Chapter 11 PowerPoint Advanced Features

Once you build a basic presentation, your clients will soon be begging for more. Actually, some of these advanced features (like the Master Slide feature) can be so helpful, you'll want to use them right away. Others, like animations and transitions, are fun to watch (and program). Used judiciously, these will dazzle your clients!

Now that your clients are impressed, you're ready to tackle a little more pizzazz in your presentations. Master Slides is a feature that ensures that all your slides have a consistent appearance. This chapter also covers some fancy features to animate shapes, control the transitions between slides, add multimedia, and create hot spots that run other programs. At the end are ways to add notes to be printed with your presentation.

Achieving consistency with Master Slides

The SlideMaster object sets the defaults for each slide's layout and appearance. The SlideMaster stores the defaults for colors and fonts, standard item placement, as well as objects that should appear on any slide (perhaps a company logo in the corner). The SlideMaster object is referenced by the Presentation's SlideMaster property.

Backgrounds

The plain white background on all slides is something your clients will insist that you change. You can set backgrounds individually on every slide, but that's a lot of redundant code, and it's a big performance hit. One of the first SlideMaster properties to explore is the Background property. This sets the background of all slides to be the same using a minimum of code with maximum performance.

The Background property stores a ShapeRange object. While the ShapeRange object has a number of properties, the most useful from the Background property's viewpoint is the Fill property, which references a FillFormat object, the same object used to fill shapes. You can fill the background with solid, soft yellow like this:

```
#DEFINE rgbSoftYellow  RGB(255,255,192)
oPresentation.SlideMaster.Background.Fill.ForeColor.RGB =  rgbSoftYellow
```

To provide a patterned background, set the ForeColor and BackColor properties of the FillFormat object. Then use the Patterned method, which accepts one parameter, a numeric value corresponding to the pattern (msoPattern constants). The Pattern property is set to the same value you pass to the Patterned method.

```
#DEFINE msoPatternDottedGrid  45
#DEFINE rgbDarkGray           RGB(128, 128, 128)
#DEFINE rgbMediumGray         RGB(192, 192, 192

* For a Patterned Background:
```

```
WITH oPresentation.SlideMaster.Background.Fill
  .ForeColor.RGB = rgbDarkGray
  .BackColor.RGB = rgbMediumGray
  .Patterned(msoPatternDottedGrid)
ENDWITH
```

This code produces a medium gray slide with a dotted grid (about every eight pixels) in dark gray. We're not sure this is the most aesthetically pleasing background—the pattern is too small and busy to use for projected slides. The Pattern tab on the Fill Effects dialog box in **Figure 1** shows examples of the patterns. Use of patterns may best be left to smaller shapes, which can handle smaller patterns.

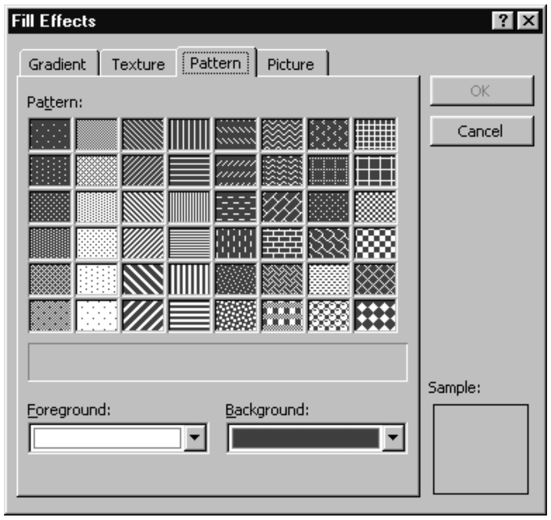


***Figure 1**. The Fill Effects dialog box. This dialog shows samples of the available patterns.*

The ForeColor represents the pattern, in this case, the dotted lines, and the BackColor shows through the pattern. Some patterns, like the dotted grid, show more background color; others show more foreground color.

Textures

If you or your client has used PowerPoint for very long, you know (or will soon be informed by your client) that there are textured backgrounds. Office provides about 25 preset textures to use. The FillFormat's PresetTextured method sets the background to the specified texture.

```
#DEFINE msoTextureSand  8
oPresentation.SlideMaster.Background.Fill.PresetTextured(msoTextureSand)
```

This line of code sets the background to a deep sand texture. If you want to find out what texture is in use, use the read-only PresetTexture property to return the numeric value.

What if your client wants a custom texture? Not to worry, PowerPoint provides a UserTextured method. This method takes one parameter, which is the fully qualified filename of a picture file to tile across the background. This filename is stored in the TextureName property (read-only). The TextureType property indicates which kind of texture is in use. The TextureType property returns msoTexturePreset (1) or msoTextureUserDefined (2).

Picture fills

You can use a bitmap as a background, too. The UserPicture method accepts a parameter consisting of a fully qualified bitmap filename. It forces the bitmap to take up the entire background—it does not tile it (use the UserTextured method to tile it). It uses some interesting smoothing techniques when you try to stretch a small bitmap across the whole slide. For an example, you might try:

```
oPresentation.SlideMaster.Background.Fill.UserPicture( ;
     GETENV("WINDIR" + "\TILES.BMP"))
```

This is a small bitmap that resembles brick; when enlarged, it has kind of a futuristic red and black look. This won't win any visual awards, but it is a striking example of how a small bitmap is smoothed over the whole screen. There isn't a documented property that corresponds to the UserPicture method to tell you what bitmap is used.

The Background object is available for the SlideMaster as well as Shapes. These methods and properties for textures, picture fills, and gradient fills are available for shapes, too.

Gradient fills

Perhaps the most sought after background is the gradient fill—you know the kind, a nice medium blue fades from the top to a nearly black color at the bottom. **Figure 2** shows the PowerPoint dialog box that allows you to select the gradient fills interactively. This is a handy cue to remember all the various properties for setting a gradient fill.

There are three gradient color types available in PowerPoint. The GradientColorType property stores the currently selected gradient type. It is a read-only property; separate methods are used to set the gradient. This is to your benefit, as the methods used to set each type take multiple parameters, ensuring that you set all the necessary properties for the specific gradient. Here are the three gradient color types:

- Preset colors: usually employing three or more colors, there are 24 presets that have names like Daybreak, Ocean, Fog, Moss, Wheat, and Parchment (which, incidentally, are the most professional looking schemes; the rest can look very garish depending on the text colors used).

- One-color fill: the selected color graduates to shades of the same color that are lighter or darker than the selected color (as light as white or as dark as black).

- Two-color fill: the first color graduates into the second color.

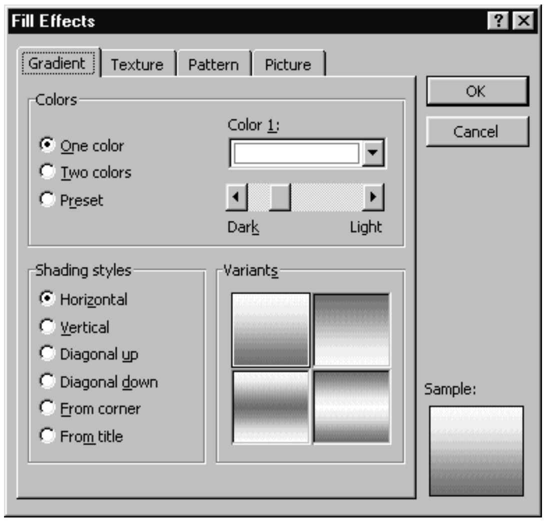


***Figure 2**. PowerPoint's gradient fill option dialog box. This is a handy visual reminder of the properties that need to be set for a gradient fill.*

The preset colors are set with the PresetGradient method. This method takes three parameters:

```
oPresentation.SlideMaster.Background.Fill.PresetGradient(nStyle,
               nVariant, nType)
```

nStyle	Numeric	Indicates the shading style of the gradient:			
		msoGradientDiagonalDown	4	msoGradientDiagonalUp	3
		msoGradientFromCenter	7	msoGradientFromCorner	5
		msoGradientFromTitle	6	msoGradientHorizontal	1
		msoGradientVertical	2		
nVariant	Numeric	Indicates the choice of color order (no constants):			
		1 = Color 1 to Color 2 (top left box in the gradient dialog)			
		2 = Color 2 to Color 1 (top right box in the gradient dialog)			
		3 = Color 1 to Color 2 back to Color 1 (lower left box in the gradient dialog)			
		4 = Color 2 to Color 1 back to Color 2 (lower right box in the gradient dialog)			
nType	Numeric	Indicates the preset color scheme. Some of the 24 are:			
		msoGradientDaybreak	4	msoGradientOcean	7
		msoGradientFog	10	msoGradientParchment	14
		msoGradientMoss	11	msoGradientWheat	13

Most clients may object to the preset colors. As stated before, many are garish, and those that are professional are probably overused by their competitors. Your client may be more sophisticated than the preset colors (by the way, this is an excellent argument to dissuade a client who chooses one of the garish presets—take a look at the Rainbow type [16] to see what we mean).

