Chapter 15
Draw and Impress Documents

This chapter introduces methods to manipulate and modify the content contained in Draw and Impress documents. The drawing functionality is the same in Draw and Impress, but Impress contains extra functionality to facilitate presentations.

Draw and Impress are vector-oriented graphical applications. They can also display bitmap images, but their strength is not in photo editing. Vector-oriented applications represent many graphical objects as objects rather than as bitmapped images. For example, lines, circles, rectangles, and text are each represented as special objects. One advantage of vector graphics is that you can independently manipulate and transform multiple overlapping elements without worrying about resolution and pixels.

Photo-editing graphics packages typically represent and manipulate images as bitmaps. A bitmap representing an image is characterized by the width and height of the image in pixels. Each pixel represents a single, colored dot in the image. The drawing capability in OpenOffice.org, however, is focused on vector operations.

Every Draw document supports the com.sun.star.drawing.DrawingDocument service and every Impress document supports the com.sun.star.presentation.PresentationDocument service. When I write a macro that must be user friendly and requires a specific document type, I verify that the document is the correct type by using the object method supportsService(). See **Listing 1**.

***Listing 1**. Check for an Impress document before checking for a Draw document.*

```
REM If it really matters, you should check the document type
REM to avoid a run-time error.
sDraw$ = "com.sun.star.drawing.DrawingDocument"
sImpress$ = "com.sun.star.presentation.PresentationDocument"
If ThisComponent.SupportsService(sImpress$) Then
  MsgBox "The current document is an Impress document", 0, "Impress Document"
ElseIf ThisComponent.SupportsService(sDraw$) Then
  MsgBox "The current document is a Draw document", 0, "Draw Document"
Else
  MsgBox "The current document is not the correct type", 48, "Error"
  Exit Sub
End If
```

The PresentationDocument service implements the DrawingDocument service. This means that every presentation document looks like a drawing document. To distinguish between the two document types, you must first check for a presentation (Impress) document and then check for a drawing document.

Draw pages

The primary function of Draw and Impress documents is to contain graphical data, which is stored in draw pages. The primary draw-page functionality is implemented by a GenericDrawPage. There are two types of draw pages—the MasterPage and the DrawPage. Both draw-page types implement the GenericDrawPage and are therefore able to hold and manipulate the same content.

A master page acts as a common background for zero or more draw pages. Each draw page may link to one master page. Each master page is constrained as follows:

- A master page, unlike a regular draw page, may not link to a master page.
- A master page may not be removed from a document if any draw page links to it.
- Modifications made to a master page are immediately visible on every draw page that uses that master page.

The method getMasterPages() returns the document's collection of master pages. The method getDrawPages() returns the document's collection of draw pages. Both methods return the same object type; they differ only in how their contents are used (see **Table 1**).

***Table 1**. Methods supported by the com.sun.star.drawing.XDrawPages interface.*

Method	Description
insertNewByIndex(Long)	Create and insert a new draw page at the specified index.
hasByName(String)	Return True if the named page exists.
hasElements()	Return True if any draw pages exist.
remove(DrawPage)	Remove a specific draw page.
getCount()	Return the number of contained objects as a Long Integer.
getByIndex(Long)	Get the specified draw page.
getByName(String)	Get the specified draw page.
duplicate(DrawPage)	Duplicate the DrawPage and return the new DrawPage.
hasElements()	Return True if there are draw pages to return.

Each draw page has a name, which you can access by using the methods getName() and setName(). A draw page that is not a master supports the methods getMasterPage() and setMasterPage() to get and set the master page.**Listing 2** demonstrates enumerating the document's draw pages and their corresponding master pages.

***Listing 2**. getPages is found in the Graphic module in this chapter's source code files as SC15.sxi.*

```
Sub getPages()
  Dim s$
  s = s & getPagesInfo(ThisComponent.getDrawPages(), "Draw Pages")
  s = s & CHR$(10)
  s = s & getPagesInfo(ThisComponent.getMasterPages(), "Master Pages")
  MsgBox s, 0, "Pages"
End Sub

Function getPagesInfo(oDPages, sType$) As String
  Dim i%, s$
  Dim oDPage, oMPage
  s = s & "*** There are " & oDPages.getCount() & " " & sType & CHR$(10)
  For i = 0 To oDPages.getCount()-1
    oDPage = oDPages.getByIndex(i)
    s = s & "Page " & i & " = '" & oDPage.getName() & "'"
    If NOT oDPage.supportsService("com.sun.star.drawing.MasterPage") Then
      oMPage = oDPage.getMasterPage()
      s = s & " master = "
      If NOT IsNull(oMPage) AND NOT IsEmpty(oMPage) Then
        s = s & "'" & oMPage.getName() & "'"
      End If
    End If
    s = s & CHR$(10)
  Next
  getPagesInfo = s
End Function
```

Although you can get a draw page based on its name, you can have more than one draw page with the same name. If multiple draw pages use the same name and you retrieve the draw page based on the name, it is not specified which draw page is returned. The macro in **Listing 3**, which searches for a draw page with a specific name, is used in numerous places in this chapter.

