the extension. For example, if a script in your source project sets an object tag and you test that script, ArcView will create the object tag. If that object tag is attached to an object that gets incorporated into your extension (e.g., a dialog or menu item), it can cause problems in projects that load your extension. To avoid this problem, you should remove all object tags from your extension before you distribute it. Add the following lines of code to the end of your extension's Make script after the Commit statement.

```
myExt.Commit

RFileName = FileName.Make(" $USEREXT/myext.avx")
WFileName = FileName.Make(" $USEREXT/myext.tmp")
RFile = LineFile.Make(RFileName,#FILE_PERM_READ)
WFile = LineFile.Make(WFileName,#File_PERM_WRITE)
WFile.SetScratch(TRUE)
while (RFile.IsAtEnd.Not)
  buf = RFile.ReadElt
  if (buf.Contains("ObjectTag:").not) then
    WFile.WriteElt(buf)
  end
end
RFile.Close
WFile.Flush
File.Copy(WFilename, RFileName)
WFile.Close
```

This modification opens the avx file as a LineFile, creates a temporary file, and writes every line except ObjectTag lines in the avx file to the temporary file. Then the temporary file is copied to the avx file name.

## Does your extension deliver a customized DocGUI?

If your extension delivers a customized DocGUI, it may contain extraneous information associated with the DocGUI. For instance, the Window menu of your DocGUI contains menu options to activate all documents that are currently open in the project. Also, the separator in this menu references an object that contains a list of the open windows. These menu options and the object tag are written to the extension with the DocGUI. This will cause problems in projects that load the extension.

If your customized DocGUI is a View or Layout, you may need to remove a few controls that the Dialog Designer installs into these DocGUIs. These controls are the “Show/Hide Control Tools” on the Windows menu and the “Design/Run Controls” on the Graphics menu. If your extension will not be dependent upon the Dialog Designer extension (dialog.avx), you should remove these menu options from these GUIs before you add them to your extension.

To remove the object tag and menu controls, add the following lines of code to your Make script where you add the View or Layout GUI.

```
'Get the customized View DocGUI from the project
viewGUI = av.getproject.FindGUI("View")
viewMenuBar = viewGUI.GetMenuBar

'Find separator that contains the object tag and set the object tag to nil
winUpdate = viewMenuBar.FindByScript("WindowMenuUpdate")
winUpdate.SetObjectTag(nil)

>Delete all the activate window menu options
cSet = winUpdate.GetControlSet
cList = cSet.GetControls
last = cList.Get(cList.Count - 1)
while (last <> winUpdate)
  cSet.Remove(last)
  last = cList.Get(cList.Count - 1)
end

'Find the controls that were added by Dialog Designer and remove them
showControlTools = viewMenuBar.FindByScript("GraphicControl.ShowHideTools")
designControls = viewMenuBar.FindByScript("GraphicControl.DesignRun")
showControlTools.GetControlSet.Remove(showControlTools)
designControls.GetControlset.Remove(designControls)

'Then add your DocGUI to your extension
myExt.Add(viewGUI)
```

## Distributing your extension

Once you’ve built your extension, you’ll want to distribute it. In order to run an extension with dialogs, ArcView requires your extension file (.avx) and two additional files: avdlog.dll and avdlog.dat. These two files are installed with the Dialog Designer

and are located in the *bin* and *lib* directories of the ArcView install directory (*bin16* and *lib16* on Windows® 3.1; *bin32* and *lib32* on Windows 95® and Windows NT®). In general, you should distribute these files with your extension and not expect an end user to install the Dialog Designer on their system.

Where you install these files depends partly upon how you've built your extension. You specify the location of *avdlog.dll* in the extension's Make script (typically located in the *bin* directory). *Avdlog.dat* must be installed into the *lib* directory. Your extension file (*.avx*) can be installed in the ArcView *ext* directory (*ext16* or *ext32* on Windows) or in the directory specified with the *USEREXT* environment variable.

---

**Note** Deploying an applicaiton under Windows 3.1 requires one additional DLL, *ndwinx.dll*. This file should be installed in the *bin16* directory.

---

## Delivering dialogs in a project

If you've designed a dialog for a specific project and this is the only project that you'll ever use it in, then it isn't necessary to create an extension. You can load the Dialog Designer into this project and create the dialog. The project would remain dependent on the Dialog Designer extension. Anyone who can access the Dialog Designer extension can load the project and edit the dialogs.

Suppose you want to distribute a project with dialogs, yet not let anyone see or edit the dialog editor documents in the project window. You can create a new project based on the original source project that contains your dialogs but without the associated dialog editor documents. This new project is not dependent on *dialog.avx*, but instead on a special extension, *dlogcore.\_\_\_\_* (located in *\$AVEXT*), that loads only the dialog library (*avdlog.dll*) into the project. Thus, you'll be able to display dialogs, yet you won't see the dialog editor icon in the project window.

### To create a project without dialog editors

1. Make sure all the dialogs in your development project are compiled.
2. From the project's File menu, choose Save Detached.
3. Choose Yes to save your current development project. All changes that you've made to this project during this session will be saved.
4. Specify a name for the new detached project. Your current development project will be saved to the new project and the dialog editor icon will be removed from the project window.

---

**Note** Once you've detached a project, keep the source project if you need to edit the dialogs in the future. You can't interactively edit the dialogs in the detached project.

---

## Creating a personal working environment

If you need to access your own dialogs every time you start ArcView, you could create a user default project in your working directory.

### To create user default project with your dialogs

1. Open a project and load the Dialog Designer.
2. Create your dialogs and make any other changes to the interface you want.
3. From the project's File menu, choose Save Detached to detach the project from the Dialog Designer.
4. Open your new detached project and choose Make Default from the Customize Dialog box.

ArcView creates a file in your home directory called `default.apr`. Now, whenever you start ArcView, all your customizations, including your dialogs, will be available.

## Creating a system default environment

If everyone in your organization will be using your dialogs in all their work, you could create a new system default environment. This is an easy way to deliver your dialogs without having to learn how to create an extension.

### To create user default project with your dialogs

1. Open a project and load the Dialog Designer.
2. Create your dialogs and make any other changes to the interface you want.
3. From the project's File menu, choose Save Detached to detach the project from the Dialog Designer.
4. In your new detached project, create and run a new script containing the following line of code:

```
av.GetProject.MakeSysDefault("$HOME/asysdef.apr".AsFileName, False)
```

Prefix this script's name with a dot (`.`), for example, `.MyScript`. This script creates a system default file in your home directory called `asysdef.apr`.

5. Copy this file into ArcView's *etc* directory and name it `default.apr`.

Now, whenever anyone starts ArcView, all your customizations, including your dialogs, will be available.

## APPENDIX A

# Control properties descriptions

Every control on the dialog, and the dialog itself, is defined by a set of properties that govern its behavior. For example, you set a property to define the text label of a check box, to set the upper and lower bounds of a slider, and to set the script to run when you click a button. While designing your dialog, you set these properties through the Control Properties dialog. To change them when the dialog is running, you use the appropriate Avenue request in your scripts.

This appendix lists the properties for each control and contains a complete description of each one.

## Control properties

This section lists the control properties for each control and for the dialog. See the next section for a more thorough discussion of each property.

### **Button**

Use a button to begin, end, or interrupt a process. Buttons are typically represented by an icon that indicates what it does. If you need to display text, use a label button instead.

*Click* - the name of the script to execute when you click the button.

*Disabled* - enables or disables the button.

*Help* - the status bar help string and tool tip.

*HelpTopic* - does not apply to buttons, but listed to maintain compatibility with ArcView GIS Version 3.0a. Displays a help topic when set as dialog property; see Dialog below.

*Icon* - the name of the icon to display.

*Invisible* - specifies if the button is visible.

*Tag* - stores a text string with the control.

*Update* - the name of the script to execute when an Update event occurs.

### **Check box**

Use a check box to provide a true/false or yes/no option on your dialog. When selected, the check box will appear checked.

*Click* - the name of the script to execute when you click the check box.

*Disabled* - enables or disables the check box.

*Help* - the status bar help string and tool tip.

*Invisible* - specifies if the check box is visible.

*Label* - the visible text string that annotates the check box.

*Selected* - sets the initial state of the check box when you first display the dialog.

*Tag* - stores a text string with the control.

*Update* - the name of the script to execute when an Update event occurs.



 **Combo box**

Use a combo box to provide a list of choices on your dialog when there is a minimum of space. The list of available choices drops down allowing you to select a choice.