The next gradient type is the one-color gradient, which uses the OneColorGradient method to set the appropriate properties. Like the PresetGradient method, it also takes three parameters;

the first two are identical to the PresetGradient method:

```
oPresentation.SlideMaster.Background.Fill.OneColorGradient(nStyle,
              nVariant, nDegree)
```

nStyle	Numeric	The shading style of the gradient. See PresetGradient for constants.
nVariant	Numeric	The choice of color order. See PresetGradient for values.
nDegree	Numeric	A value from 0 to 1, representing the darkness of the resulting color. 0 is black, and 1 is white; .25 is a dark shade of the color, and .75 is a pale shade of the color. Represents Color 2.

Notice that there is no mention of what color to use as Color 1. The ForeColor property is used for Color 1. To set the background color on the slide to go from a medium royal blue to a pale shade of the same blue, use the following code:

```
#DEFINE msoGradientHorizontal  1
#DEFINE rgbMediumBlue          RGB(0, 0, 150)

WITH oPresentation.SlideMaster.Background.Fill
  .ForeColor.RGB = rgbMediumBlue
  .OneColorGradient(msoGradientHorizontal, 1, .75)
ENDWITH
```

The final gradient type is the two-color gradient. Yep, you guessed it: use the TwoColorGradient method. This one only takes two parameters:

```
oPresentation.SlideMaster.Background.Fill.TwoColorGradient(nStyle, nVariant)
```

nStyle	Numeric	The shading style of the gradient. See PresetGradient for constants.
nVariant	Numeric	The choice of color order. See PresetGradient for values.

The two colors are set by the Foreground and Background colors. The following example sets the color to graduate from a royal blue to a pale teal:

```
#DEFINE msoGradientHorizontal  1
#DEFINE rgbMediumBlue          RGB(  0,  0, 150)
#DEFINE rgbPaleTeal            RGB(192, 255,255)

WITH oPresentation.SlideMaster.Background.Fill
  .ForeColor.RGB = rgbMediumBlue
  .BackColor.RGB = rgbPaleTeal
  .TwoColorGradient(msoGradientHorizontal, 1)
ENDWITH
```

Since you're setting a series of properties by calling methods, just what properties are you setting? **Table 1** shows the FillFormat's properties that are set through each method. These are all read-only properties that can be queried to determine what the current settings are. Be sure to check the GradientColorType property first, then query only the properties that are applicable to the gradient type. Unused properties (such as GradientDegree, if a preset or two-color gradient type) are not reset to a default value when the gradient type is changed. For example, determining that the GradientDegree is .75 does not guarantee that a one-color gradient is used—you must query GradientColorType to be sure.

***Table 1**. The Gradient properties of the FillFormat object. Each type of gradient uses most of the Gradient properties—but not all. Check GradientColorType to ensure you set the appropriate properties for the gradient fill.*

Property	PresetGradient	OneColorGradient	TwoColorGradient
GradientColorType	msoGradientPresetColors (3)	msoGradientOneColor (1)	msoGradientTwoColors (2)
GradientStyle	nStyle parameter	nStyle parameter	nStyle parameter
GradientVariant	nVariant parameter	nVariant parameter	nVariant parameter
GradientDegree	NA	0 (black) – 1 (white)	NA
PresetGradientType	nType parameter	NA	NA

All the types of backgrounds (solid, patterned, picture, textured, and gradient) set properties through the FillFormat's methods. **Table 2** shows a compilation of all the properties that are set or used with each method. Remember, setting to a different background does not reset any unused properties to a default. Query the FillFormat's Type property to ensure which background format is in use.

***Table 2**. The FillFormat properties set by the various FillFormat methods. Remember that the Type property is the only indicator of the background format in use—do not rely on values in the other properties solely to determine the type of background format.*

Property	Solid	Patterned	Preset Texture	User Defined Texture
Read Only				
Type	msoFillSolid (1)	msoFillPatterned (2)	msoFillTextured (4)	msoFillTextured (4)
Pattern	-	msoPattern constants	-	-
TextureType	-	-	msoTexturePreset (1)	msoTextureUserDefined (3)
PresetTexture	-	-	msoTexture constants	-
TextureName	-	-	-	BMP filename
GradientColorType	-	-	-	-
GradientStyle	-	-	-	-
GradientVariant	-	-	-	-
PresetGradientType	-	-	-	-
GradientDegree	-	-	-	-
Read/Write				
ForeColor	✓	✓	-	-
BackColor	-	✓	-	-

Standardizing the appearance of text

Text is controlled by the TextStyles collection of the SlideMaster object. The collection has three objects. The first is the default style, used when a shape with text is added. The second is the title style used for all title placeholder objects. The last object is the body style, used for all of the other placeholders in an AutoLayout.

Within each TextStyle object is a collection of Level objects (Help refers to these as TextStyleLevel objects, though you access them through the Level property). Text levels are easiest to explain in terms of bullets: normally, each level is indented from the previous, and each level has a different bullet character and font characteristics—though each level does not necessarily have to be indented or bulleted. **Figure 3** shows the master slide as seen in PowerPoint, which shows the default characteristics for each level. PowerPoint supports five levels.

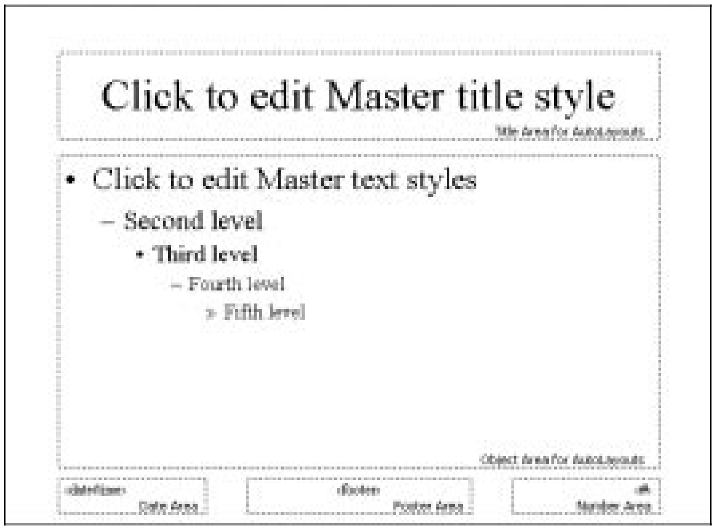
Each Level object contains a Font object and a ParagraphFormat object. The ParagraphFormat object contains the Bullet object. These objects are covered in Chapter 10, "PowerPoint Basics," and have the same properties and methods when used for Levels as they do for TextRanges.

You might use the TextStyles and Levels collections to change the color and font for the Title objects. Usually slide show backgrounds are a darker color, like medium to dark blue, to ensure readability. When the background is dark, the text needs to be light—perhaps a shade of yellow. Additionally, it is usual to use an eye-catching sans-serif font for a title. The following code demonstrates this. **Figure 4** shows the results.

***Table 2**, continued*

UserPicture	**Preset Gradient**	**One Color Gradient**	**Two Color Gradient**
msoFillPicture (6)	msoFillGradient (3)	msoFillGradient (3)	msoFillGradient (3)
-	-	-	-
-	-	-	-
-	-	-	-
-	-	-	-
-	msoGradientPresetColors (3)	msoGradientOneColor (1)	msoGradientTwoColors (2)
-	nStyle	nStyle	nStyle
-	nVariant	nVariant	nVariant
-	nType	-	-
-	-	0.0-1.0	-
-	-	✓	✓
-	-	-	✓

```
#DEFINE ppTitleStyle  2
#DEFINE rgbYellow     RGB(255,255,0)

WITH oPresentation.SlideMaster.TextStyles[ppTitleStyle].Levels[1].Font
  .Name = "Arial"
  .Color.RGB = rgbYellow
ENDWITH
```



Click to edit Master title style

Title Area for AutoLayouts

- Click to edit Master text styles
  - Second level
    - Third level
      - Fourth level
        - Fifth level

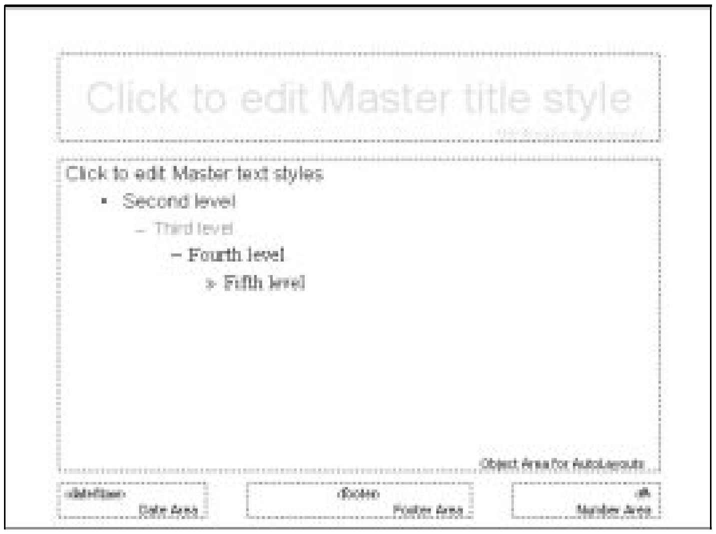
Object Area for AutoLayouts

<date/time> Date Area

<footer> Footer Area

<#> Number Area

***Figure 3**. The default SlideMaster as seen in PowerPoint. Notice the levels shown in the Object Area.*



Click to edit Master title style

Click to edit Master text styles
- Second level
  - Third level
    - Fourth level
      - Fifth level

Object Area for AutoLayouts

<date/time> Date Area

<footer> Footer Area

<#> Number Area

***Figure 4**. The SlideMaster in PowerPoint. This shows the results of the example code that sets the Titles and Placeholder default text.*

As the example illustrates, a Title uses the Levels collection. A Title can have multiple Levels, perhaps for a title and subtitle in the same rectangle. While levels are useful for changing fonts in Titles, they really excel in placeholders, especially when bullets are used.