***Listing 3**. createDrawPage is found in the Graphic module in this chapter's source code files as SC15.sxi.*

```
Function createDrawPage(oDoc, sName$, bForceNew As boolean) As Variant
  Dim oPages 'All of the draw pages
  Dim oPage  'A single draw page
  Dim i%     'General index variable
  oPages = oDoc.getDrawPages()
  If oPages.hasByName(sName) Then
    REM If we require a new page then delete
    REM the page and get out of the for loop.
    If bForceNew Then
      oPages.remove(oPages.getByName(sName))
```

```
    Else
      REM Did not request a new page so return the found page
      REM and then get out of the function.
      createDrawPage = oPages.getByName(sName)
      Exit Function
    End If
  End If

  REM Did not find the page, or found the page and removed it.
  REM Create a new page, set the name, and return the page.
  oPages.insertNewByIndex(oPages.getCount())
  oPage = oPages.getByIndex(oPages.getCount()-1)
  oPage.setName(sName)
  createDrawPage = oPage
End Function
```

Generic draw page

Both regular and master draw pages support the GenericDrawPage service. As its name implies, the GenericDrawPage service provides the primary generic drawing functionality. Writer and Calc documents provide specific styles for formatting entire pages. Draw pages, however, use the properties shown in **Table 2**.

***Table 2**. Properties defined by the com.sun.star.drawing.GenericDrawPage service.*

Property	Description
BorderBottom	Bottom border in 1/100 mm, represented as a Long Integer.
BorderLeft	Left border in 1/100 mm, represented as a Long Integer.
BorderRight	Right border in 1/100 mm, represented as a Long Integer.
BorderTop	Top border in 1/100 mm, represented as a Long Integer.
Height	Height in 1/100 mm, represented as a Long Integer.
Width	Width in 1/100 mm, represented as a Long Integer.
Number	Draw page number as a Short Integer. This read-only value labels the first page 1.
Orientation	Page orientation as a com.sun.star.view.PaperOrientation enumeration. Two values are supported—PORTRAIT and LANDSCAPE.
UserDefinedAttributes	User-defined XML attributes.
IsBackgroundDark	True if the averaged background fill color's luminance is below a specified threshold value.

The primary purpose of a draw page is to contain shapes. The methods addShape(Shape) and removeShape(Shape) add and remove a shape from a document. Before a shape can be added to a draw page, it must be created by the document. Each line produced by **Listing 4** and shown in **Figure 1** is a separate, unrelated shape. You can indepently manipulate each of these shapes.

***Listing 4**. drawFirstGraphic is found in the Graphic module in this chapter's source code files as SC15.sxi.*

```
Sub drawFirstGraphic()
  Dim oPage  'Page on which to draw
  Dim oShape 'Shape to insert
  Dim oPoint 'Initial start point of the line
  Dim oSize  'Width and height of the line
  Dim i%     'Index variable
  Dim n%     'Number of iterations to perform
  oPage = createDrawPage(ThisComponent, "Test Draw", True)
  n = 20
```

```
  For i = 0 To n
    oShape = ThisComponent.createInstance("com.sun.star.drawing.LineShape")
    oShape.LineColor = RGB( 255, 0, i+20 )
    oShape.LineWidth = 20
    oPoint = oShape.Position
    oPoint.X = oPage.Width / 4
    oPoint.Y = i * oPage.Height / n / 4
    oShape.Position = oPoint
    oSize = oShape.Size
    oSize.Height = (oPage.Height - 2 * i * oPage.Height / n) / 4
    oSize.Width = oPage.Width / 2
    oShape.Size = oSize
    oPage.add(oShape)
  Next
End Sub
```

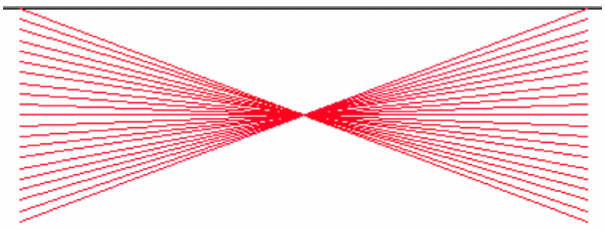

***Figure 1**. Twenty lines in an Impress document.*

Combining shapes

The macro in Listing 4 creates 20 independent lines. You can group shapes together and then manipulate them as a single shape. The method group(XShapes) accepts a collection of shapes and turns them into a single group; an XShapeObject is returned. The macro in **Listing 5** starts by calling Listing 4 and then it adds all of the lines to a single group.

***Listing 5**. groupShapes is found in the Graphic module in this chapter's source code files as SC15.sxi.*

```
Sub groupShapes
  Dim oPage   'Page on which to draw
  Dim oShapes 'Shapes to group
  Dim i%      'Index variable

  REM Create the shapes!
  drawFirstGraphic()
  oPage = ThisComponent.getDrawPages().getByName("Test Draw")

  REM Create a shape collection object to group the shapes
  oShapes = createUnoService("com.sun.star.drawing.ShapeCollection")
  For i = 0 To oPage.getCount()-1
    oShapes.add(oPage.getByIndex(i))
  Next
  oPage.group(oShapes)
End Sub
```

When several shapes are grouped together using the group() method, the entire group is added as a single shape into the draw page, which you can retrieve by using oPage.getByIndex(). It's no longer possible to select a single line and independently manipulate it. To convert a group of shapes back to independent shapes, use the ungroup(XShapeObject) method. The ungroup() method removes the objects from the group and adds them back to the draw page as individual objects (see **Listing 6**).