*Disabled* - enables or disables the combo box.

*Help* - the status bar help string and tool tip.

*Invisible* - specifies if the combo box is visible.


*Label* - the visible text string that annotates the combo box.

*NextControl* - the control that will receive keyboard focus when the tab key is pressed.

*Select* - the name of the script to execute when a selection is made.

*Tag* - stores a text string with the control.

*Update* - the name of the script to execute when an Update event occurs.

 **Dialog (continued)**

**DefaultButton** - identifies a label button on the dialog that will execute its Click script when the return key is pressed.

**DocActivate** - the name of the script to execute whenever a document is activated.

**EscapeEnabled** - allows you to dismiss a dialog by pressing the escape key.

**HasTitleBar** - determines whether the dialog has a title bar.

**HelpTopic** - the help topic to display when the F1 key is pressed over the dialog.

**Modal** - makes a dialog the focus of input, preventing any interaction with ArcView.

**Open** - the name of the script to run when the dialog is opened.

**Resizable** - specifies if a dialog can be resized once displayed.

**ServerActivated** - the name of the script to run when the dialog's server document becomes active.

**ServerClosed** - the name of the script to run when the dialog's server document closes.

**ServerDeactivated** - the name of the script to run when the dialog's server document deactivates.

**ServerDefinitionChanged** - the name of the script to run when the definition of the dialog's server VTab changes.

**ServerOpened** - the name of the script to run when the dialog's server document opens.

**ServerRecordsAdded** - the name of the script to run when records are added to the dialog's server VTab.

**ServerRecordsDeleted** - the name of the script to run when records are deleted from the dialog's server VTab.

**ServerSelectionChanged** - the name of the script to run when the selection changes in a dialog's server VTab.

**Title** - the text appearing in the dialog's title bar.

**Update** - the name of the script to execute when an Update event occurs.

 **Icon box**

Use an icon box to display an icon on the dialog. Create icons from: Windows bitmap (.bmp), GIF (.gif), TIFF (.tif), X bitmap (.xbm), Sun® Raster files (.rs), Claris® MacPaint™ (.mcp), and Neuron Data's Open Interface icon resources.

**Icon** - the name of the icon to display.

**Invisible** - specifies if the icon box is visible.

**Tag** - stores a text string with the control.

**Update** - the name of the script to execute when an Update event occurs.

 **Label button**

Use a label button to begin, end, or interrupt a process. Label buttons contain a text label that indicates what the label button does.

*Click* - the name of the script to execute when you click the label button.

*Disabled* - enables or disables the label button.

*Help* - the status bar help string and tool tip.

*Invisible* - specifies if the label button is visible.

*Label* - the visible text string that annotates the label button.

*Tag* - stores a text string with the control.

*Update* - the name of the script to execute when an Update event occurs.

 **List box**

Use a list box to present a list of choices to select from. A list box can contain many rows and columns and can be defined from a list object or a VTab.

*Apply* - the name of the script to run when a cell is double-clicked.

*Disabled* - enables or disables the list box.

*Help* - the status bar help string and tool tip.

*HorizontalScroll* - add a horizontal scroll bar to the list box.

*Invisible* - specifies if the list box is visible.

*NextControl* - the control that will receive keyboard focus when the tab key is pressed.

*Select* - the name of the script to execute when a selection is made.

*SelectionStyle* - defines whether you can select a single cell, multiple cells, a single row, or a single column from the list box.

*Tag* - stores a text string with the control.

*Update* - the name of the script to execute when an Update event occurs.

*VerticalScroll* - adds a vertical scroll bar to the list box.

 **Radio button**

Use a radio button to present a set of mutually exclusive options. You group a set of radio buttons by drawing them inside a control panel. Selecting one radio button deselects the other radio buttons on the same panel.

*Click* - the name of the script to execute when you click the radio button.

*Disabled* - enables or disables the radio button.

 **Radio button (continued)**

*Help* - the status bar help string and tool tip.

*Invisible* - specifies if the radio button is visible.

*Label* - the visible text string that annotates the radio button.

*Tag* - stores a text string with the control.

*Update* - the name of the script to execute when an Update event occurs.

 **Slider**

Use a slider to set a value from a discrete range of continuous values.

*AuxIncrement* - adds tick marks along either the top or left side of the slider, depending upon its orientation.

*Click* - the name of the script to execute when you click the slider.

*Disabled* - enables or disables the slider.

*Drag* - the name of the script to execute while you drag the slider handle.

*Help* - the status bar help string and tool tip.

*Horizontal* - creates a horizontal or vertical slider.

*Invisible* - specifies if the slider is visible.

*Lower* - sets the lower bound of the slider's range.

*MainIncrement* - adds tick marks along either the bottom or right side of the slider, depending upon its orientation.

*NextControl* - the control that will receive keyboard focus when the tab key is pressed.

*ReadOnly* - creates a read-only slider.

*StepButtons* - adds or removes buttons from the slider that increment the slider's value.

*Tag* - stores a text string with the control.

*Update* - the name of the script to execute when an Update event occurs.

*Upper* - sets the upper bound of the slider's range.

*Value* - sets the initial value for the slider.

*ValueIncrement* - restricts the values a slider can assume (e.g., only odd numbers).

 **Text box**

Use a text box when you need to display or edit multiple lines of text. If you only need a single line, use a text line instead.

*Changed* - the name of the script to execute whenever the text changes.

*Click* - the name of the script to execute when you click the text box.

*Disabled* - enables or disables the text box.

*Empty* - the name of the script to execute when you enter text into an empty text box or clear all the text from the text box.

*Help* - the status bar help string and tool tip.

*HorizontalScroll* - adds a horizontal scroll bar to the text box.

*Invisible* - specifies if the text box is visible.

*Label* - the visible text string that annotates the text box.

*NextControl* - the control that will receive keyboard focus when the tab key is pressed.

*ReadOnly* - creates a read-only text box.

*Size* - sets the maximum number of characters that can be entered in the text box.

*Tag* - stores a text string with the control.

*Text* - sets the default text that appears in the text box.

*Update* - the name of the script to execute when an Update event occurs.

*VerticalScroll* - adds a vertical scroll bar to the text box.

### **Text label**

Use a text label to annotate your dialog. There is no user interaction with a text label; it simply displays static text.

*Disabled* - enables or disables the text label.

*Invisible* - specifies if the text label is visible.

*Label* - the text string of the text label.

*Tag* - stores a text string with the control.

*Update* - the name of the script to execute when an Update event occurs.

### **Text line**

Use a text line when you want to provide a single line for text input. If you need more than one line, use a text box instead.

*Apply* - the name of the script to execute whenever the return key is pressed or when the text line loses focus.

*Changed* - the name of the script to execute whenever the text changes.

*Click* - the name of the script to execute when you click the text line.

*Disabled* - enables or disables the text line.

## **Text line (continued)**

*Empty* - the name of the script to execute when you enter text into an empty text line or clear all the text from the text line.

*FocusLost* - the script to execute when the tab key is pressed or the mouse is moved.

*Help* - the status bar help string and tool tip.

*HiddenText* - replaces all characters with an asterisk (\*).

*Invisible* - specifies if the text line is visible.

*Label* - the visible text string that annotates the text line.

*LabelSize* - sets the width of the label.

*NextControl* - the control that will receive keyboard focus when the tab key is pressed.

*ReadOnly* - creates a read-only text line.

*Size* - sets the maximum number of characters that can be entered in the text line.

*Tag* - stores a text string with the control.

*Text* - sets the default text that appears in the text line.

*TextType* - defines the types of characters that a text line can accept: alphanumeric characters, integers, or real numbers.

*Update* - the name of the script to execute when an Update event occurs.

## **Tool**

Use a tool when you need to interact with a document. A tool generally stays depressed when clicked, but usually nothing happens until you click somewhere in the active document. For instance, you'd use a tool if you wanted a user to specify an area on the display to zoom in to.

*Apply* - the name of the script to execute when you click on a document such as a view.

*Click* - the name of the script to execute when you click the tool.

*Cursor* - identifies the cursor to use.

*Disabled* - enables or disables the tool.

*Help* - the status bar help string and tool tip.

*HelpTopic* - does not apply to tools, but listed to maintain compatibility with ArcView GIS Version 3.0a. Displays a help topic when set as dialog property; see Dialog above.

*Icon* - the name of the icon to display.

*Invisible* - specifies if the tool is visible.