The next example shows how to format the first three Levels in placeholders. The ppBodyStyle constant is used to reference the third TextStyle object. The first level is formatted to dark blue. The bullet character is a dot, but we've turned it off. The character is set so it can be turned on when needed. The second level is set similarly, but the bullet is left on. The third level is slightly smaller, medium cyan, and uses an en-dash bullet character. The result is shown in Figure 4.

```
#DEFINE ppBodyStyle   3
#DEFINE rgbDarkBlue   RGB(0,   0, 128)
#DEFINE rgbMediumCyan RGB(0, 192, 192)

* Set Level One
WITH oPresentation.SlideMaster.TextStyles[ppBodyStyle].Levels[1]
  WITH .Font
    .Name      = "Arial"
    .Color.RGB = rgbDarkBlue
    .Size      = 20
  ENDWITH
  WITH .ParagraphFormat.Bullet
    * Set bullet character to a dot (use 149 if using PowerPoint 97)
    .Character = 8226
    .Visible   = .F.    && Don't show the bullet!
  ENDWITH
ENDWITH

* Set Level Two
WITH oPresentation.SlideMaster.TextStyles[ppBodyStyle].Levels[2]
  WITH .Font
    .Name      = "Arial"
    .Color.RGB = rgbDarkBlue
    .Size      = 20
  ENDWITH
  * Set bullet character to a dot (use 149 if using PowerPoint 97)
  .ParagraphFormat.Bullet.Character = 8226

ENDWITH

* Set Level Three
WITH oPresentation.SlideMaster.TextStyles[ppBodyStyle].Levels[3]
  WITH .Font
    .Name      = "Arial"
    .Color.RGB = rgbMediumCyan
    .Size      = 18
  ENDWITH
  * Set bullet character to an en-dash (use 150 if using PowerPoint 97)
  .ParagraphFormat.Bullet.Character = 8211
ENDWITH
```

Similar code can be used to set the default text for added shapes using the constant ppDefaultStyle to access the first TextStyle object.

Standardizing colors with ColorSchemes

An alternative to setting the colors on the SlideMaster is to use a ColorScheme object, which stores the colors for the eight standard elements on a slide. **Figure 5** shows the PowerPoint dialog box that allows you to customize the colors. To display it in PowerPoint, select Format|Slide Color Scheme… from the menu, then click the Custom tab. Up to 16 ColorScheme objects can exist; they're stored in the ColorSchemes collection.

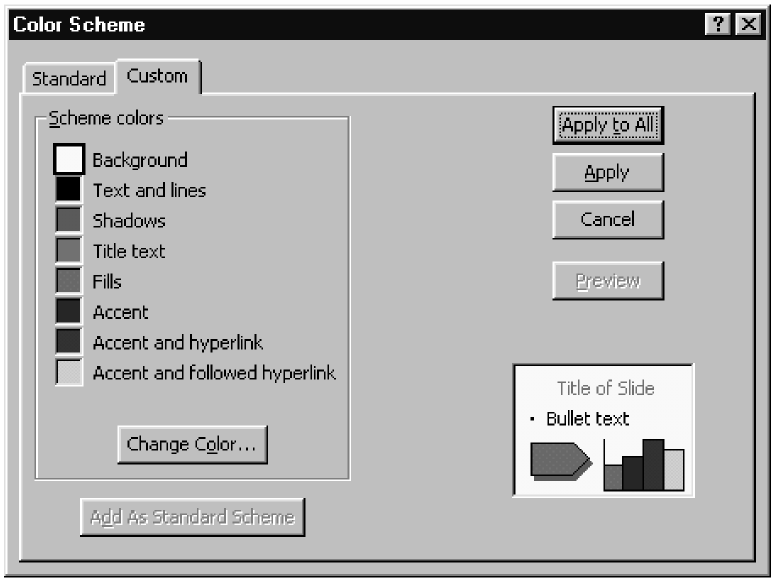


***Figure 5**. PowerPoint's Color Scheme dialog box. Color schemes set the eight standard colors for objects.*

Although the ColorSchemes collection belongs to the Presentation object, any ColorScheme object can be referenced by the SlideMaster object, or by any slide. The Help file has quite a lengthy discussion of how to use multiple ColorSchemes within a presentation (for example, having one ColorScheme for title slides, and another for standard slides).

The most basic and practical use of a single ColorScheme is to dictate the colors of the eight elements on the SlideMaster. A ColorScheme contains a collection of eight colors, each referenced by a constant. The constants for the eight colors are: ppBackground (1), ppForeground (2), ppShadow (3), ppTitle (4), ppFill (5), ppAccent1 (6), ppAccent2 (7), and ppAccent3 (8). Beware: setting the background and fill colors in a ColorScheme overrides any of the Background or Fill object properties that are set—if you set up a gradient fill background, do not set the background color of the color scheme!

The following code changes the title colors to red and the text (ppForeground) to dark blue. Be sure to use this code instead of setting colors in the SlideMaster. Colors set explicitly through the SlideMaster's TextStyles will take precedence over the ColorSchemes (though if you look at the dialog box, the ColorSchemes color will change, but it won't change the colors on the slides).

```
#DEFINE ppTitle      4
#DEFINE ppForeground 2
#DEFINE rgbRed       RGB(255, 0,   0)
#DEFINE rgbDarkBlue  RGB(  0, 0, 128)

WITH oPresentation.SlideMaster.ColorScheme
  .Colors[ppTitle].RGB = rgbRed
  .Colors[ppForeground].RGB = rgbDarkBlue
ENDWITH
```

More on SlideMaster shapes

The five shapes on the SlideMaster are exactly that: Shape objects. You can set any of the Shape properties, not just fonts. Changing the size or borders on the SlideMaster shapes affects the placeholders on all shapes. To access the properties of the shapes, use the Shapes collection. The indices are from 1 to 5, in this order: Title Area, Object Area, Date Area, Footer Area, and Number Area. With the myriad of constants available in VBA, it is surprising there are not constants for these object. (If you expect to work with them a lot, you can certainly define your own.)

All shapes get their color from the ColorScheme object on the SlideMaster. In the example in the previous section, all the text is dark blue. Perhaps you want the footer area to be dark green instead. Override the ColorScheme setting by accessing the shape directly. The following lines of code set the text of the footer to “Tasmanian Traders” and set the color to dark green.

```
#DEFINE rgbDarkGreen  RGB(0, 128, 0)

WITH oPresentation.SlideMaster.Shapes[4].TextFrame.TextRange
  .Font.Color.RGB = rgbDarkGreen
  .Text = "Tasmanian Traders"
ENDWITH
```

The first two shapes, the Title Area and the Object Area, get their font properties from the TextStyles objects. The other three (Date Area, Footer Area, and Number Area) are not affected by TextStyles. Set font properties of these three objects separately.

Fancy features

Warning: the following features are fun to program and interesting to watch. However, the maxim “less is more” is very appropriate here. There is elegance in simplicity. Used sparingly, these features can set your presentation apart from others. Overuse them and the audience tires of all the gimmicks, leading them to ignore the slides and, quite possibly, the speaker who is presenting them. Remember, the audience sees only the presentation, not the way-cool tricks in your code.

Animations

Animations are special effects—visual and audio—that highlight important elements in a slide. Some animation effects that PowerPoint supports are objects flying in from the edge (choose your edge), text appearing letter-by-letter, and sound files playing. These animations can be timed to happen sequentially or all at once. They can happen automatically, or in response to mouse clicks. There are so many combinations and permutations of animations; we won't attempt to cover each and every one of them. The Animation dialog gives you an excellent interface in which to explore them. We'll give you an idea of what's possible to get you started.

There are 19 animation effects supported by PowerPoint 2000, many with several options, bringing the total number of effects to about 80. **Figure 6** shows PowerPoint's Custom Animation dialog, with the Effects dropdown box shown open. This dialog also gives some insight into the available collections, objects, and properties. In the upper left corner, there is a list of all the objects on the slide; each can be animated separately (or not animated at all). The Effects tab is shown, displaying the many properties involved in customizing the effect. The other tabs indicate that there are properties to set the Order and Timing of effects, to animate Charts (if any are present), and to add Multimedia effects.