***Listing 6**. unGroupShapes is found in the Graphic module in this chapter's source code files as SC15.sxi.*

```
Sub unGroupShapes
  Dim oPage   'Page on which to draw
  Dim oShape  'Single shape
  Dim i%      'Index variable
  oPage = ThisComponent.getDrawPages().getByName("Test Draw")
```

```
  For i = 0 To oPage.getCount()-1
    oShape = oPage.getByIndex(i)
    If oShape.supportsService("com.sun.star.drawing.GroupShape") Then
      oPage.ungroup(oShape)
    End If
  Next
End Sub
```

Although shapes that are grouped together are manipulated as one shape, they are really a collection of shapes. The combine(XShapes) method, on the other hand, converts each shape into a PolyPolygonBezierShape and then combines them into one PolyPolygonBezierShape. See **Listing 7**. The new shape is added to the draw page, and the original shapes are removed and deleted. The split(XShape) method converts the shape to a PolyPolygonBezierShape (if it is not already) and then the shape is split into several shapes of type PolyPolygonBezierShape. The new shapes are added to the draw page and the original shape is removed and deleted.

***Listing 7**. combineShapes is found in the Graphic module in this chapter's source code files as SC15.sxi.*

```
Sub combineShapes
  Dim oPage, oShapes, i%

  REM Create the shapes!
  drawFirstGraphic()
  oPage = ThisComponent.getDrawPages().getByName("Test Draw")

  oShapes = createUnoService("com.sun.star.drawing.ShapeCollection")
  For i = 0 To oPage.getCount()-1
    oShapes.add(oPage.getByIndex(i))
  Next
  oPage.combine(oShapes)
End Sub
```

The bind(XShapes) method is similar to combine() except that the individual shapes are connected before they are combined. The output from Listing 7, therefore, is the same as Figure 1. **Listing 8**, on the other hand, produces the output in **Figure 2**. The shapes are connected by a line, each of which is really a Bezier curve.

***Listing 8**. bindShapes is found in the Graphic module in this chapter's source code files as SC15.sxi.*

```
Sub bindShapes()
  Dim oPage, oShapes, i%

  REM Create the shapes!
  drawFirstGraphic()
  oPage = ThisComponent.getDrawPages().getByName("Test Draw")

  oShapes = createUnoService("com.sun.star.drawing.ShapeCollection")
  For i = 0 To oPage.getCount()-1
    oShapes.add(oPage.getByIndex(i))
  Next
  oPage.bind(oShapes)
End Sub
```

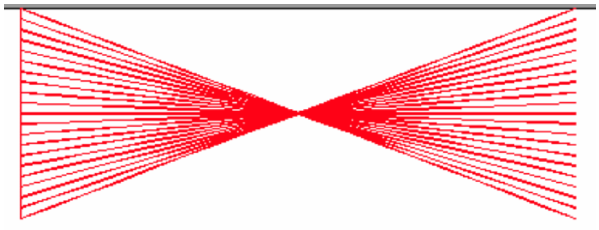

***Figure 2**. Twenty lines bound together are connected with a line.*

The unbind(XShape) method converts the shape to a PolyPolygonBezierShape (if it is not already) and then each line segment is converted to a new PolyPolygonBezierShape. The original shape is removed from the draw page and deleted. If unbind() is used on the shape in Figure 2, the output still looks like Figure 2, but 776 shapes are used.

The methods group() and ungroup() act like "undo" for each other. The methods bind() and combine() are not "undo" for unbind() and split(), primarily because each shape is converted to a PolyPolygonBezierShape.

Shapes

Graphical content is expressed as a shape object. Shape objects are created by the document and then added to the draw page. OOo supports numerous types of shapes (see **Table 3**).

***Table 3**. Shape types supported by OOo.*

ShapeType	Description
ClosedBezierShape	A series of Bezier shapes that are closed.
ConnectorShape	Used to connect shapes or glue points.
ControlShape	A shape that displays a control such as a button.
EllipseShape	Draw a circle, an ellipse, or an arc.
GraphicObjectShape	Display a graphic object such as a bitmap image. There are separate types for presentation documents and drawing documents.
GroupShape	Represent multiple shapes as a single shape.
LineShape	A single line.
MeasureShape	A shape used for measuring in a diagram.
OLE2Shape	Display an OLE object in a presentation document. There are separate types for presentation documents and drawing documents.
OpenBezierShape	A series of Bezier lines.
PageShape	Display a preview of another page. There are separate types for presentation documents and drawing documents.
PolyLineShape	A series of connected straight lines.
PolyPolygonBezierShape	A polygon using Bezier curves.
PolyPolygonShape	A series of straight lines with the first and last points connected.
RectangleShape	Draw rectangles.
TextShape	Box designed to hold text.
PluginShape	Represent a media type that is not directly supported.
TitleTextShape	A TextShape for titles in a presentation document.
SubtitleShape	A TextShape for subtitles in a presentation document.
OutlinerShape	A TextShape for outlines in a presentation document.
ChartShape	Chart shape in a presentation document.
NotesShape	A TextShape for notes in a presentation document.
HandoutShape	A drawing document PageShape for handouts in a presentation document.

A shape's position is stored in a com.sun.star.awt.Point structure, which contains two Long Integer values, X and Y, representing the upper-left corner in 1/100 mm. A shape's size is stored in a com.sun.star.awt.Size structure, which contains two Long Integer values, Width and Height, in 1/100 mm. See **Table 4**.

***Table 4**. Methods supported by Shape objects.*

Method	Description
getPosition()	Get the shape's current position in 1/100 mm.
setPosition(Point)	Set the shape's current position in 1/100 mm.
getSize()	Get the shape's current size in 1/100 mm.
setSize(Size)	Set the shape's current size in 1/100 mm.
getGluePoints()	Get an object that provides indexed access to a set of glue points used internally by the object. Each glue point is a com.sun.star.drawing.GluePoint2 structure (see Table 16).
getShapeType()	String representing the shape's type.

Macros that deal with shape objects frequently require the Size and Point structures. The two methods in **Listing 9** make it easier to create and set these structures.