*Tag* - stores a text string with the control.

*Update* - the name of the script to execute when an Update event occurs.

## Property descriptions

This section contains an alphabetical listing of all properties.

### **Activate**

The name of a script attached to the dialog's activate event. This script is executed when the dialog becomes the active dialog (for example, when the mouse enters the window of a modeless dialog).

*Applies to:* Dialog

*Avenue Requests:* SetActivate, GetActivate

### **AlwaysOnTop**

Controls whether or not the dialog will stay on top of the ArcView application window. When true, the dialog will always stay on top of ArcView's application window. Setting this property will not keep the dialog on top of the windows of other applications running on your system. This property is not supported on the Macintosh and some UNIX workstations.

*Applies to:* Dialog

*Avenue Requests:* SetAlwaysOnTop, IsAlwaysOnTop

### **Apply**

Specifies the name of the script to execute when an Apply event occurs. A tool, text line, and list box control each have different implementations of the Apply event.

An Apply event on a tool is triggered by clicking the mouse on a document such as a view.

An Apply event on a text line occurs when you press the return key while the text line has the keyboard focus and the dialog doesn't have a default button set. This event is not triggered by the text line losing focus such as when you press the tab key or use the mouse to move to another control; see the FocusLost property below. While you can use the Apply script to validate the text entered into the text line, it may be better to place the validation on the button that executes the dialog such as an OK button.

An Apply event on a list box occurs when the user double-clicks a cell. Double-clicking the cell runs the Select script (Select property) first, then the script specified with the Apply property. The Apply event typically performs some action on the selection. For example, double-clicking a cell might delete its contents.

With a list box, you can also run scripts when the user

- Clicks a cell; set the Select property or use the SetSelect request.
- Selects a specific cell or group of cells; use the SetRangeSelect request.
- Double-clicks a specific cell or group of cells; use the SetRangeApply request.

*Applies to:* List box, Text line, Tool

*Avenue Requests:* SetApply, GetApply

## AuxIncrement Property

Adds tick marks along either the top or left side of the slider, depending upon the slider orientation. The default value for this property is 0, or no tick marks. The value you specify represents the distance between the tick marks in the units of the slider and should be appropriate for the range of the slider. For example, a value of 1 on a slider with a range from 0 to 10 would create a tick mark every one unit along the slider. However, a value of 1 is not appropriate where the range of the slider is from 0 to 1000.

This property has no effect on the slider's current value and is only used to enhance the display of the slider. Some operating systems don't support tick marks on sliders.

*Applies to:* Slider

*Avenue Requests:* GetAuxIncrement, SetAuxIncrement

## Changed Property

Specifies the name of the script to execute when you change the text in the text line or text box. This script runs after every keystroke. This event is not triggered when text is cut from or pasted into the text line.

*Applies to:* Text box, Text line

*Avenue Requests:* GetChanged, SetChanged



## Click Property

Specifies the name of the script to execute when you click on a control.

*Applies to:* Button, Check box, Label button, Radio button, Slider, Text box, Text line, Tool

*Avenue Requests:* GetClick, SetClick

## Close Property

The name of the script to execute when the dialog is closed.

*Applies to:* Dialog

*Avenue Requests:* GetClose, SetClose

## Closeable Property

Indicates whether the dialog can be closed from the standard window controls on the dialog's frame. Note: Different operating systems implement different window frame controls. Thus, even though this property is set to false, the dialog may still be closeable under some operating systems.

*Applies to:* Dialog

*Avenue Requests:* IsCloseable, SetCloseable

## Cursor Property

Identifies the cursor associated with a tool. Double-click the Cursor property to display the Cursor Manager, a dialog box that contains a list of available cursors. The cursor you select appears when you use the tool on the active document.

*Applies to:* Tool

## DefaultButton Property

Identifies a label button inside a dialog that will have its click script executed when the user presses the return key. The default button only works when a text line, slider, list box, or combo box control currently has the keyboard focus in the running dialog. Thus, if a check box currently has the focus in the dialog, it will be checked or unchecked

when the user presses return. Similarly, if another label button has the keyboard focus, its click script will execute, not the default label button's click script.

If, at run time, the default button is disabled, its click script will not execute. Setting a default button will override the Apply property on a text line.

Use this property, for instance, on dialogs that contain a series of text line controls where the user enters several pieces of information, all of which are validated by pressing return to execute the click script of an OK button.

*Applies to:* Dialog

*Avenue Requests:* GetDefault, SetDefault

## Disabled Property

Enables or disables the control. You disable a control when it's inappropriate to use it. Typically you'll enable or disable a control through the control's Update script. The Update script determines whether or not the control should be enabled based on the state of other controls on the dialog or on the state of ArcView. For example, you might include a line of code like this:

```
self.SetEnabled(some condition)
```

The self object in a control's Update script is the control itself.

Disabling a control panel will disable all the controls on that panel. The dialog itself is a control panel; thus, to disable all the controls on a dialog, you could do this:

```
aDialog.GetControlPanel.SetEnabled(false)
```

*Applies to:* Button, Check box, Combo box, Control panel, Label button, List box, Radio button, Slider, Text box, Text label, Text line, Tool

*Avenue Requests:* IsEnabled, SetEnabled

## DocActivate Property

The name of a script to execute whenever an ArcView document is activated. This lets the dialog know about any document that becomes active. For example, if you want to disable the controls on a modeless dialog when the active document is not a view or layout, include this code in your DocActivate script:

```
'Dim control when active doc is not view or layout  
theDoc = self.GetActiveDoc  
enabled = theDoc.Is(View) or theDoc.Is(Layout)  
'the self object is the dialog  
self.GetControlPanel.SetEnabled(enabled)
```

ArcView sends a DocActivate event to all open dialogs that have a DocActivate property set whenever a document becomes active.

The DocActivate script is also where you might include code to update individual controls on your dialog. For example, suppose your dialog lists the active themes in a view. As different views become active, you'd want to update the list of themes to reflect those in the current view.

```
theDoc = self.GetActiveDoc
aListBox = self.FindByName("aListBox1")
if (theDoc.Is(View) and theDoc.GetThemes.IsEmpty.Not) then
    theThemes = theDoc.GetThemes
    aListBox.DefineFromList(theThemes)
else
    aListBox.DefineFromList({})
end
```

If, for some reason, the DocActivate script contains an error (e.g., syntax error), the dialog will no longer respond to the DocActivate event (and thus, the DocActivate script will not execute). After correcting the problem with the script, you must reset the DocActivate property on the Control Properties dialog or use the ResetDocActivate request on the Dialog class.

When finding the active document in a DocActivate script or any script called from it, use the GetActiveDoc request on the Dialog object (the self object in the examples above), not the application object, av.

*Applies to:* Dialog

*Avenue Requests:* GetDocActivate, ResetDocActivate, SetDocActivate

## Drag Property

The name of the script that will execute repeatedly while you drag the slider handle.

*Applies to:* Slider

*Avenue Requests:* GetDrag, SetDrag

## Empty Property

Specifies the name of the script to execute whenever you enter text into an empty text line or text box, or when you clear out all of the text in a text line or text box.

*Applies to:* Text box, Text line

*Avenue Requests:* GetEmpty, SetEmpty

## EscapeEnabled Property

Allows you to dismiss a dialog by pressing the escape key. Pressing escape sends the Close request to the dialog. This property is a design time property that prevents you from creating a modal dialog that can't be dismissed for some reason. When you've finished editing your dialog and are ready to incorporate it into your application, you may want to set this property to false so that a user of your dialog can't dismiss it by pressing escape.

*Applies to:* Dialog

*Avenue Requests:* IsEscapeEnabled, SetEscapeEnabled

## FieldNamesVisible Property

Specifies whether or not field names will appear as the first row of the list box when the list box is defined from a VTab. If the field has an alias, it will be displayed.

*Applies to:* List box

*Avenue Requests:* GetFieldNamesVisible, SetFieldNamesVisible

## FocusLost Property

Specifies the name of the script to execute when the text line loses focus. This occurs when you press the tab key or use the mouse to move to another control.

*Applies to:* Text line

*Avenue Requests:* GetFocusLost, SetFocusLost

## HasTitleBar Property

Determines whether or not the dialog frame will have a title bar. To set the text that appears in a title bar, use the Title property.