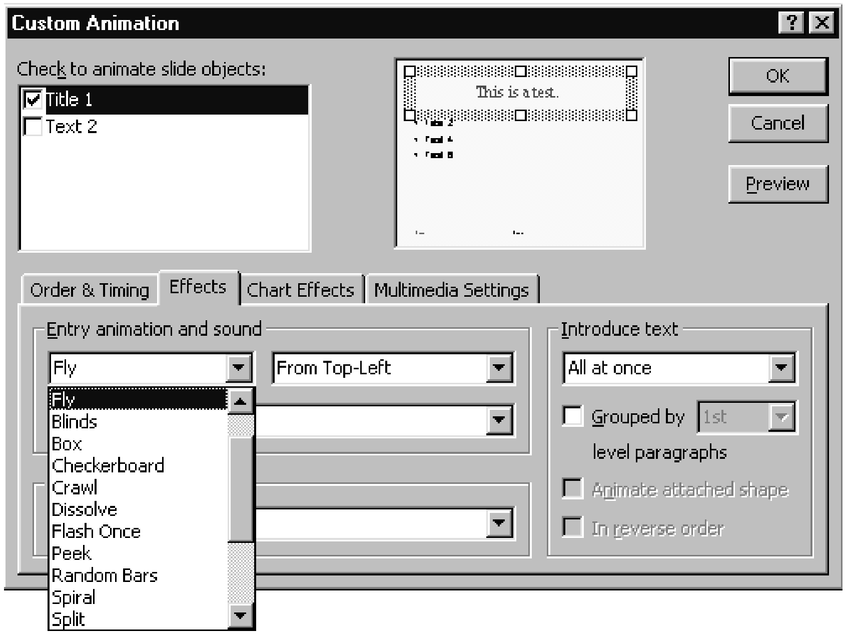


***Figure 6**. PowerPoint's Custom Animation dialog. The dropdown for Effects is open, showing many of the available animation effects.*

The AnimationSettings object stores all the objects, collections, and properties that pertain to animations. The AnimationSettings object is stored in the AnimationSettings property of every Shape object. Shapes on the SlideMaster can be animated, too.

Setting the effect

The EntryEffect property sets the effect. This property is set to a numeric value, for which there are VBA constants. The constants are set up to combine the two fields in the dialog box describing what is to happen (shown in Figure 6 as "Fly") and any available options for that effect, such as where it might move from (shown in Figure 6 as "From Top-Left"). Some of the constants for the effects are: ppEffectAppear (3844), ppEffectFlashOnceFast (3841), ppEffectFlashOnceMedium (3842), ppEffectFlyFromLeft (3329), ppEffectFlyFromTopLeft (3333), ppEffectRandom (513), ppEffectZoomIn (3345), and ppEffectZoomInSlightly (3346). Use the Object Browser to look up the other 72 constants. See "Take me for a browse" in Chapter 2 for details on the Object Browser.

The following code sets the first shape on the slide to fly in from the left:

```
#DEFINE ppEffectFlyFromLeft   3329
oSlide.Shapes(1).AnimationSettings.EntryEffect = ppEffectFlyFromLeft
```

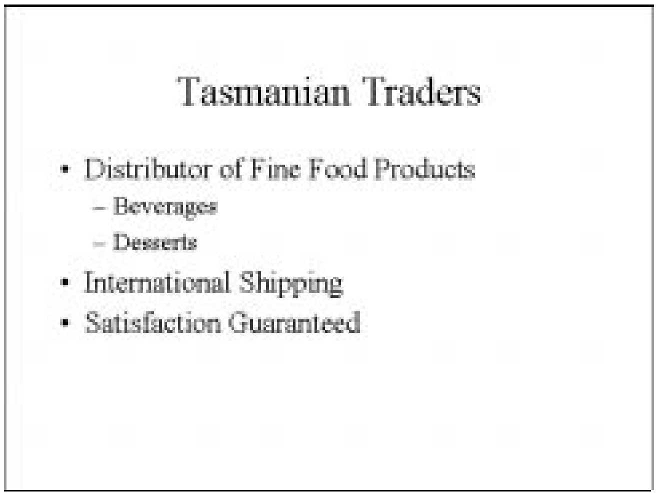
Modifying the effects

A very popular effect is to have a slide containing bulleted text, and have each of the bullets fly in from the left as the user clicks the mouse. A modification to the "fly in from left" effect is to set which levels of bullets fly in together. The TextLevelEffect tells which levels are animated. TextLevelEffect is set to a numeric value—some of the constants are ppAnimateByFirstLevel (1), ppAnimateBySecondLevel (2), ppAnimateByAllLevels (16), and ppAnimateLevelNone (0). Setting the effect to animate the first level sends each first level bullet in separately, with all its subordinates. Setting the effect to animate the second level sends each first and second level item in separately—any subordinates to the second level come in with their parent.

The following code generates an example Tasmanian Traders marketing slide, showing a bulleted list with two levels. The code sets the title and bullets to fly in from the left, one each time the mouse is clicked. The code is set to fly each of the bullets in separately (at level 2). Modify the code for a second run by changing the TextLevelEffect to ppAnimateByFirstLevel, to see that the first bullet flies in with its two subordinate bullets. **Figure 7** shows the resulting slide.

```
#DEFINE ppEffectFlyFromLeft    3329
#DEFINE ppLayoutText           2
#DEFINE ppAnimateByFirstLevel  1
#DEFINE ppAnimateBySecondLevel 2

* Add a slide--title and text objects on the layout

oSlide = oPresentation.Slides.Add(1, ppLayoutText)

* Put in the title text and set the effect
WITH oSlide.Shapes[1]
  .TextFrame.TextRange.Text = "Tasmanian Traders"
  .AnimationSettings.EntryEffect = ppEffectFlyFromLeft
ENDWITH

* Put in some bullets (at 2 levels) and set the effect
m.BulletString = "Distributor of Fine Food Products" + CHR(13) + ;
                 "Beverages" + CHR(13) + ;
```

```
                   "Desserts" + CHR(13) + ;
                 "International Shipping" + CHR(13) + ;
                 "Satisfaction Guaranteed"
WITH oSlide.Shapes[2]
  .TextFrame.TextRange.Text = m.BulletString

  * Indents "Beverages" and "Desserts"
  .TextFrame.TextRange.Sentences[2, 2].IndentLevel = 2

  .AnimationSettings.EntryEffect = ppEffectFlyFromLeft
  .AnimationSettings.TextLevelEffect = ppAnimateBySecondLevel

  * Run this again, but change the TextLevelEffect constant in the line above
  * to ppAnimateByFirstLevel to see how setting it to first level brings
  * in the two "level 2" bullets with their "level 1" parent.

ENDWITH
```



Tasmanian Traders

- Distributor of Fine Food Products
  - Beverages
  - Desserts
- International Shipping
- Satisfaction Guaranteed

***Figure 7**. An example of a bulleted list. Levels are used to format the bullets and text. Level 1 has a circular bullet. Level 2 has a dash for a bullet, and has slightly smaller text.*

Automatic timing

To eliminate manual mouse clicks and let the show run automatically, use the AdvanceMode property. There are two constants: ppAdvanceOnClick (1) and ppAdvanceOnTime (2). When the AdvanceMode property is set to ppAdvanceOnTime, it checks the number of seconds stored in the AdvanceTime property. Add the following lines to the end of the previous example, and it automatically animates items one second apart.

```
#DEFINE ppAdvanceOnTime 2

* Change the Title shape's settings
WITH oSlide.Shapes[1].AnimationSettings
```

```
  .AdvanceMode = ppAdvanceOnTime
  .AdvanceTime = 1
ENDWITH

* Change the bullet text shape's settings
WITH oSlide.Shapes[2].AnimationSettings
  .AdvanceMode = ppAdvanceOnTime
  .AdvanceTime = 1
ENDWITH
```

If there are multiple shapes, the order in which the shapes are animated is set through the AnimationOrder property. By default, AnimationOrder is set to the creation order of the Shape (assuming, of course, that the Shape is to be animated at all). By setting the AnimationOrder property, you can change the order in which the objects are animated.

These are just a few of the many properties and methods available to animate shapes. See the Help file under “AnimationSettings Object” for a comprehensive list.

Transitions

Fades, dissolves, wipes; these are all various ways of transitioning between slides. Transitions can be applied to the SlideMaster, which affects all slides, or just to a single slide. The transitions are stored in the SlideShowTransition object. The SlideShowTransition object is similar to the AnimationSettings object—transitioning to another slide is similar to transitioning a single object onto the screen.

The EntryEffect property is used to store the transition effect. It uses the same set of constants as the AnimationSettings’ EntryEffect property. Not all constants for EntryEffects are available for transitions, though (for example, ppFlyInFromLeft is only for animations).