***Listing 9**. CreatePoint and CreateSize are found in the Graphic module in this chapter's source code files as SC15.sxi.*

```
Function CreatePoint(ByVal x As Long,ByVal y As Long) As com.sun.star.awt.Point
  Dim oPoint
  oPoint = createUnoStruct( "com.sun.star.awt.Point" )
  oPoint.X = x : oPoint.Y = y
  CreatePoint = oPoint
End Function

Function CreateSize(ByVal x As Long,ByVal y As Long) As com.sun.star.awt.Size
  Dim oSize
  oSize = createUnoStruct( "com.sun.star.awt.Size" )
  oSize.Width = x : oSize.Height = y
  CreateSize = oSize
End Function
```

Common properties

Interfaces define methods and can be derived from other interfaces. Services, on the other hand, implement interfaces and other services. Services also define properties. Some services are defined strictly to define a group of related properties. The properties defined by the Shape service are general and applicable to most shape types (see **Table 5**).

***Table 5**. Properties defined by the com.sun.star.drawing.Shape service.*

Property	Description
ZOrder	Long Integer representing the ZOrder of this shape. This controls the drawing order of objects, effectively moving an object forward or backward.
LayerID	Short Integer identifying the layer that contains the shape.
LayerName	Name of the layer that contains the shape.
Printable	If True, the shape is included in printed output.
MoveProtect	If True, the shape cannot be moved interactively by the user.
Name	Shape name as a String.
SizeProtect	If True, the user may not change the shape's size.
Style	Shape's style as a com.sun.star.style.XStyle object.
Transformation	Transformation matrix of type com.sun.star.drawing.HomogenMatrix3 that can contain translation, rotation, shearing, and scaling.

OOo defines separate services that encapsulate properties and methods specific to lines, text, shadows, shape rotation, and filling area. Not all shape types support all of these services. For example, it makes no sense for a line shape to support the properties and methods related to filling areas. **Table 6** provides a quick overview of the special services supported by each shape type.

***Table 6**. Which shapes support which service.*

ShapeType	Text	Line	Fill	Shadow	Rotation
ClosedBezierShape	x	x	x	x	x
ConnectorShape	x	x		x	x
ControlShape					
EllipseShape	x	x	x	x	x
GraphicObjectShape	x			x	x
GroupShape					
LineShape	x	x		x	x
MeasureShape	x	x		x	x
OLE2Shape					
OpenBezierShape	x	x		x	x
PageShape					
PolyLineShape	x	x		x	x
PolyPolygonBezierShape	x	x	x	x	x
PolyPolygonShape	x	x	x	x	x
RectangleShape	x	x	x	x	x
TextShape	x	x	x	x	x
TitleTextShape	x	x	x	x	x
SubtitleShape	x	x	x	x	x
OutlinerShape	x	x	x	x	x
ChartShape					
NotesShape	x	x	x	x	x
HandoutShape					

Drawing text service

Any shape that supports the com.sun.star.drawing.Text service has the ability to contain text. The drawing text service supports the standard com.sun.star.text.XText interface and a special set of drawing text properties. Besides character and paragraph properties, the drawing text properties service defines properties specifically designed for shape objects (see **Table 7**).

***Table 7**. Properties defined by the com.sun.star.drawing.TextProperties service.*

Property	Description
IsNumbering	If True, numbering is ON for the text in this shape.
NumberingRules	Describes the numbering levels as a sequence of com.sun.star.style.NumberingRule.
TextAutoGrowHeight	If True, the shape height changes automatically when text is added or removed.
TextAutoGrowWidth	If True, the shape width changes automatically when text is added or removed.
TextContourFrame	If True, the text is aligned with the left edge of the shape.
TextFitToSize	Enumerated value of type com.sun.star.drawing.TextFitToSizeType: • NONE – The text size is defined by the font properties. • PROPORTIONAL – Scale the text if the shape is scaled. • ALLLINES – Like PROPORTIONAL, but the width of each row is also scaled. • RESIZEATTR – If the shape is scaled, scale the font attributes.

Property	Description
TextHorizontalAdjust	Enumerated value of type com.sun.star.drawing.TextHorizontalAdjust: • LEFT – The left edge of the text is adjusted to the left edge of the shape. • CENTER – The text is centered inside the shape. • RIGHT – The right edge of the text is adjusted to the right edge of the shape. • BLOCK – The text extends from the left to the right edge of the shape.
TextVerticalAdjust	Enumerated value of type com.sun.star.drawing.TextVerticalAdjust: • TOP – The top edge of the text is adjusted to the top edge of the shape. • CENTER – The text is centered inside the shape. • BOTTOM – The bottom edge of the text is adjusted to the bottom edge of the shape. • BLOCK – The text extends from the top to the bottom edge of the shape.
TextLeftDistance	Distance from the left edge of the shape to the text as a Long Integer.
TextRightDistance	Distance from the right edge of the shape to the text as a Long Integer.
TextUpperDistance	Distance from the upper edge of the shape to the text as a Long Integer.
TextLowerDistance	Distance from the lower edge of the shape to the text as a Long Integer.
TextMaximumFrameHeight	Limit a shape's height as it grows automatically as you enter text.
TextMaximumFrameWidth	Limit a shape's width as it grows automatically as you enter text.
TextMinimumFrameHeight	Limit a shape's minimum height as it grows automatically as you enter text.
TextMinimumFrameWidth	Limit a shape's minimum width as it grows automatically as you enter text.
TextAnimationAmount	Number of pixels the text is moved in each animation step.
TextAnimationCount	Number of times the text animation is repeated.
TextAnimationDelay	Delay, in thousandths of a second, between each animation step.
TextAnimationDirection	Enumerated value of type com.sun.star.drawing.TextAnimationDirection: LEFT, RIGHT, UP, and DOWN.
TextAnimationKind	Enumerated value of type com.sun.star.drawing.TextAnimationKind: • NONE – No animation. • BLINK – Continuously switch the text between visible and invisible. • SCROLL – Scroll the text. • ALTERNATE – Scroll the text from one side to the other and back. • SLIDE – Scroll the text from one side to the final position and stop.
TextAnimationStartInside	If True, the text is visible at the start of the animation.
TextAnimationStopInside	If True, the text is visible at the end of the animation.
TextWritingMode	Enumerated value of type com.sun.star.text.TextWritingMode: • LR_TB – Text is written left to right and top to bottom. • RL_TB – Text is written right to left and top to bottom. • TB_RL – Text is written top to bottom and lines are placed right to left.