*Applies to:* Dialog

*Avenue Requests:* GetTitle, HasTitleBar, SetTitle, UseTitleBar

## Help Property

Specifies the help string for the control when you move the mouse over it. This help string has two components: a string that appears in ArcView's status bar and a tool tip that appears directly over the control under the Windows 95 and Windows NT operating systems. The syntax for the help string is

```
tool tip//status bar help
```

Typically the tool tip is one or two words that describe the control, whereas the status bar help string is a longer description of the control. If you just want a tool tip, enter the string as

```
tool tip//
```

Tool tips, however, do not appear over control panels. If you just want the status bar help string, enter the string without '//'.

**Applies to:** Button, Check box, Combo box, Control panel, Label button, List box, Radio button, Slider, Text box, Text label, Text line, Tool

**Avenue Requests:** GetHelp, SetHelp

## HelpTopic Property

Specifies the name of the help topic to display as context sensitive help for the dialog when the F1 key is pressed over the dialog. This property is only functional on the dialog itself. In order to maintain compatibility with ArcView GIS Version 3.0a, this property also appears on the Control Properties dialog for buttons and tools. However, it is not functional with buttons and tools on dialogs.

The value of this property is the name of the help topic, given as its topic ID (# footnote in the source help file), followed by @, then the name of the help file containing the topic. Use the following syntax:

```
<aTopicName>@<aHelpFileName>
```

for example, mytopic@myhelp.hlp. ArcView will search for the help file in the Help directory of ArcView's install directory. While you can optionally specify a pathname to the help file, the best method for integrating your help file with ArcView's help system is to place it in ArcView's Help directory. Then you can link your help file to ArcView's contents file (arcview.cnt).

For information on building help files, search the on-line help index for 'help files'.

**Applies to:** Dialog

**Avenue Requests:** GetHelpTopic, SetHelpTopic

## HiddenText Property

When set to true, this property replaces any characters typed into the text line with an asterisk (\*). For example, set this property when you want the text line to accept a password. The default value is false.

*Applies to:* Text line

*Avenue Requests:* HasHiddenText, SetHiddenText

## Horizontal Property

Orients the slider either horizontally or vertically. When true, the slider will be oriented horizontally. This is the default.

*Applies to:* Slider

*Avenue Requests:* IsHorizontal, SetHorizontal

## HorizontalScroll Property

Adds a horizontal scroll bar to a list box or text box.

If a text box doesn't have a horizontal scroll bar and the text is longer than the width of the text box, the text will wrap. With a horizontal scroll bar, the text will wrap when the new line (NL) character is encountered in the text string displayed in the text box.

If a list box doesn't have a horizontal scroll bar, its columns will be stretched or compressed to fit the width of the list box. To avoid this behavior set a column width for each column explicitly or add a horizontal scroll bar.

*Applies to:* List box, Text box

*Avenue Requests:* HasHorizontalScroll, SetHorizontalScroll, FitRows, SetColumnWidth

## Icon Property

Specifies the name of the icon associated with a button, a tool, or an icon box.

To set an icon, double-click the Icon property to display the Icon Manager, a dialog box that contains a list of available icons. From this dialog, you can load your own icon. Valid file formats for an icon include: Windows bitmap, GIF, TIFF, X bitmap, Sun® Raster files, Claris® MacPaint™, and Neuron Data's Open Interface icon resources. For information on loading your own icons into the Icon Manager, search the on-line Help for 'loading an icon'.

The icon you specify will be centered over the control. If the icon is larger than the control, it will get clipped by the control; ArcView does not scale icons. Screen resolution may also impact the visible display of the icon.

*Applies to:* Button, Icon box, Tool

*Avenue Requests:* GetIcon, SetIcon

## Invisible Property

Determines if the control is visible on the dialog. Use this property to hide or show controls on the dialog. This property can be set in a script with the SetVisible request. Making a control panel invisible will also hide all the controls on that panel. The dialog itself is a control panel; thus to hide all the controls on a dialog, you could do this:

```
aDialog.GetControlPanel.SetVisible(false)
```

*Applies to:* Button, Check box, Combo box, Control panel, Icon box, Label button, List box, Radio button, Slider, Text box, Text label, Text line, Tool

*Avenue Requests:* IsVisible, SetVisible

## Label Property

The visible text string that annotates the control. For example, on a label button, this is the text on the button; on a check box, this is the text next to the check box. If you don't want a label on your control, highlight the property in the Control Properties dialog and press the delete key.

If you want a multiline text label, here's a tip. Double-click the Label property to display the dialog that allows you to enter a text string. Select all the text and press the delete or backspace, then immediately press the return key. You can now use the arrow keys on the keyboard to enter text on two lines. If you need more than two lines of text for your label, use the keyboard shortcut to copy and paste the return character to get the number of lines you need. Then, use the arrow keys to enter in text. Another way to create a multiline label is with Avenue code. For example, you could include this in a dialog's Open script:

```
l = self.FindByName("aTextLabel1")  
l.SetLabel("First line"+NL+"Second line"+NL+"Third line")
```

The effect a multiline label has on a control will vary.

*Applies to:* Check box, Combo box, Control panel, Label button, Radio button, Text box, Text label, Text line

*Avenue Requests:* GetLabel, SetLabel

## LabelSize Property

Specifies the width in pixels that the label associated with a text line will occupy. When set to 0 (the default value), the length of the label determines the amount of space required for the label.

Typically, you'll set this property to when you want to align the type-in area of several text line controls, making the type-in area the same width. For example, add two text line controls and set the LabelSize for both to 100. This will make the width of both labels equivalent, regardless of the length of the actual text string. Common settings for this property will range from 25 to 500 pixels, depending upon the size of your dialog. The label will be truncated if the width you specify is not sufficient enough to display the entire text of the label.

*Applies to:* Text line

*Avenue Requests:* GetLabelSize, SetLabelSize

## Lower Property

Specifies the lower bound of the slider's range. The default value is 0. After adjusting the slider's range, make sure the ValueIncrement property is set appropriately for the range; if not, it can prevent the slider handle from moving.

*Applies to:* Slider

*Avenue Requests:* GetLower, SetLower

## MainIncrement Property

Adds tick marks along either the bottom or the right side of the slider, depending upon the slider orientation. The value you specify represents the distance between the tick marks in the units of the slider and should be appropriate for the range of the slider. For example, the default value of 2 on a slider with a range from 0 to 10 would create a tick mark every two units along the slider. However, a value of 2 is not appropriate where the range of the slider is from 0 to 1,000. Set the property to 0 to remove the tick marks.

This property has no effect on the slider's current value and is only used to enhance the display of the slider. Some operating systems don't support tick marks on sliders.

*Applies to:* Slider

*Avenue Requests:* GetMainIncrement, SetMainIncrement



## Modal Property

Makes the dialog the focus of input and prevents the user from working with any other part of ArcView (or any other dialogs) while the modal dialog is displayed. Typically, you use a modal dialog to solicit required information from a user before beginning some process or activity. In order to continue, the user must explicitly dismiss the dialog. Thus, your dialog must provide a method for dismissing itself, such as an OK or Cancel button with the appropriate Avenue code behind it.

While designing your dialog, you can dismiss it by pressing the escape key if you haven't yet implemented a method for dismissing it. When you've finished editing your dialog and are ready to incorporate it into your application, you may want to set the `EscapeEnabled` property to false so that a user of your dialog can't dismiss it by pressing escape.

*Applies to:* Dialog

*Avenue Requests:* `IsModal`, `SetModal`, `SetModalResult`

## NextControl Property

Specifies the control that will receive keyboard focus when the tab key is pressed. You might set this property when, for example, you have several text line controls on your dialog and want your user to be able to use the tab key to move between them in a specific order.

You can set the `NextControl` to one of the following controls: a text line, text box, list box, combo box, or slider. Setting the `NextControl` of a control to itself stops the tab key from moving when that control is encountered.

*Applies to:* Combo box, List box, Slider, Text box, Text line

*Avenue Requests:* `GetNextControl`, `SetNextControl`

## Open Property

Specifies the name of the script to execute when the dialog is opened.

This script will typically establish starting values for controls, broadcaster-listener relationships between controls to handle Update events, and a client/server relationship between a dialog and a document. For example:

```
aTextLine = self.FindByName("aTextLine1")
'initialize a control
aTextLine.SetText( " ")
```

```
'set up listeners for update events
aButton1 = self.FindByName("aLabelButton1")
aButton2 = self.FindByName("aLabelButton2")
aButton3 = self.FindByName("aLabelButton3")
aTextLine.SetListeners({aButton1, aButton2, aButton3})
'set up a server document
self.SetServer(av.GetProject.FindDoc("view1"))
```

The self object of the Open script is the dialog.