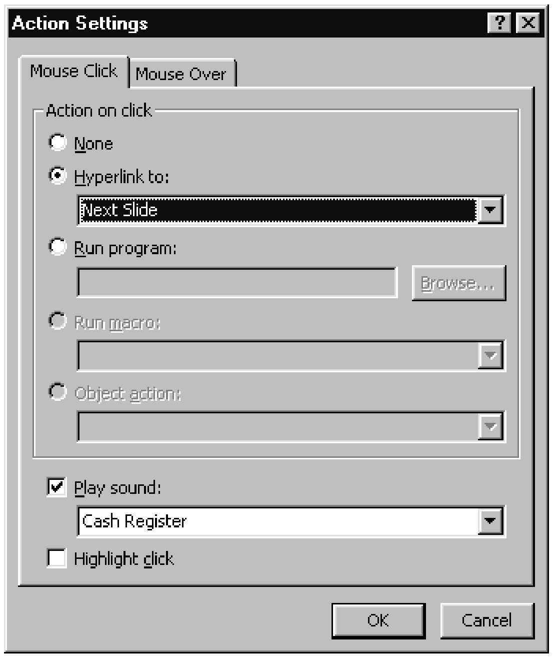
Like AnimationSettings, the SlideShowTransition object has properties to advance automatically. The syntax is different than AnimationSettings, though. The SlideShowTransition object has an AdvanceOnTime logical property. The AdvanceTime property stores the number of seconds (just as in the AnimationSettings object).

The following code adds a transition to the first slide to “cover down” and automatically advance after two seconds. If this code were applied to the SlideMaster, it would affect every slide, but as written here, it only affects the first slide.

```
#DEFINE ppEffectCoverDown  1284

WITH oPresentation.Slides[1].SlideShowTransition
  .EntryEffect = ppEffectCoverDown
  .AdvanceOnTime = .T.
  .AdvanceTime   = 2
ENDWITH
```

Taking action

PowerPoint allows specific actions to be taken when the mouse is moved over or clicked on a shape. Actions include jumping to a specific slide in the slide show, running another slide show, running a separate program, playing a sound, as well as a few others. **Figure 8** shows PowerPoint’s interactive Action Settings dialog, which lists the possible actions.



Hyperlink to: Next Slide
Run program:
Play sound: Cash Register

***Figure 8**. PowerPoint's Action Settings dialog. Action settings define the action taken on mouse overs and mouse clicks.*

The ActionSettings collection stores these actions. The collection contains two ActionSettings objects: one for a mouse click and one for a mouse over. The indices for the collections are ppMouseClick (1) and ppMouseOver (2).

The Action property is the property that controls the action. Set the Action property to one of the constants listed in **Table 3**.

***Table 3**. Where the action is. Set the Action property to determine the action taken when a user clicks on a shape.*

Constant	Value	Constant	Value
ppActionNone	0	ppActionHyperlink	7
ppActionNextSlide	1	ppActionRunMacro	8
ppActionPreviousSlide	2	ppActionRunProgram	9
ppActionFirstSlide	3	ppActionNamedSlideShow	10
ppActionLastSlide	4	ppActionOLEVerb	11
ppActionLastSlideViewed	5	ppActionPlay	12
ppActionEndShow	6		

The following code adds an arrow shape in the lower right corner of a slide, and sets its Action property to move to the previous slide.

```
#DEFINE msoShapeRightArrow   33
#DEFINE ppMouseClick          1
```

```
#DEFINE ppActionPreviousSlide 2

oShape = oSlide.Shapes.AddShape(msoShapeRightArrow, 600, 450, 50, 50)
oShape.ActionSettings[ppMouseClick].Action = ppActionPreviousSlide
```

PowerPoint has a feature called Action Buttons, which are a series of predefined buttons. These buttons have a consistent look, and they help the presentation designer maintain a consistent look and feel. When entered interactively in PowerPoint, the dialog box comes up with the Action defaults for the type of button selected. For example, placing the button with the End picture on it sets the Action default to jump to the end of the slide show. When these buttons are added in code (Automation or VBA macros), no default Action is specified—you must specify it explicitly using code like that shown previously.

The ActionSettings collection has additional properties to support other kinds of Actions. To enter a hyperlink, first set the Action property to ppActionHyperlink, then set the properties of the Hyperlink object. To create a hot link, set the Address property of the Hyperlink object, which stores the URL. Here's a code sample that adds a shape and attaches a hyperlink to a URL.

```
#DEFINE msoShapeRectangle 1
#DEFINE ppMouseClick      1
#DEFINE ppActionHyperlink 7

oShape = oSlide.Shapes.AddShape(msoShapeRectangle, 300, 200, 150, 100)
WITH oShape.ActionSettings[ppMouseClick]
  .Action = ppActionHyperlink
  .Hyperlink.Address = "http://www.hentzenwerke.com"
ENDWITH
```

You can even run any program, including compiled FoxPro programs, from a mouse click or mouse over. This is accomplished through the Run method. Pass it the fully qualified program name.

```
#DEFINE msoShapeRectangle 1
#DEFINE ppMouseClick      1

oShape = oSlide.Shapes.AddShape(msoShapeRectangle, 300, 200, 150, 100)
oShape.ActionSettings[ppMouseClick].Run = ;
      "C:\Program Files\Microsoft Office\Office\WINWORD.EXE"
```

Multimedia

In this world of high-tech movies and video games, multimedia in business presentations is almost expected. Multimedia has the power to enhance a presentation, making it more interesting to watch and easier for the viewer to retain what was presented. However, there's a fine line between "very interesting" and "very distracting"—again, we listen to the maxim "less is more."

While you are developing automated presentations, be sure that your program can handle scaling to the wide variety of hardware available for presentations. Even today, not all computers have good sound systems (or speakers that are decent enough to project to the whole room full of attendees), high-powered graphics cards, and lots of RAM and processor to allow the presentation to be successful. Ensure that your users have an alternative to awesome multimedia displays—if the presentation computer can't handle multimedia, the "incredibly awesome" presentation quickly is perceived as "incredibly awful."

Sounding off

Sounds are managed through the SoundEffect object. The ActionSettings, AnimationSettings, and SlideShowTransition objects (discussed earlier in this chapter) each have a SoundEffect property to access the SoundEffect object.

There are a number of sound effects available in PowerPoint without using separate sound files. These built-in sounds are accessed using the names shown in any of the PowerPoint sound effect dropdown boxes. **Figure 9** shows the Play sound dropdown on the Action Settings dialog.

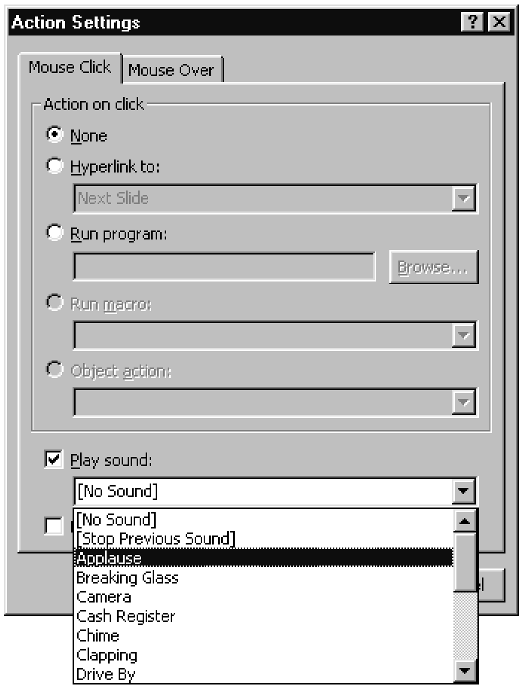


***Figure 9**. PowerPoint's Action Settings dialog, showing the built-in sound effects.*

Set the Name property to the text string corresponding to the sound you wish to use. This is an exception to the long list of numeric constants usually used by VBA! The list of options are: Applause, Breaking Glass, Camera, Cash Register, Chime, Clapping, Drive By, Drum Roll, Explosion, Gun Shot, Laser, Ricochet, Screeching Brakes, Slide Projector, Typewriter, and Whoosh.

```
#DEFINE ppMouseClick  1
oShape.ActionSettings[ppMouseClick].SoundEffect.Name = "Slide Projector"
```

Setting the Name property automatically sets the Type property to ppSoundFile (2). To turn off the sound, set the Type property to ppSoundNone (0). The constant ppStopPrevious (1) stops any sound currently being played.

To specify a WAV file to play, use the ImportFromFile method to both link to the WAV file and set the object to play it. Pass the fully qualified WAV filename. Once you have imported the WAV file, it is added to the list of sounds available so it can be referenced by other objects (by its filename without the path), just like the built-in sounds. The following lines of code import the WAV file, set the shape to play it when clicked, and then use the imported file by its name to set the same sound to play on the slide show transition.

```
#DEFINE ppMouseClick  1

SoundFile = GETENV("WINDIR") + "\Media\Chord.WAV"

oShape.ActionSettings[ppMouseClick].SoundEffect.ImportFromFile(SoundFile)
oSlide.SlideShowTransition.SoundEffect.Name = "CHORD.WAV"
```

The SoundEffect object has a Play method, which plays the sound on demand. Use the Play method only when you want the sound to play while your code is running; the use of the Play method does not affect whether sounds play in the slide show. It plays the sound set for the specified object:

```
oSlide.SlideShowTransition.SoundEffect.Play()
```

This line plays whatever sound is set in the slide's SlideShowTransition object.