The default behavior of a MeasureShape (see **Figure 3**) is to display the actual length of the shape. The macro in **Listing 10** creates two measure shapes and changes the text of one of them to "Width". To help illustrate setting the properties in Table 7, the TextAnimationKind property is set to SCROLL so that the text continuously scrolls from right to left.

***Listing 10**. drawMeasureShape is found in the Graphic module in this chapter's source code files as SC15.sxi.*

```
Sub drawMeasureShape()
  Dim oPage  'Page on which to draw
  Dim oShape 'Shape to insert
  Dim oStart As new com.sun.star.awt.Point
  Dim oEnd As new com.sun.star.awt.Point

  oPage = createDrawPage(ThisComponent, "Test Draw", True)
  oShape = ThisComponent.createInstance("com.sun.star.drawing.MeasureShape")
  oPage.add(oShape)
  REM The following values MUST be set AFTER the object is inserted.
```

```
  oStart.X = oPage.Width / 4 : oEnd.X = oPage.Width / 2
  oStart.Y = oPage.Height/4  : oEnd.Y = oPage.Height/4
  oShape.StartPosition = oStart
  oShape.EndPosition = oEnd
  oShape.setString("Width")
  oShape.TextAnimationKind = com.sun.star.drawing.TextAnimationKind.SCROLL

  oShape = ThisComponent.createInstance("com.sun.star.drawing.MeasureShape")
  oPage.add(oShape)
  oStart.X = oPage.Width / 5 : oEnd.X = oPage.Width / 5
  oStart.Y = oPage.Height/4  : oEnd.Y = oPage.Height/2.5
  oShape.StartPosition = oStart
  oShape.EndPosition = oEnd
End Sub
```

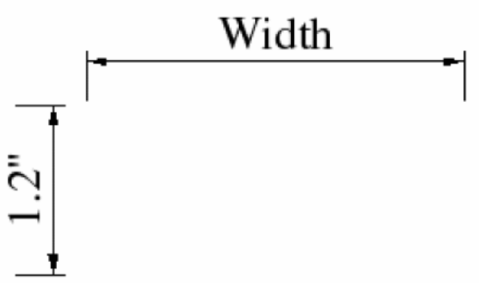

Figure 3. *By default, measure shapes show the actual size—you can override this.*

Drawing line properties

Shapes that support the com.sun.star.drawing.LineProperties service can influence how lines are drawn. Most shapes support line properties because most shapes contain lines of some sort. The specific properties dealing with line endpoints and start points are supported only by shapes with open ends. See **Table 8**.

Table 8*. Properties defined by the com.sun.star.drawing.LineProperties service.*

Property	Description
LineStyle	Enumerated value of type com.sun.star.drawing.LineStyle: NONE (the line is hidden), SOLID, and DASH.
LineDash	Enumerated value of type com.sun.star.drawing.LineDash that defines how dashed lines are drawn. • Style – Enumerated value of type com.sun.star.drawing.DashStyle: RECT, ROUND, RECTRELATIVE, and ROUNDRELATIVE. • Dots – Number of dots in this LineDash as a Long Integer. • DotLen – Length of a dot as a Long Integer. • Dashes – Number of dashes as a Short Integer. • DashLen – Length of a single dash as a Long Integer. • Distance – Distance between the dots as a Long Integer.
LineColor	Line color as a Long Integer.
LineTransparence	Line transparency percentage as a Short Integer.
LineWidth	Line width in 1/100 mm as a Long Integer.
LineJoint	Enumerated value of type com.sun.star.drawing.LineJoint: • NONE – The joint between lines is not connected. • MIDDLE – The middle value between joints is used. • BEVEL – The edges are joined by lines. • MITER – Lines join at intersections. • ROUND – Lines are joined with an arc.
LineStartName	Name of the line start point's PolyPolygonBezierCoords.
LineStart	Line start in the form of a PolyPolygonBezierCoords .

Property	Description
LineEnd	Line end in the form of a PolyPolygonBezierCoords.
LineStartCenter	If True, the line starts from the center of the polygon.
LineStartWidth	Width of the line start polygon.
LineEndCenter	If True, the line ends in the center of the polygon.
LineEndWidth	Width of the line end polygon.

Filling space

Shapes that support the com.sun.star.drawing.FillProperties service (see **Table 9**) are able to control how open area in the shape is filled. In general, if the shape is closed, it can be filled.