*Applies to:* Dialog

*Avenue Requests:* GetOpen, SetOpen

## ReadOnly Property

Creates a read-only slider, text box, or text line control that can't be modified by dialog interaction. The default value of this property is false. Set the value to true to create a read-only control. The information displayed by the control can still be modified in your scripts using the appropriate Avenue requests.

*Applies to:* Slider, Text line, Text box

*Avenue Requests:* IsReadOnly, SetReadOnly

## Resizable Property

Controls whether a dialog can be resized once displayed.

If you want to prevent a user from changing the size of your dialog, set this property to false. You might consider making a dialog resizable when, for example, you have a list box on the dialog and want to show more of the list when the dialog is enlarged. In addition to making the dialog resizable, you'll also need to specify how the list box should respond when the dialog is enlarged. You do this through the Control Fasteners dialog, available on the Control menu of the dialog editor.

*Applies to:* Dialog

*Avenue Requests:* IsResizable, SetResizable

## Select Property

Specifies the name of the script to execute when a user interactively selects a row in a combo box or when a user interactively selects or deselects a cell in a list box. Select scripts typically inform other controls, or other parts of ArcView, that the selection has

changed. For example, making a selection might cause a button to activate. This script does not run when you programmatically select a row through another script associated with the dialog.

With a list box, you can also run scripts when the user

- Double-clicks a cell; set the Apply property or use the SetApply request.
- Selects a specific cell or group of cells; use the SetRangeSelect request.
- Double-clicks a specific cell or group of cells; use the SetRangeApply request.

*Applies to:* Combo box, List box

*Avenue Requests:* GetSelect, SetSelect

## Selected Property

Sets the initial state of the check box when you first display the dialog. Once you change the selected state of the check box, it will retain that state when you display the dialog again. If you want the check box to initialize to a particular selected state each time you display the dialog, you'll need to add a line of code such as this to the dialog's Open script:

```
self.FindByName("aCheckBox1").SetSelected(true)
```

*Applies to:* Check box

*Avenue Requests:* IsSelected, SetSelected

## SelectionMode Property

Defines how you can make selections from the list box. You can define the list box so that you can select

- A single cell.
- A single row.
- A single column.
- A single range of cells.
- Multiple ranges of cells.

Making a selection in the list box will clear any previous selection, except with the multiple range option. With this option, holding down the shift key will add to the selection.

You can run scripts in response to selections made in the list box. To run a script when the user

- Selects any cell, set the Select property or use the SetSelect request.
- Double-clicks a cell, set the Apply property or use the SetApply request.
- Selects a specific cell or group of cells, use the SetRangeSelect request.
- Double-clicks a specific cell or group of cells, use the SetRangeApply request.

*Applies to:* List box

*Avenue Requests:* GetSelectionStyle, SetSelectionStyle

## ServerActivated Property

The name of the script to execute when the dialog's server document becomes active. For example, suppose you wanted to enable the controls on the dialog when its server document became active. You could include this code in the ServerActivated script:

```
self.GetControlPanel.SetEnabled(true)
```

where the self object is the dialog. This property does not apply when the server is a VTab.

This script runs whenever the server document becomes active, regardless of whether the dialog is open or closed.

A dialog can establish a client/server relationship with a particular document or VTab. You typically establish the server document in the dialog's Open or DocActivate scripts using the SetServer request:

```
self.SetServer(av.GetProject.FindDoc("view1"))
```

However, if an individual control needs to change the server, the SetServer request may instead be part of a script attached to that control.

*Applies to:* Dialog

*Avenue Requests:* GetServerActivated, SetServerActivated, SetServer

## ServerClosed Property

The name of the script to execute when the dialog's server document window is closed. For example, if you want to close the dialog when the server document is closed. The ServerClosed script would contain

```
self.Close
```

where the self object is the dialog. This property does not apply when the server is a VTab. This script runs whenever the server document is closed, regardless of whether the dialog is open or closed.

A dialog can establish a client/server relationship with a particular document or VTab. You typically establish the server document in the dialog's Open or DocActivate scripts using the SetServer request:

```
self.SetServer(av.GetProject.FindDoc("view1"))
```

However, if an individual control needs to change the server, the SetServer request may instead be part of a script attached to that control.

*Applies to:* Dialog

*Avenue Requests:* GetServerClosed, SetServerClosed, SetServer

## ServerDeactivated Property

The name of the script to execute when the dialog's server document becomes inactive. For example, suppose you wanted to disable the controls on the dialog when its server document became inactive. You could include this code in the ServerDeactivated script:

```
self.GetControlPanel.SetEnabled(false)
```

where the self object is the dialog. This property does not apply when the server is a VTab. This script runs whenever the server document is deactivated, regardless of whether the dialog is open or closed.

A dialog can establish a client/server relationship with a particular document or VTab. You typically establish the server document in the dialog's Open or DocActivate scripts using the SetServer request:

```
self.SetServer(av.GetProject.FindDoc("view1"))
```

However, if an individual control needs to change the server, the SetServer request may instead be part of a script attached to that control.

*Applies to:* Dialog

*Avenue Requests:* GetServerDeactivated, SetServerDeactivated, SetServer

## ServerDefinitionChanged Property

The name of the script to execute when the definition of the dialog's server VTab is changed. This script runs whenever the definition changes, regardless of whether the dialog is open or closed.

A dialog can establish a client/server relationship with a particular document or VTab. You typically establish the server document in the dialog's Open or DocActivate scripts using the SetServer request:

```
self.SetServer(av.GetProject.FindDoc("Table1").GetVTab)
```

However, if an individual control needs to change the server, the SetServer request may instead be part of a script attached to that control.

*Applies to:* Dialog

*Avenue Requests:* GetServerDefinitionChanged, SetServerDefinitionChanged, SetServer

## ServerOpened Property

The name of the script to execute when the dialog's server document window is opened. For example, if you want to open the dialog when the server document is opened, you could include this code in the ServerOpened script:

```
self.Open
```

where the self object is the dialog. This property doesn't apply if the server is a VTab. This script runs whenever the server document opens, regardless of whether the dialog is open or closed.

A dialog can establish a client/server relationship with a particular document or VTab. You typically establish the server document in the dialog's Open or DocActivate scripts using the SetServer request:

```
self.SetServer(av.GetProject.FindDoc("view1"))
```

However, if an individual control needs to change the server, the SetServer request may instead be part of a script attached to that control.

*Applies to:* Dialog

*Avenue Requests:* GetServerOpened, SetServerOpened, SetServer

## ServerRecordsAdded Property

The name of the script to execute when records are added to the dialog's server VTab or Table document. This script runs whenever records are added to the server document, regardless of whether the dialog is open or closed.

A dialog can establish a client/server relationship with a particular document or VTab. You typically establish the server document in the dialog's Open or DocActivate scripts using the SetServer request:

```
self.SetServer(av.GetProject.FindDoc("Table1"))
```

However, if an individual control needs to change the server, the SetServer request may instead be part of a script attached to that control.

*Applies to:* Dialog

*Avenue Requests:* GetServerRecordsAdded, SetServerRecordsAdded, SetServer

## ServerRecordsDeleted Property

The name of the script to execute when records are deleted from the dialog's server VTab or Table document. This script runs whenever records get deleted from the server document, regardless of whether the dialog is open or closed.

A dialog can establish a client/server relationship with a particular document or VTab. You typically establish the server document in the dialog's Open or DocActivate scripts using the SetServer request:

```
self.SetServer(av.GetProject.FindDoc("Table1"))
```

However, if an individual control needs to change the server, the SetServer request may instead be part of a script attached to that control.

*Applies to:* Dialog

*Avenue Requests:* GetServerRecordsDeleted, SetServerRecordsDeleted, SetServer

## ServerSelectionChanged Property

The name of the script to execute whenever the selection changes in a dialog's server VTab (or FTab). For instance, this can occur when a user of the dialog:

- Selects a feature using theme selection tools
- Selects a record in a table

- Deletes a data element from a chart
- Performs theme on theme selection
- Interacts with the Query Builder

This script runs whenever the selection changes in the server document, regardless of whether the dialog is open or closed.

A dialog can establish a client/server relationship with a particular document or VTab. You typically establish the server document in the dialog's Open or DocActivate scripts using the SetServer request:

```
self.SetServer(av.GetProject.FindDoc("table1").GetVTab)
```

However, if an individual control needs to change the server, the SetServer request may instead be part of a script attached to that control.