Motion

To add video clips, you use the Shapes collection's AddMediaObject method. As Chapter 10, "PowerPoint Basics," points out, while AutoLayouts exist with placeholders for OLE objects, they cannot be used with Automation or macros.

The AddMediaObject method takes up to five parameters. The first is the filename, and is not optional. Next are the Left and Top. These are optional; the default is zero for both. The last two, Width and Height, are also optional, and default to the width and height of the object. The following line of code shows how to add one of FoxPro's sample AVI files (it is a spinning globe, not a fox, as the name seems to indicate).

```
oShape = oSlide.Shapes.AddMediaObject( ;
         _SAMPLES + "Solution\Forms\Fox.AVI", 240, 156)
```

This adds the AVI file roughly centered in the slide.

There wasn't any magic involved in finding the Left and Top parameters. It was done interactively. Using the FoxPro Command Window, we opened an instance of PowerPoint and added a new presentation with a blank slide. Then we issued the preceding command, leaving out the Left and Top parameters, which placed the image in the upper left corner. Activating PowerPoint, we moved the image using the *mouse to the location we wanted. Back in FoxPro's Command Window, we asked PowerPoint for the Top and Left properties, and typed the results into our code:*

```
? oShape.Left
? oShape.Top
```

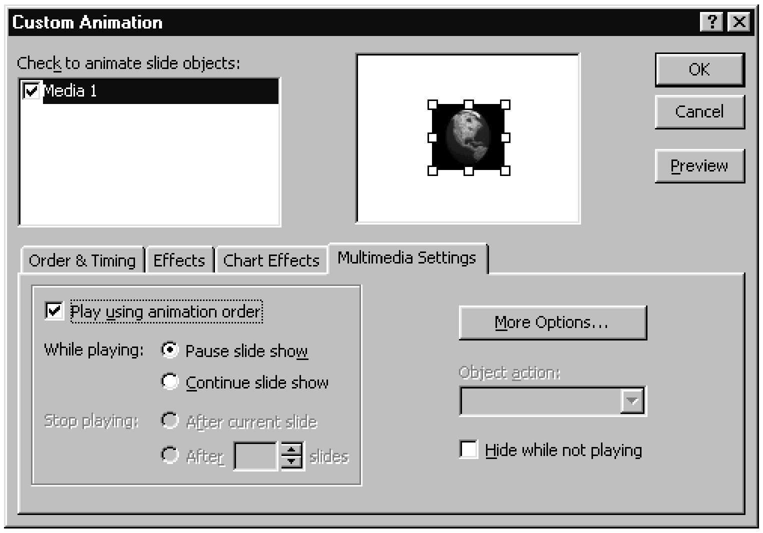
It's important to remember to let the tools do the work for you. FoxPro's object references persist until the variable is released, or the object in PowerPoint is unavailable (whether the presentation is closed or the object itself is deleted). Setting a reference to an object in FoxPro, switching to PowerPoint to interactively manipulate the PowerPoint objects, then querying the properties from FoxPro is a very powerful way to quickly determine the desired properties of the PowerPoint objects.

Adding the media clip with AddMediaObject sets it to play when the user clicks on it during the presentation. The AnimationSettings and ActionSettings objects change the behavior, setting it to play when the slide is selected, to continuously play while the slide is viewed, or to play when the mouse is moved over it. The ActionSettings object, which pertains to mouse clicks and movement, has no additional properties for multimedia (see the section "Taking action" earlier in this chapter). The AnimationSettings object, however, does have a number of properties relating to multimedia.

Figure 10 shows the Custom Animation dialog box, with the Multimedia Settings tab selected. This tab sets the properties of the AnimationSettings' PlaySettings object. The PlaySettings object contains a number of properties that determine how the media clip plays during a slideshow. The check box labeled "Play using animation order" corresponds to the logical PlayOnEntry property. When PlayOnEntry is true, it plays the media clip when the slide is displayed (based on the AnimationSettings object's AnimationOrder property). PlayOnEntry also respects the other AnimationSettings properties, such as AdvanceMode and AdvanceTime—if other objects that are higher in the order need mouse clicks to animate, then those lower in the order aren't animated until the mouse clicks force the preceding animations to happen. Set PlayOnEntry to false to ensure that the user must animate it manually.

The two radio buttons that set the "While playing" action correspond to the PauseAnimation property. When set to true (when the radio button reads "Pause slide show"), other automatic features of the slide show wait until the video clip has finished playing. When set to false, the other automatic features are played at the same time. This can be useful if you want several animated GIFs to play simultaneously. Set PauseAnimation to true when you only have one object to play, and do not want the slide show to advance before the object completes its play.

On the Custom Animation dialog, the Multimedia Settings tab has a "More Options" button, which corresponds to more animation properties. The check boxes correspond to the LoopUntilStopped and RewindMovie properties. To loop the animation until the user selects another action (or another action is automatically scheduled to run), set the LoopUntilStopped property to true. Setting it to false runs it once. The RewindMovie property controls whether the movie returns the view to the first frame when finished (true) or whether the movie stays on the last frame (false).



[✓] Media 1

[✓] Play using animation order

While playing: [✓ Pause slide show] Continue slide show

[ ] Hide while not playing

***Figure 10**. Multimedia Custom Animation settings. The Multimedia Settings tab shows the properties that are available to media clips.*

A common scenario is to play the video continuously when the slide is selected. The following code adds the video clip object and sets the appropriate properties:

```
oShape = oSlide.Shapes.AddMediaObject( ;
         _SAMPLES + "\Solution\Forms\Fox.AVI", 240, 156)
WITH oShape.AnimationSettings.PlaySettings
  .PlayOnEntry = .T.
  .LoopUntilStopped = .T.
ENDWITH
```

Many properties, such as PlayOnEntry, can be set with a logical value (.T. or .F.) or a numeric value (1 or -1). However, when you read them, they will always be numeric. This is a “feature” of how FoxPro’s logical values are translated to a numeric property; the native numeric value will always be returned.

Adding notes

Notes can be entered for each slide. Notes are displayed in the bottom panel (in PowerPoint 2000), or printed with a picture of the slide on the top and the notes below, one slide per page. The notes can be used as speaker notes, or printed and passed out as handouts, with the notes annotating the slide images.

Explaining how to put notes on each slide actually is easier if we start by explaining the notes master. **Figure 11** shows the notes master view in PowerPoint. Like the Placeholders collection for slides (discussed in Chapter 10, “PowerPoint Basics”), the indices of the Placeholders collection make sense when we see them in the notes master view: 1 is the header area, 2 is the date area, 3 is the SlideMaster object, 4 is the notes body area, 5 is the footer area, and 6 is the number area (Microsoft didn't provide constants for these, but you can certainly define your own). Just as with the SlideMaster (see "Achieving consistency with Master Slides" earlier in this chapter), these Placeholders can be formatted with color, fonts, and standardized text. The NotesMaster object contains the properties that are accessed when printing the notes pages.

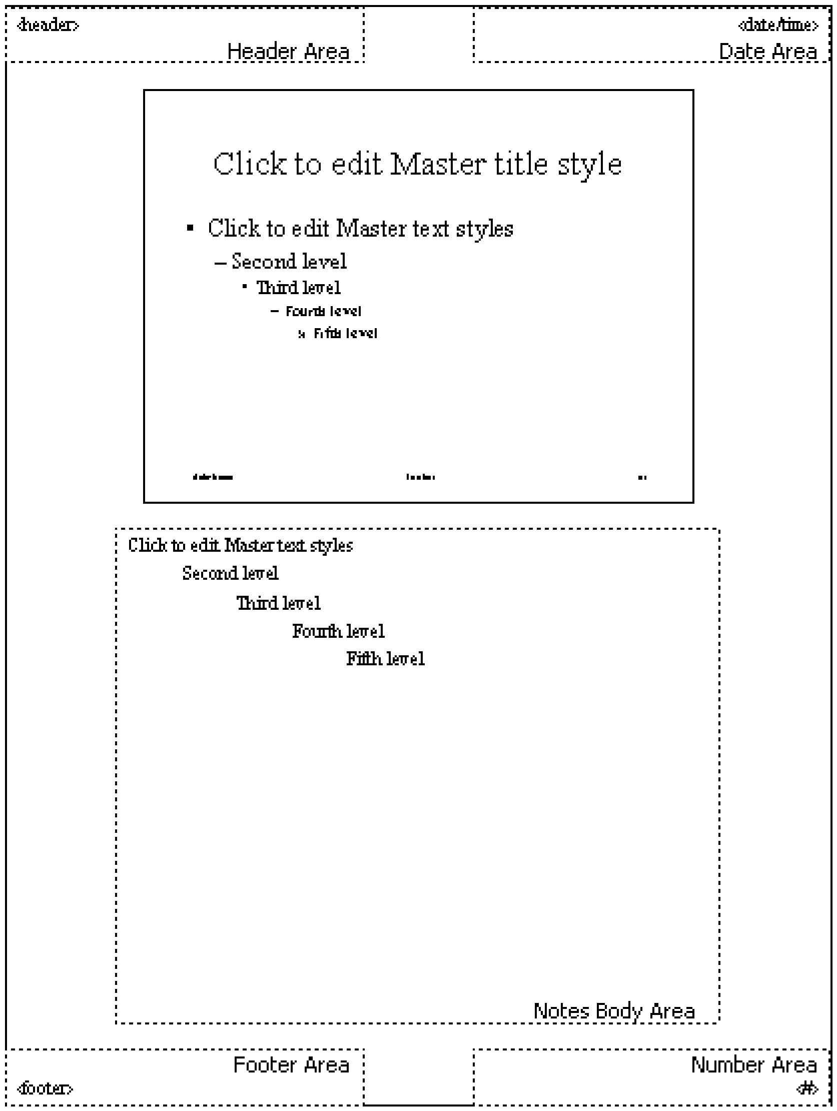


***Figure 11**. The notes master page, as seen in PowerPoint. The Notes Placeholders are ordered in a left to right, top to bottom order.*