***Table 9**. Properties defined by the com.sun.star.drawing.FillProperties service.*

Property	Description
FillStyle	Enumerated value of type com.sun.star.drawing.FillStyle: NONE, SOLID, GRADIENT, HATCH, and BITMAP.
FillColor	Color to use if the FillStyle is SOLID.
FillTransparence	Transparency percentage if the FillStyle is SOLID.
FillTransparenceGradientName	Name of the gradient style to use; empty is okay.
FillTransparenceGradient	Defines the gradient with a com.sun.star.awt.Gradient structure: • Style – Enumerated value of type com.sun.star.awt.GradientStyle: LINEAR, AXIAL, RADIAL, ELLIPTICAL, SQUARE, and RECT. • StartColor – Color at the start of the gradient. • EndColor – Color at the end of the gradient. • Angle – Angle of the gradient in 1/10 degree. • Border – Percent of the total width, where just the start color is used. • XOffset – X-coordinate, where the gradient begins. • YOffset – Y-coordinate, where the gradient begins. • StartIntensity – Intensity at the start of the gradient. • EndIntensity – Intensity at the end of the gradient. • StepCount – Number of times the color changes.
FillGradientName	If the FillStyle is GRADIENT, this is the name of the fill gradient style used.
FillGradient	If the FillStyle is GRADIENT, this describes the gradient used.
FillHatchName	If the FillStyle is GRADIENT, this is the name of the fill hatch style used.
FillHatch	If the FillStyle is HATCH, this describes the hatch used.
FillBitmapName	If the FillStyle is BITMAP, this is the name of the fill bitmap style used.
FillBitmap	If the FillStyle is BITMAP, this is the bitmap used.
FillBitmapURL	If the FillStyle is BITMAP, this is a URL to the bitmap used.
FillBitmapOffsetX	The horizontal offset where the tile starts.
FillBitmapOffsetY	The vertical offset where the tile starts. It is given as a percentage in relation to the width of the bitmap.
FillBitmapPositionOffsetX	Every second line of tiles is moved the given percentage of the witdh of the bitmap.
FillBitmapPositionOffsetY	Every second row of tiles is moved the given percentage of the width of the bitmap.
FillBitmapRectanglePoint	The RectanglePoint specifies the position inside the bitmap to use as the top-left position for rendering.
FillBitmapLogicalSize	Specifies if the size is given in percentage or as an absolute value.
FillBitmapSizeX	The width of the tile for filling.
FillBitmapSizeY	The height of the tile for filling.
FillBitmapMode	This enum selects how an area is filled with a single bitmap.
FillBackground	If True, the transparent background of a hatch-filled area is drawn in the current background color.

The macro in **Listing 11** draws a closed Bezier shape. The fill style is set to use a gradient, which means that the darkness of the shape changes over the shape. The resulting shape (see **Figure 4**) contains narrow bands of each color or intensity. You can smooth the appearance of the gradient by using the FillTransparenceGradient property as mentioned in Table 9.

***Listing 11**. DrawClosedBezierShape is found in the Graphic module in this chapter's source code files as SC15.sxi.*

```
Sub DrawClosedBezierShape
  Dim oDoc
  Dim oPage   'Page on which to draw
  Dim oShape  'Shape to insert
  Dim oCoords 'Coordinates of the polygon to insert

  oCoords = createUnoStruct("com.sun.star.drawing.PolyPolygonBezierCoords")

  REM Fill in the actual coordinates. The first and last points
  REM are normal points and the middle points are Bezier control points.
  oCoords.Coordinates = Array(_
    Array(_
      CreatePoint( 1000, 1000 ),_
      CreatePoint( 3000, 4000 ),_
      CreatePoint( 3000, 4000 ),_
      CreatePoint( 5000, 1000 )_
    )_
  )
  oCoords.Flags = Array(_
    Array(_
      com.sun.star.drawing.PolygonFlags.NORMAL,_
      com.sun.star.drawing.PolygonFlags.CONTROL,_
      com.sun.star.drawing.PolygonFlags.CONTROL,_
      com.sun.star.drawing.PolygonFlags.NORMAL _
    )_
  )

  oDoc = ThisComponent
  oPage = createDrawPage(ThisComponent, "Test Draw", True)
  oShape = oDoc.createInstance("com.sun.star.drawing.ClosedBezierShape")
  oPage.add(oShape)
  oShape.FillStyle = com.sun.star.drawing.FillStyle.GRADIENT
  oShape.PolyPolygonBezier = oCoords
End Sub
```

***Figure 4**. Bezier shape using a gradient fill.*

Shadows

Shapes that support the ShadowProperties service can be drawn with a shadow. You can set the shadow location and color by using the properties in **Table 10**.

***Table 10**. Properties defined by the com.sun.star.drawing.ShadowProperties service.*

Property	Description
Shadow	If True, the shape has a shadow.
ShadowColor	Color of the shadow as a Long Integer.
ShadowTransparence	Shadow transparency as a percentage.
ShadowXDistance	Horizontal distance of the left edge of the shape to the shadow.
ShadowYDistance	Vertical distance of the top edge of the shape to the shadow.

A common method for drawing shadows is to draw the shape at an offset location using a shadow color, and then draw the shape normally (see **Figure 5**). With this in mind, consider the properties ShadowXDistance and ShadowYDistance as the distance that the "shadow object" is shifted when it is drawn. The default values for ShadowXDistance and ShadowYDistance are positive, which shifts the shadow right and down. A negative shadow distance shifts the shadow left and up. The macro in **Listing 12** draws two boxes; the first box uses a standard shadow that is shifted right and down, and the second box has the shadow shifted left and down (see Figure 5).