*Applies to:* Dialog

*Avenue Requests:* GetServerSelectionChanged, SetServerSelectionChanged, SetServer

## Size Property

Specifies the length of the string the text line or text box can accept. The default value of 0 indicates that the control will accept an unlimited number of characters. Set this property when you need to restrict the number of characters that should be entered into the control.

*Applies to:* Text box, Text line

*Avenue Requests:* GetSize, SetSize

## StepButtons Property

Adds or removes step buttons from the slider. The default value is false, or no step buttons. When set to true, the slider displays step buttons providing another method for setting the slider value. When you click a step button, the slider will move by the value specified in the ValueIncrement property. If ValueIncrement is 0, the slider will move by the value specified in MainIncrement. If this is also 0, the slider will move 10 percent of the interval. Step buttons are not available under the UNIX operating system.

*Applies to:* Slider

*Avenue Requests:* HasStepButtons, ShowStepButtons



## Tag Property

Stores any additional text string that's needed for a control. ArcView does not use the Tag property, so you can use it without affecting any of the control's other property settings or causing any other side effects. The Tag property can only reference a string; use an ObjectTag to reference other objects.

*Applies to:* Button, Check box, Combo box, Control panel, Icon box, Label button, List box, Radio button, Slider, Text box, Text label, Text line, Tool

*Avenue Requests:* GetTag, SetTag

## Text Property

Specifies the text string to display in the text line or text box when the dialog is first displayed. If you change the text string, the dialog will retain the value when subsequently displayed. To initialize the text string each time you display the dialog, you need to add a line of code like this to the dialog's Open script:

```
self.FindByName("aTextLine1").SetText("Initial value")
```

ArcView does not check whether the text you supply for this property matches the text type you specify with the TextType property. For example, if you enter a character string here but define the text line to accept only integers, the character string will display on the dialog, but the end user will need to erase the character string before they can enter an integer.

*Applies to:* Text box, Text line

*Avenue Requests:* GetText, SetText

## TextType Property

Defines the types of characters that a text line can accept: alphanumeric characters, integers, or only real numbers. The first will be left-justified; the latter two will be right-justified.

If you specify a default text string with the Text property, it's a good idea that the string matches the type you specify here.

*Applies to:* Text line

*Avenue Requests:* GetTextType, SetTextType

## Title Property

Sets the title for the dialog. If you don't want a title bar at all on your dialog, set the `HasTitleBar` property to `false`.

*Applies to:* Dialog

*Avenue Requests:* `GetTitle`, `HasTitleBar`, `SetTitle`

## Update Property

Sets the name of the script to execute when an Update event occurs. The Update script typically performs some operation in response to a change in the dialog that triggers an Update event. For example, you might use an Update script to enable an OK button on your dialog once a user provides some input the dialog requires—such as typing some text into a text line control.

With dialogs, you control when to broadcast an Update event and which controls will respond to that event. This is different from how the Update property works with controls on the menu, button, and tool bar. For instance, ArcView broadcasts an Update event whenever a document is activated. The controls on the menu, button, and tool bar respond by running their respective Update scripts. In the case of controls on a dialog, ArcView does not attempt to determine which controls need to respond to the Update event; it's up to you to decide when a control's Update script should be run.

For a given control on your dialog, you specify which other controls should listen to it. When you want to run the Update scripts of the listening controls, you broadcast an Update event. For example, if you have a dialog that contains a text line and a button, and you want to dim the button when the text line is empty, you would make the button a listener of the text line. Typically, you set this broadcaster–listener relationship in the dialog's Open script:

```
'the self object is the dialog
aTextLine = self.FindByName("aTextLine1")
aButton = self.FindByName("aLabelButton2")
aTextLine.SetListeners({aButton})
```

If the text line is cleared out, it would broadcast an Update event causing the button's Update script to execute. For example, the text line's `Empty` property would reference a script that contained this code:

```
'the self object is the text line control
self.BroadcastUpdate
```

It would then be the responsibility of the button's Update script to dim the button.

A control can have many listeners. Thus, if other controls on your dialog also need to respond, you can add them to the list. For example, if you have a few more buttons on your dialog:

```
aTextLine.SetListeners({aButton1, aButton2, aButton3})
```

The list of listeners is dynamic—you can add or remove controls at any time.

```
aTextLine.AddListener(aButton4)
```

Listeners need not be confined to a particular dialog either. They may be part of other dialogs or included on ArcView's menu, button, and tool bar.

***Applies to:*** Button, Check box, Combo box, Control panel, Dialog, Icon box, Label button, List box, Radio button, Slider, Text box, Text label, Text line, Tool

***Avenue Requests:*** BroadcastUpdate, GetListeners, GetUpdate, SetListeners, SetUpdate

## Upper Property

Specifies the upper bound of the slider's range. The default value is 10.

After adjusting the slider's range, make sure the ValueIncrement property is set appropriately for the range. An inappropriate ValueIncrement can prevent the slider handle from moving.

***Applies to:*** Slider

***Avenue Requests:*** GetUpper, SetUpper

## Value Property

Sets the initial value for the slider the first time you display the dialog. The default setting for this property is 0. Once you change the slider's value, it will retain that value when you display the dialog again. If you want the slider to initialize to a particular value each time you display the dialog, you'll need to add a line of code such as this to the dialog's Open script:

```
self.FindByName("aSlider1").SetValue(4)
```

You can also set the slider's value equal to a percentage of the slider's interval with the SetValuePercentage request:

```
self.FindByName("aSlider1").SetValuePercentage(50)
```

***Applies to:*** Slider

***Avenue Requests:*** GetValue, SetValue, GetValuePercentage, SetValuePercentage

## ValueIncrement Property

Restricts the values that a slider can take on. The default value is 1. When ValueIncrement is 0, the slider can assume any real number defined within its range. When ValueIncrement is greater than 0, the slider's value will always be in the form

$\text{slider lower bound} + (i * \text{ValueIncrement})$

where  $i$  is a positive integer. For example, suppose you want to restrict a slider's value to an integer. Given a lower bound of 0 and an upper bound of 10, setting ValueIncrement to 1 allows the slider to assume all integer values between 0 and 10. A ValueIncrement of 2 restricts values to even numbers. To restrict the slider to odd numbers, you'd have to set the slider's lower bound to 1 and keep ValueIncrement at 2.

*Applies to:* Slider

*Avenue Requests:* GetValueIncrement, SetValueIncrement

## VerticalScroll Property

Adds a vertical scroll bar to a list box or text box.

If a text box does not have a vertical scroll bar, you can still scroll the text by dragging the mouse through the text box. If a list box does not have a vertical scroll bar, the height of its rows will be stretched or compressed to fit the height of the list box. To avoid this behavior set a row height for each row explicitly or add a vertical scroll bar.

*Applies to:* List box, Text box

*Avenue Requests:* HasVerticalScroll, SetVerticalScroll

## VisibleBorder Property

Determines whether the control panel will have a visible border. When True, the control panel will have a line drawn around its perimeter.