Notes can be added to each slide using the NotesPage object. The NotesPage object contains a collection of two Shape objects: one for the slide picture, and the second for the text. When viewing the notes master, these two shapes change on each page, while the others remain constant (the slide number can be set to a field that calculates the slide number). The slide image has an index of 1, while the text has an index of 2 (again, there are no Microsoft-issue constants). The following lines of code add sample speaker note text for slides 1 and 2:

```
WITH oPresentation
  .Slides[1].NotesPage.Shapes.Placeholders[2].TextFrame.TextRange.Text = ;
     "Remember to welcome the audience." + CHR(13) + "Introduce the company."
  .Slides[2].NotesPage.Shapes.Placeholders[2].TextFrame.TextRange.Text =  ;
     "Explain the marketing slogan."
ENDWITH
```

You can format notes just like any other text. See the section “Adding and formatting text” in Chapter 10, as well as “Standardizing the appearance of text” earlier in this chapter, for additional information on formatting the text.

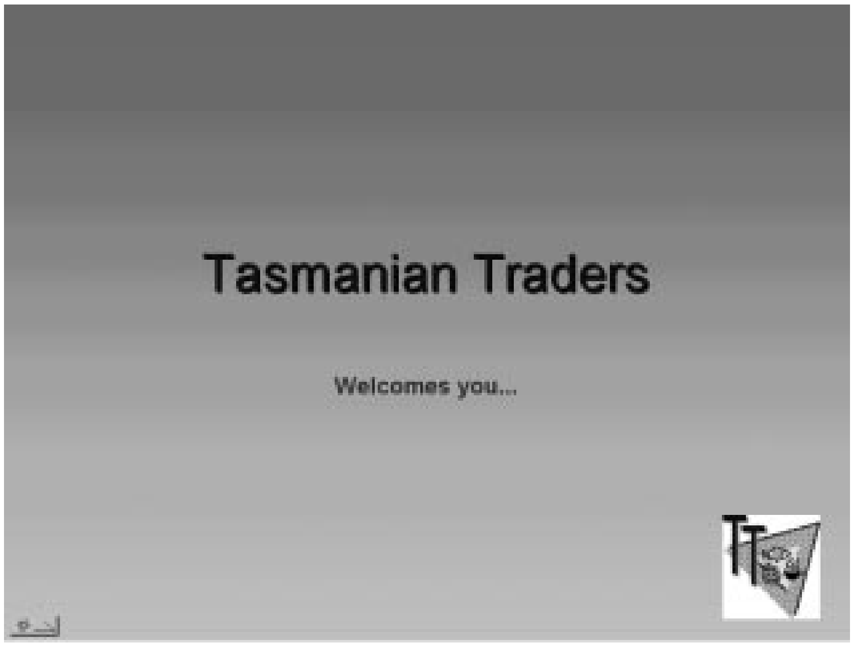
Putting it all together

In Chapter 10, “PowerPoint Basics,” we put together a little slide show demonstrating the topics we covered. In this chapter, we use the same basic slides, and add many of the additional features we covered. **Listing 1** (PPTSample2.PRG in the Developer Download files available at www.hentzenwerke.com) shows the code, while **Figure 12**, **Figure 13**, and **Figure 14** show the finished slides. Note that with the addition of color, sound, and animation, this code must be run to see the presentation. Black and white screen shots cannot show all the features.

***Listing 1**. A sample presentation for Tasmanian Traders.*

```
CLOSE DATA
#DEFINE ppTitleStyle             2
#DEFINE ppBodyStyle              3
#DEFINE ppLayoutTitle            1
#DEFINE ppLayoutText             2
#DEFINE ppEffectCoverDown      1284
#DEFINE ppEffectDissolve       1537
#DEFINE ppEffectAppear         3844
#DEFINE ppAnimateByFirstLevel    1
#DEFINE ppAdvanceOnTime          2
#DEFINE msoGradientHorizontal    1
#DEFINE ppBulletArabicPeriod     3
#DEFINE msoLineThickBetweenThin   5
#DEFINE autoIn2Pts              72

#DEFINE rgbDarkBlue    RGB(  0,   0, 138)
#DEFINE rgbMediumBlue  RGB( 96,  96, 204)
#DEFINE rgbPaleGray    RGB(192, 192, 192)
```

```
#DEFINE rgbYellow      RGB(255, 255,   0)
#DEFINE rgbBurgundy    RGB(128,   0,   0)
#DEFINE rgbLineColor   RGB(255, 255,   0)

SET PATH TO (_SAMPLES + "\TasTrade\Data") ADDITIVE

******************
* Set up the data
******************
OPEN DATABASE TasTrade
LogoFile = (_SAMPLES + "\TasTrade\Bitmaps\TTradeSm.bmp")

USE OrdItems IN 0
USE Products IN 0

* Select the top 5 selling items of all time
SELECT TOP 5 ;
       P.English_Name, ;
       SUM(O.Unit_Price * O.Quantity) AS TotQuan ;
  FROM OrdItems O, Products P ;
 WHERE O.Product_ID = P.Product_ID ;
 GROUP BY 1 ;
 ORDER BY 2 DESC;
  INTO CURSOR TopSellers

* Select the number of products
SELECT Count(*) ;
  FROM Products ;
  INTO ARRAY aProducts

* Clean out any existing references to servers.
* This prevents memory loss to leftover instances.
RELEASE ALL LIKE o*

* For demonstration purposes, make oPowerPoint and oPresentation
* available after this program executes.
PUBLIC oPowerPoint, oPresentation

* Open PowerPoint
oPowerPoint = CreateObject("PowerPoint.Application")
oPowerPoint.Visible = .T.

* Create the presentation
oPresentation = oPowerPoint.Presentations.Add()

SlideNum = 1

******************
* MASTER SLIDE
******************
WITH oPresentation.SlideMaster

  * Set the background to a gradient fill
  WITH .Background.Fill
    .ForeColor.RGB = rgbMediumBlue
    .BackColor.RGB = rgbPaleGray
    .TwoColorGradient(msoGradientHorizontal, 1)
  ENDWITH

  * Set titles
  WITH .TextStyles[ppTitleStyle].Levels[1].Font
    .Name = "Arial"
    .Shadow = .T.
    .Color.RGB = rgbDarkBlue
  ENDWITH

  * Set Body Style levels
  WITH .TextStyles[ppBodyStyle]
    WITH .Levels[1]
      WITH .Font
        .Name      = "Arial"
        .Bold      = .T.
        .Color.RGB = rgbBurgundy
        .Size      = 20
      ENDWITH
      WITH .ParagraphFormat.Bullet
        * Set bullet character to a dot (use 149 if using PowerPoint 97)
        .Character = 8226
        .Visible   = .F.    && Don't show the bullet!
      ENDWITH
    ENDWITH
    WITH .Levels[2]
      WITH .Font
        .Name      = "Arial"
        .Color.RGB = rgbDarkBlue
        .Size      = 20
      ENDWITH
     * Set bullet character to a dot (use 149 if using PowerPoint 97)
     .ParagraphFormat.Bullet.Character = 8226

    ENDWITH

  ENDWITH

  * Add the logo.
  .Shapes.AddPicture(LogoFile, .F., .T., ;
        8.5 * autoIn2Pts, 6.0 * autoIn2Pts)

  * PowerPoint 97 users: the last two parameters,
  * height and width, are not optional. Use this
  * code instead:
  *.Shapes.AddPicture(LogoFile, .F., .T., ;
  *      8.5 * autoIn2Pts, 6.0 * autoIn2Pts, ;
  *      1.0 * autoIn2Pts, 1.0 * autoIn2Pts)

ENDWITH
```