***Listing 12**. drawRectangleWithShadow is found in the Graphic module in this chapter's source code files as SC15.sxi.*

```
Sub drawRectangleWithShadow()
  Dim oPage  'Page on which to draw
  Dim oShape 'Shape to insert

  oPage = createDrawPage(ThisComponent, "Test Draw", True)
  oShape = ThisComponent.createInstance("com.sun.star.drawing.RectangleShape")
  oPage.add(oShape)
  oShape.setPosition(createPoint(1000, 1000))
  oShape.setSize(createSize(4000, 1000))
  oShape.setString("box 1")
  oShape.Shadow = True

  oShape = ThisComponent.createInstance("com.sun.star.drawing.RectangleShape")
  oPage.add(oShape)
  oShape.setPosition(createPoint(6000, 1000))
  oShape.setSize(createSize(4000, 1000))
  oShape.setString("box 2")
  oShape.Shadow = True
  oShape.ShadowXDistance = -150
  oShape.CornerRadius = 100
End Sub
```

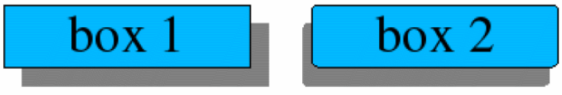

***Figure 5**. Notice the different shadows and that box 2 has rounded corners.*

Rotation and shearing

The com.sun.star.drawing.RotationDescriptor provides the ability to rotate and shear a shape. Shear stretches a shape and would, for example, change a rectangle into a parallelogram. The RotateAngle property is a Long Integer measured in 1/100 degrees. The shape is rotated counterclockwise around the center of the shape's bounding box. The ShearAngle property is also a Long Integer measured in 1/100 degrees, but the shape is sheared clockwise around the center of the bounding box.

The macro in **Listing 13** rotates a rectangle 20 degrees counterclockwise and shears a rectangle 25 degrees clockwise. This code also draws a normal rectangle with no rotation or shear to help you visualize the effects (see **Figure 6**).

***Listing 13**. drawRectangleWithShadow is found in the Graphic module in this chapter's source code files as SC15.sxi.*

```
Sub drawRotateRectangle()
  Dim oPage  'Page on which to draw
  Dim oShape 'Shape to insert

  oPage = createDrawPage(ThisComponent, "Test Draw", True)
  oShape = ThisComponent.createInstance("com.sun.star.drawing.RectangleShape")
  oPage.add(oShape)
  oShape.setPosition(createPoint(1000, 1000))
  oShape.setSize(createSize(4000, 1500))
  oShape.setString("box 1")
  oShape.RotateAngle = 2000  '20 degrees

  oShape = ThisComponent.createInstance("com.sun.star.drawing.RectangleShape")
  oPage.add(oShape)
  oShape.setPosition(createPoint(1000, 1000))
  oShape.setSize(createSize(4000, 1500))
  oShape.FillStyle = com.sun.star.drawing.FillStyle.NONE
  oShape.LineStyle = com.sun.star.drawing.LineStyle.DASH

  oShape = ThisComponent.createInstance("com.sun.star.drawing.RectangleShape")
  oPage.add(oShape)
  oShape.setPosition(createPoint(6000, 1000))
  oShape.setSize(createSize(4000, 1500))
  oShape.setString("box 2")
  oShape.ShearAngle = 2500   '25 degrees

  oShape = ThisComponent.createInstance("com.sun.star.drawing.RectangleShape")
  oPage.add(oShape)
  oShape.setPosition(createPoint(6000, 1000))
  oShape.setSize(createSize(4000, 1500))
  oShape.FillStyle = com.sun.star.drawing.FillStyle.NONE
  oShape.LineStyle = com.sun.star.drawing.LineStyle.DASH
End Sub
```

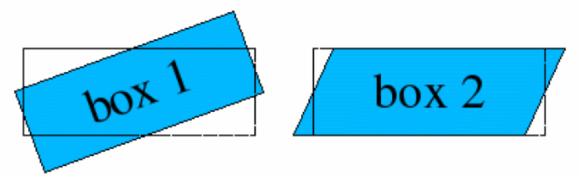

***Figure 6**. The rectangles with dashed lines are the original rectangles.*

Shape types

OOo supports many different shape types, which build on each other. Most of the shape types are obvious from their names. For example, a LineShape is a line. I was initially confused, however, by the gratuitous use of the word "Poly" in shape names such as PolyLineShape and PolyPolygonShape. The prefix "Poly" comes from the Greek and it means "many." So in OOo, a Polygon is a figure containing many angles, a PolyLineShape contains many line shapes, and a PolyPolygonShape contains many polygon shapes.

Simple lines

The purpose of the LineShape service is to draw a simple line. A LineShape requires an initial position and a size. The macro in **Listing 14** draws a line from the point (1000, 1000) to the point (1999, 1999). The endpoint of the line is set by setting the shape's size.

***Listing 14**. SimpleLine is found in the Graphic module in this chapter's source code files as SC15.sxi.*

```
Sub SimpleLine
  Dim oPage  'Page on which to draw
  Dim oShape 'Shape to insert

  oPage = createDrawPage(ThisComponent, "Test Draw", True)
  oShape = ThisComponent.createInstance("com.sun.star.drawing.LineShape")
  oPage.add(oShape)
  oShape.setPosition(CreatePoint(1000, 1000))
  oShape.setSize(CreateSize(1000, 1000))
End Sub
```

Although I've never seen it used, the LineShape service supports the PolyPolygonDescriptor service (see **Table 11**). The implication is that internally simple lines are represented as an open polygon that contains one line. The PolyPolygonDescriptor is used in other services as well.