*Applies to:* Control panel

*Avenue Requests:* HasVisibleBorder, SetVisibleBorder

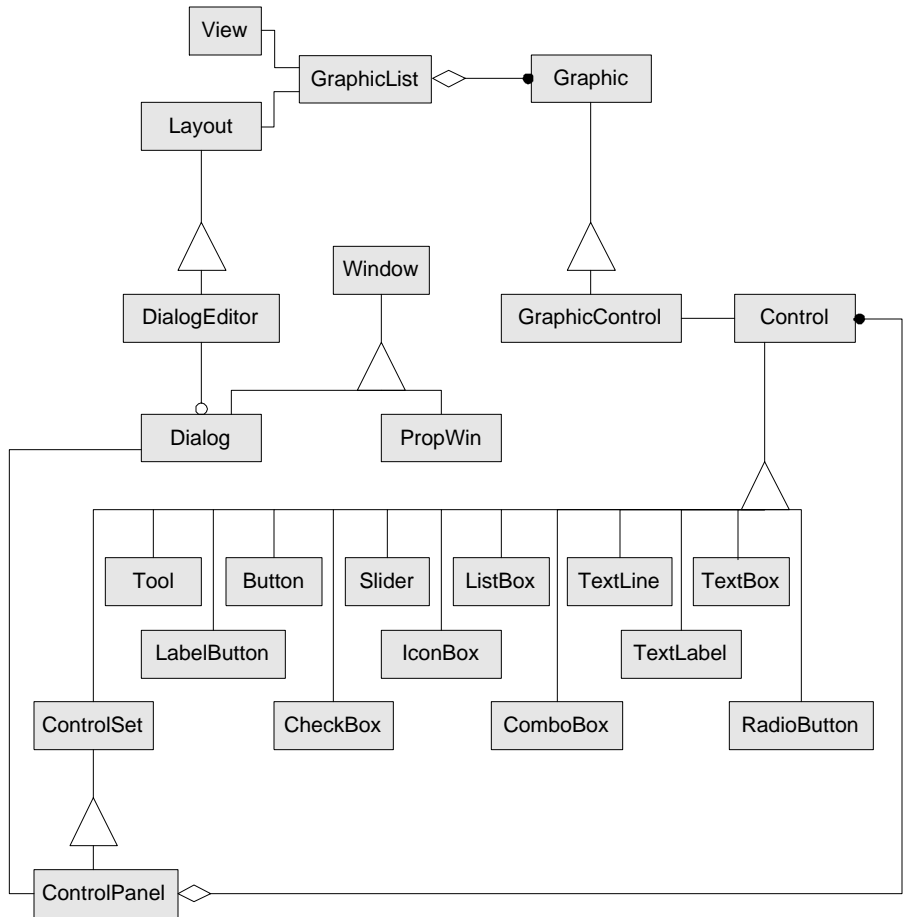
## APPENDIX B

# Object model diagram

The Dialog Designer implements several new classes that are incorporated into ArcView when you load the extension. Understanding the relationship between these classes will help you build applications with dialogs.

This appendix presents the object model diagram for the Dialog Designer and briefly describes each class.

# Object model



## Class descriptions

### Button

A Button control displays an icon and recognizes the Click event. A Click event occurs when the user clicks on a button or when an Avenue script issues the button Click request. Buttons can be used on the button bars of DocGUIs, in dialogs, or as graphic controls on views and layouts.

### CheckBox

A CheckBox control displays a Boolean value. A true value is represented by a checked box. A false value is represented by an unchecked box.

### ComboBox

A ComboBox control provides an interface for selecting an object from a list of related objects. A combo box consists of a set of rows, each of which is associated with a single object of any kind. One row is always presented to the user. This is the selected row. When the user wants to select another row, they click on a part of the combo box in order to drop down a single-column scrolling list that shows all rows in the combo box. The contents of a combo box can be defined from a list, dictionary, or VTab.

### Control

The Control class is the abstract class defining the ArcView user interface controls: menu choices, buttons, tools, spaces, and control sets. The Dialog Designer extension adds more controls that can be used in dialogs or as graphic controls on views and layouts.

### ControlPanel

A ControlPanel is a kind of control set that organizes controls in two dimensions inside a Dialog, rather than linearly inside a DocGUI. When you add a control to a control panel, you specify the bounding box for that control (rather than its index relative to other controls).

## ControlSet

ControlSet is the abstract class that defines general properties and methods for collections of controls. To display a control in the DocGUI, it must belong to a control set. Each DocGUI contains three control sets: a menu bar, a button bar, and a tool bar. In dialogs, controls can be organized in a control set called a control panel.

## Dialog

The Dialog class lets you organize a single task or set of related tasks onto a separate window, much like you can organize related tasks under a particular menu item or on the button bar. A dialog is a modal or modeless window that contains controls. This window is not clipped to the ArcView application window. It can be dragged anywhere on the computer screen. You can create dialogs interactively using a dialog editor document or by sending the Make request to the Dialog class, then add controls to it using other requests.

## DialogEditor

The DialogEditor class, a subclass of Layout, lets you interactively create dialogs. Instead of manipulating graphic shapes and frames positioned on a “page” as in a layout, you manipulate graphic controls positioned inside a “dialog.” The frame inside the dialog editor window represents the visible portion of the dialog. You can size the frame as needed to fit all your controls.

## GraphicControl

A GraphicControl is a kind of graphic that lets you place control objects on view or layout documents. By using graphic controls in these documents, you can merge the controls that manipulate the map with the map itself, providing a more intuitive and interactive experience for a map user.

## IconButton

An IconButton control displays an icon. The user cannot interact with an icon box. An icon box is associated with an ArcView Icon object. Refer to the class description in the on-line help for a list of the supported image types.

## LabelButton

A LabelButton control is a button that displays text rather than icons. Label buttons recognize the Click event. Label buttons are found at the top of the project window. They can also be used in dialogs or as graphic controls on views and layouts.



## ListBox

A ListBox control provides an interface for examining data organized into multiple rows and columns. Each row and column intersection is called a cell. The object attached to a cell is referred to as a cell value. Zero or more cells can be selected. The contents of a list box can be defined from a list, dictionary, or VTab.

## PropWin

A PropWin is a modeless window that displays properties for controls. You can access this window by double-clicking on a control in a dialog, view, or layout. In your scripts, you can access it using the class request PropWin.The.

## RadioButton

A RadioButton control cooperates with other radio buttons to show the selected choice among a small number of choices. Each radio button has a label that describes the choice. Place related radio buttons in a control panel; the control panel ensures that only one radio button in the group is selected at a time.

## Slider

A Slider control displays a selected value in a range of real values defined over a finite interval. A slider can be oriented either vertically or horizontally and has a handle that marks the current value.

## TextBox

A TextBox control presents an interface for displaying and editing multiple lines of text.

## TextLabel

A TextLabel control annotates a dialog, providing titles and instructions for using the dialog. The user does not interact with a text label.

## TextLine

A TextLine control is a single line of editable text, optionally including a label.

## Tool

A Tool control provides a means of interacting with the document display. A tool supports a special event, the Apply event. The Apply event is triggered whenever the user clicks on the active document display. Tools display icons. Tools can be used on the tool bars of DocGUIs, in dialogs, and or as graphic controls on views and layouts.

# Index

## A

Activate property 20, 43  
 AlwaysOnTop property 43  
 Apply property 13, 22, 43  
 AuxIncrement property 44  
 Avdlog.dat 32  
 Avdlog.dll 26, 27, 29, 32, 33

## B

Broadcasters and listeners 17  
 Button (class) 67  
 Buttons  
   defined 5  
   disabling. *See* Disabled property  
   hiding. *See* Invisible property  
   icon. *See* Icon property  
   naming 11  
   properties 36  
   scripts, executing. *See* Click property

## C

CanUnload script 29  
 Changed property 44  
 Check boxes  
   defined 5  
   disabling. *See* Disabled property  
   hiding. *See* Invisible property  
   initializing. *See* Selected property  
   naming 11  
   properties 36  
   scripts, executing  
     on click. *See* Click property  
 CheckBox (class) 67  
 Click property 9, 13, 45  
 Click scripts 17  
 Client/Server relationships 18  
 Close property 45  
 Closeable property 45  
 Combo boxes  
   defined 5  
   defining contents of 19

Combo boxes (continued)  
   emptying 28  
   hiding. *See* Invisible property  
   naming 11  
   properties 37  
   scripts, executing  
     on selection. *See* Select property  
   used 16  
 ComboBox (class) 67  
 Compiling a dialog 10  
 Control (class) 67  
 Control Fasteners 16  
 Control panels  
   borders. *See* VisibleBorder property  
   defined 5  
   disabling. *See* Disabled property  
   naming 11  
   properties 37  
 Control properties 4  
   and Avenue requests 9  
   PropWin (class) 69  
   reference 36  
   setting 9  
 Control Tools 22  
 ControlPanel (class) 67  
 Controls  
   adding  
     to dialogs 8, 12  
     to views and layouts 22  
   as GraphicControl 23  
   defined 4  
   disabling 13. *See also* Disabled property  
   enabling 14  
   finding by name 17  
   initializing 13  
   list of 5  
   naming 9, 11  
   responding to pan/zoom 22  
   scripts, attaching to 9  
   sizing at run time 16

Controls (continued)  
   status bar help 49  
   tab order. *See* NextControl property  
   tool tips 49  
   updating 17. *See also* Update property  
 ControlSet (class) 68  
 Cursor property 45