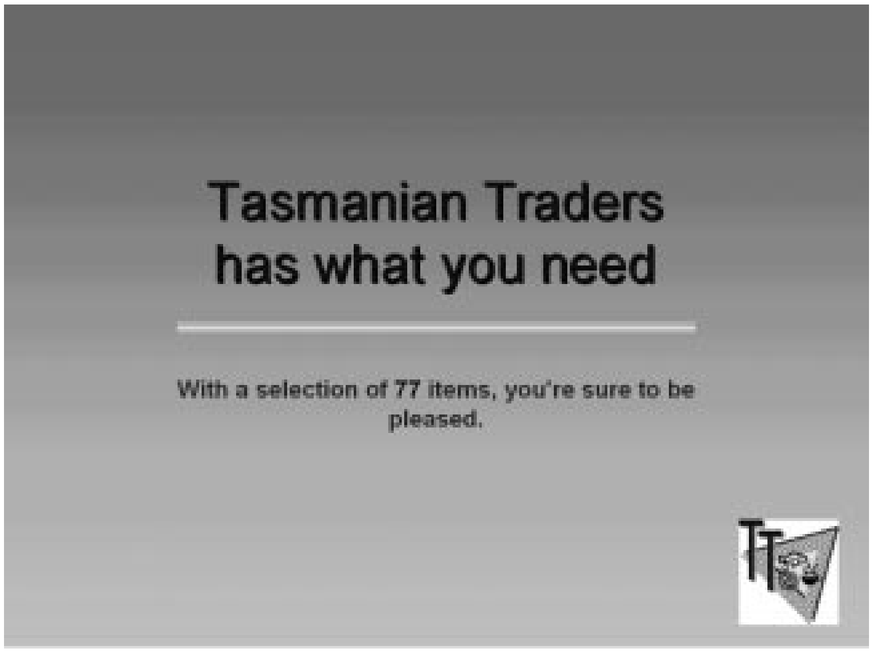


Tasmanian Traders

Welcomes you...

***Figure 12**. The Tasmanian Traders sample title slide. The gradient fill background, title fonts and colors, and the logo on each slide are done through the Master Slide.*

```
******************
* TITLE SLIDE
******************
* Add the slide
oSlide = oPresentation.Slides.Add(SlideNum, ppLayoutTitle)

* Set the title text.
oSlide.Shapes[1].TextFrame.TextRange.Text = "Tasmanian Traders"

* Set the subtitle text
oSlide.Shapes[2].TextFrame.TextRange.Text = "Welcomes you..."

WITH oSlide.SlideShowTransition
  .EntryEffect = ppEffectCoverDown
  .AdvanceOnTime = .T.
  .AdvanceTime   = 2
ENDWITH

SlideNum = SlideNum + 1
```

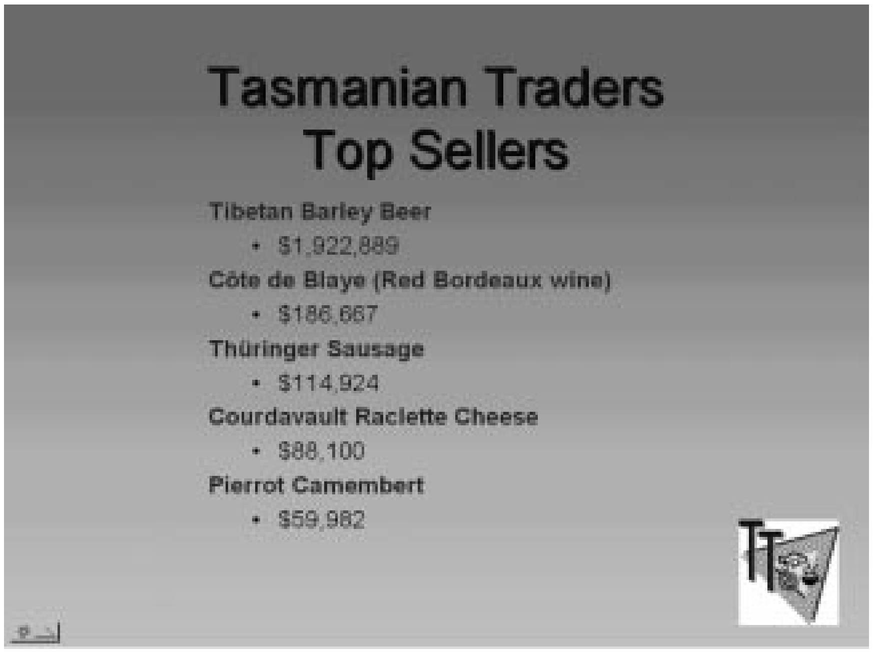

Tasmanian Traders has what you need

With a selection of 77 items, you're sure to be pleased.

***Figure 13**. The Tasmanian Traders sample second slide. As with the first slide, there is minimal formatting in the code, because the SlideMaster takes care of the majority of the formatting.*

```
******************
* SECOND SLIDE
******************

* Add the slide
oSlide = oPresentation.Slides.Add(SlideNum, ppLayoutTitle)

* Bring the slide to the front
oSlide.Select()

* PowerPoint 97 users: oSlide.Select() will
* generate an error. Use this line instead:
* oPresentation.ActiveWindow.View.GoToSlide(2)

* Set the text of the title
oSlide.Shapes[1].TextFrame.TextRange.Text = "Tasmanian Traders " + CHR(13) + ;
      "has what you need"

* Move the title up about half an inch
WITH oSlide.Shapes[1]
  .Top = .Top - (.5 * autoIn2Pts)
ENDWITH

* Add a line half an inch below the title that is centered and 6" long
LineTop = oSlide.Shapes[1].Top + oSlide.Shapes[1].Height + ;
     (.5 * autoIn2Pts)
LineLeft = 2 * autoIn2Pts
```

```
LineEnd = LineLeft + (6.0 * autoIn2Pts)

oLine = oSlide.Shapes.AddLine(LineLeft, LineTop, LineEnd, LineTop)

* Format the line to be a thick line
* with two thin lines on either side
WITH oLine.Line
  .ForeColor.RGB = rgbLineColor
  .Style = msoLineThickBetweenThin
  .Weight = 8
ENDWITH

* Set the text of the subtitle, and change the number to bold and blue
WITH oSlide.Shapes[2].TextFrame.TextRange
  .Text = "With a selection of " + ALLTRIM(STR(aProducts[1])) + ;
          " items, you're sure to be pleased."
  .Words[5].Font.Color = rgbDarkBlue
ENDWITH

WITH oSlide.SlideShowTransition
  .EntryEffect = ppEffectDissolve
  .AdvanceOnTime = .T.
  .AdvanceTime   = 5
ENDWITH

SlideNum = SlideNum + 1
```



Tasmanian Traders Top Sellers

Tibetan Barley Beer
- $1,922,889

Côte de Blaye (Red Bordeaux wine)
- $186,667

Thüringer Sausage
- $114,924

Courdavault Raclette Cheese
- $88,100

Pierrot Camembert
- $59,982

***Figure 14**. The Tasmanian Traders sample third slide. Again, the bulk of the formatting is done in the SlideMaster. Fonts, colors, and backgrounds are extremely consistent, and your code is easy to maintain.*

```
***********************
* TOP SELLERS SLIDE
***********************
* Add the slide
oSlide = oPresentation.Slides.Add(SlideNum, ppLayoutText)

* Bring the slide to the front
oSlide.Select()

* PowerPoint 97 users: oSlide.Select() will
* generate an error. Use this line instead:
* oPresentation.ActiveWindow.View.GoToSlide(3)

* Insert the title (note the use of the Title object, instead of
* an enumerated shape object).
oSlide.Shapes.Title.TextFrame.TextRange.Text = "Tasmanian Traders" + ;
     CHR(13) + "Top Sellers"

* Build the string to use for the top 5 sellers.
* Use a CR between each item to make each a separate bullet.
BulletString = ""
SELECT TopSellers
SCAN
  BulletString = BulletString + ;
                 TRIM(TopSellers.English_Name) + CHR(13) + "$" + ;
                 ALLTRIM(TRANSFORM(TopSellers.TotQuan, "99,999,999")) + ;
                 CHR(13)
ENDSCAN

* Add the bullet string to the text frame.
WITH oSlide.Shapes[2]
  WITH .TextFrame.TextRange
    .Text = BulletString

    * Indent all the sales quotas
    FOR I = 1 TO 5
      .Lines[I*2, 1].IndentLevel = 2
    ENDFOR
  ENDWITH

  * Move it to the right about 1.5"
  .Left = .Left + (1.5 * autoIn2Pts)

  * Each bullet (with subordinates) appears at 1 second intervals
  WITH .AnimationSettings
    .EntryEffect = ppEffectAppear
    .TextLevelEffect = ppAnimateByFirstLevel
    .AdvanceMode = ppAdvanceOnTime
    .AdvanceTime   = 0.5
    .SoundEffect.Name = "Whoosh"
  ENDWITH
ENDWITH

oSlide.SlideShowTransition.EntryEffect = ppEffectDissolve

* Run the slideshow.
oPresentation.SlideShowSettings.Run()
```

This chapter explored the visual world of PowerPoint. We've covered many of the commonly used features of PowerPoint. There is a lot more to PowerPoint, though, so don't hesitate to use that macro recorder, the Object Browser, and the Help files to find those features that your application needs.