## D

Data  
   linking a dialog to 2, 15, 18  
 DDE (Dynamic Data Exchange) 3  
 DefaultButton property 45  
 Dialog (class) 68  
 Dialog Designer  
   loading  
     automatically with project 10  
     through Extension Manager 8  
   object model diagram 66  
 Dialog Editor  
   compiling 10  
   creating 8  
   described 4  
   finding in project 15  
   placing controls on a 8  
 Dialog.avx 32, 33  
 DialogEditor (class) 68  
 Dialogs  
   compiling 10  
   creating 8, 12  
   defined 2  
   delivering  
     in a project 33  
     in an extension 26–33  
     in default.apr 34  
     in system default.apr 34  
   dismissing 9. *See also* Close property; EscapeEnabled property  
   finding 14  
   finding named controls on 17

## Dialogs (continued)

- focusing input on. *See* Modal property
- initializing 13. *See also* Open property
- linking to data 2, 18
- naming 10
- on-line help, displaying from 49
- opening 19
- properties 37
- responding to a document 20. *See also* ServerActivated property
- responding to active document 20. *See also* DocActivate property
- responding when activated 20. *See also* Activate property
- running 10, 14, 19
- title 13
- updating controls on 17. *See also* Update property
- window, keeping on top. *See* AlwaysOnTop property
- Disabled property 46
- Disabling controls 13
- DLLs
  - avdlog.dll 26, 27, 29, 32, 33
  - dlogcore.\_\_\_\_ 33
- DocActivate property 16, 20, 46
- DocGUIs
  - delivering in extensions 32
- Drag property 47
- Dynamic Data Exchange (DDE) 3

**E**

- Empty property 47
- Enabling controls 14. *See also* Disabled property
- EscapeEnabled property 48
- Extension Manager 8
- Extensions
  - building your own 26–33
  - delivering DocGUIs 32
  - dialog.avx 32, 33
  - distributing files 32
  - dlogcore.\_\_\_\_ 33
  - loading the Dialog Designer 8
  - removing object tags 31

**F**

- FieldNamesVisible property 16, 48
- FocusLost property 48

**G**

- Graphic controls 23
- GraphicControl (class) 68

**H**

- HasTitleBar property 10, 48
- Help
  - getting technical support 6
  - setting on control 49
  - setting on dialog 49
- Help property 49
- HelpTopic property 49
- HiddenText property 50
- Horizontal property 50
- HorizontalScroll property 13, 16, 50

**I**

- Icon boxes
  - defined 5
  - hiding. *See* Invisible property
  - icon. *See* Icon property
  - naming 11
  - properties 38
  - supported image formats 38
- Icon property 50
- IconButton (class) 68
- Install script 28
- Interface
  - attaching a dialog to the 14, 19
- Internet
  - ESRI's home page 6
- Invisible property 51

**L**

- Label buttons
  - defined 5
  - disabling 13. *See also* Disabled property
  - enabling 14
  - hiding. *See* Invisible property
  - naming 11
  - properties 39
  - script, attaching to 9

## Label buttons (continued)

- scripts, executing. *See* Click property
- text label, setting 9
- used 13
- Label property 9, 13, 16, 22, 51
- LabelButton (class) 68
- LabelSize property 52
- Layouts
  - activating controls on 23
  - placing controls on 3, 21
- List boxes
  - adding elements to a 14
  - defined 5
  - disabling. *See* Disabled property
  - displaying tabular data in 18
  - emptying 13, 28
  - field names, showing on. *See* FieldNamesVisible property
  - hiding. *See* Invisible property
  - naming 11
  - properties 39
  - scripts, executing
    - on double-click. *See* Apply property
    - on selection. *See* Select property
  - scroll bars. *See* HorizontalScroll property; VerticalScroll property
  - selecting from. *See* SelectionStyle property
  - setting column widths 18
  - sorting elements in a 14
  - used 13, 16
- ListBox (class) 69
- Listeners and broadcasters 17
- Load script 28
- Loading the Dialog Designer 8
- Lower property 52

**M**

- MainIncrement property 52
- Make script 26
- Modal property 53

**N**

- Naming
  - controls 11

- Naming (continued)
  - dialogs 10
  - scripts 10
- NextControl property 53
- O**
- Object model diagram 66
- Object tags
  - removing from extensions 31
- Open property 13, 16, 53
- Open script 13
- Opening a dialog 19
- P**
- Passwords. *See* HiddenText property
- Projects
  - as extension source 26
  - delivering dialogs in 33
  - removing dialog editors 33
- Properties
  - of controls 43
- PropWin (class) 69
- R**
- Radio buttons
  - defined 5
  - disabling. *See* Disabled property
  - hiding. *See* Invisible property
  - naming 11
  - properties 39
  - scripts, executing
    - on click. *See* Click property
- RadioButton (class) 69
- ReadOnly property 54
- Remote Procedure Calls (RPC) 3
- Resizable property 54
- RPC (Remote Procedure Calls) 3
- Running a dialog 10
- Running controls on views and layouts 23
- S**
- Script Manager 9
- Scripts
  - attaching to controls 9
  - click 17
  - for extensions
    - canunload 29
- Scripts (continued)
  - for extensions (continued)
    - install 28
    - load 28
    - make 26
    - uninstall 30
    - unload 30
  - naming 10
  - update 17
- Select property 13, 16, 54
- Selected property 55
- SelectionMode property 55
- Self object 13, 29
- ServerActivated property 20, 56
- ServerClosed property 20, 57
- ServerDeactivated property 20, 57
- ServerDefinitionChanged property 20, 58
- ServerOpened property 20, 58
- ServerRecordsAdded property 21, 59
- ServerRecordsDeleted property 21, 59
- Servers 18
  - clearing 28
  - events on 20
  - setting 19
- ServerSelectionChanged property 16, 19, 20, 59
- Size property 60
- Slider (class) 69
- Sliders
  - defined 5
  - disabling. *See* Disabled property
  - hiding. *See* Invisible property
  - initializing. *See* Value property
  - naming 11
  - orientation. *See* Horizontal property
  - properties 40
  - restricting values. *See* ValueIncrement property
  - scripts, executing
    - on click. *See* Click property
    - on drag. *See* Drag property
  - tick marks, adding. *See* AuxIncrement property; MainIncrement property
  - upper bound. *See* Upper property
- Starting the Dialog Designer 8
- Status bar help, setting 49
- StepButtons property 60
- T**
- Tabular data
  - displaying in list boxes 18
  - responding to changes in 20
- Tag property 61
- Text boxes
  - defined 5
  - disabling. *See* Disabled property
  - hiding. *See* Invisible property
  - limiting characters. *See* Size property
  - naming 11
  - properties 40
  - scripts, executing
    - on click. *See* Click property
    - when emptied. *See* Empty property
    - when text changes. *See* Changed property
  - scroll bars. *See* HorizontalScroll property; VerticalScroll property
- Text labels
  - defined 5
  - disabling. *See* Disabled property
  - hiding. *See* Invisible property
  - naming 11
  - properties 41
- Text lines
  - defined 5
  - disabling. *See* Disabled property
  - hiding. *See* Invisible property
  - limiting characters. *See* Size property
  - naming 11
  - passwords, entering in 50
  - properties 41
  - scripts, executing
    - on click. *See* Click property
    - when emptied. *See* Empty property
    - when focus lost. *See* FocusLost property
    - when return key pressed. *See* Apply property
    - when text changes. *See* Changed property

Text lines (continued)  
  tabbing from. *See* FocusLost  
  property  
  text type, setting. *See* TextType  
  property  
  used 13, 22

Text property 61  
TextBox (class) 69  
TextLabel (class) 69  
TextLine (class) 69  
TextType property 61  
Title property 13, 16, 62  
Tool (class) 70  
Tool tips, setting 49  
Tools  
  defined 5  
  disabling. *See* Disabled property  
  hiding. *See* Invisible property  
  icon. *See* Icon property  
  naming 11  
  properties 42  
  scripts, executing  
    on click. *See* Click property  
    when click document. *See*  
    Apply property

## U

Uninstall script 30  
Unload script 30  
Update property 16, 62  
Update scripts 17  
  calling  
    by broadcasting Update event  
    17  
    directly 19  
Updating controls on dialogs 17  
Upper property 63

## V

Value property 63  
ValueIncrement property 64  
VerticalScroll property 64  
Views  
  activating controls on 23  
  finding controls on 23  
  placing controls on 3, 21  
VisibleBorder property 64  
VTab  
  as servers to dialogs 18, 20

VTab (continued)  
  displaying in a list box 18  
  responding to a change in selection  
  19

## W

World Wide Web  
  ESRI's home page 6