The PolygonKind enumeration identifies the polygon type (see **Table 12**). The PolygonKind is a read-only property in the PolyPolygonDescriptor service (see Table 11). In other words, you can see what the type is, but you can't set it.

***Table 11**. Properties in the com.sun.star.drawing.PolyPolygonDescriptor service.*

Property	Description
PolygonKind	This read-only property identifies the polygon type (see Table 12).
PolyPolygon	Reference points for this polygon. This is an array of arrays. Each contained array is an array of com.sun.star.awt.Point structures. These points are used to draw the polygon and may have been transformed by a rotation or other transformation.
Geometry	These are the PolyPolygon points with no transformations.

***Table 12**. Values defined by the com.sun.star.drawing.PolygonKind enumeration.*

Value	Description
LINE	Identifies a LineShape.
POLY	Identifies a PolyPolygonShape.
PLIN	Identifies a PolyLineShape.
PATHLINE	Identifies an OpenBezierShape.
PATHFILL	Identifies a ClosedBezierShape.
FREELINE	Identifies an OpenFreeHandShape.
FREEFILL	Identifies a ClosedFreeHandShape.
PATHPOLY	Identifies a PolyPolygonPathShape.
PATHPLIN	Identifies a PolyLinePathShape.

The PolyPolygon property in Table 11 allows you to inspect the actual points used in the creation of the line. The code in **Listing 15** assumes that oShape contains a LineShape object and it displays the two points in the line.

***Listing 15**. Inspect the points used in a LineShape.*

```
  x = oShape.PolyPolygon(0)
  MsgBox "" & x(0).X & " and " & x(0).Y
  MsgBox "" & x(1).X & " and " & x(1).Y
```

PolyLineShape

The LineShape service defines a single line, and the PolyLineShape service defines a series of lines. A LineShape is defined by setting its position and size. A PolyLineShape, however, is defined by the PolyPolygonDescriptor (see Table 11). Although it's easy to create a PolyLineShape when you know how, it isn't widely understood.

The PolyPolygon property is an array of arrays that contain points.

The lines in the PolyLineShape are defined by the PolyPolygon property, which is an array that contains one or more arrays of points. Each array of points is drawn as a series of connected lines, but each array is not specifically connected to the other. The macro in **Listing 16** generates two arrays of points (oPoints_1 and oPoints_2) and then the arrays are stored in another array. See **Figure 7**.

***Listing 16**. SimplePolyLineShape is found in the Graphic module in this chapter's source code files as SC15.sxi.*

```
Sub SimplePolyLineShape
  Dim oPage     'Page on which to draw
  Dim oShape    'Shape to insert
  Dim oPoints_1 'First set of points to plot
  Dim oPoints_2 'Second set of points to plot

  oPoints_1 = Array(_
    CreatePoint( 1000, 1000 ),_
    CreatePoint( 3000, 2000 ),_
    CreatePoint( 1000, 2000 ),_
    CreatePoint( 3000, 1000 )_
  )

  oPoints_2 = Array(_
    CreatePoint( 4000, 1200 ),_
    CreatePoint( 4000, 2000 ),_
    CreatePoint( 5000, 2000 ),_
    CreatePoint( 5000, 1200 )_
  )

  oPage = createDrawPage(ThisComponent, "Test Draw", True)
  oShape = ThisComponent.createInstance("com.sun.star.drawing.PolyLineShape")
  oPage.add(oShape)
  oShape.PolyPolygon = Array(oPoints_1, oPoints_2)
  oShape.LineWidth = 50
End Sub
```

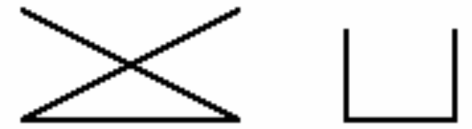

***Figure 7**. A single PolyLineShape produces two shapes that are not connected.*

The shape is added to the draw page before points are assigned to the PolyPolygon property.

The PolyPolygon property (shown in **Listing 17**) is an array of arrays. You can run the macro in Listing 16 with only one set of points, but the single array of points must still reside inside a second array.

***Listing 17**. The PolyPolygon property is an array of arrays of points.*

```
oShape.PolyPolygon = Array(oPoints_1)
```

PolyPolygonShape

The PolyPolygonShape service defines a series of closed polygons that are not connected (see **Figure 8**). This service is essentially a closed-shape version of the PolyLineShape. Because it produces closed shapes, the PolyPolygonShape service supports fill properties.

The macro in **Listing 18** uses the same set of points as the macro in Listing 16, but I moved things around to demonstrate different methods for creating an array of arrays of points. Both macros, however, create the shape and add it to the draw page before setting the properties.

***Listing 18**. SimplePolyPolygonShape is found in the Graphic module in this chapter's source code files as SC15.sxi.*

```
Sub SimplePolyPolygonShape
  Dim oPage     'Page on which to draw
  Dim oShape    'Shape to insert

  oPage = createDrawPage(ThisComponent, "Test Draw", True)
  oShape = ThisComponent.createInstance("com.sun.star.drawing.PolyPolygonShape")
  oPage.add(oShape)
  oShape.PolyPolygon = Array(_
    Array( CreatePoint( 1000, 1000 ),_
           CreatePoint( 3000, 2000 ),_
           CreatePoint( 1000, 2000 ),_
           CreatePoint( 3000, 1000 )_
         ),_
    Array( CreatePoint( 4000, 1200 ),_
           CreatePoint( 4000, 2000 ),_
           CreatePoint( 5000, 2000 ),_
           CreatePoint( 5000, 1200 )_
         )_
    )

  oShape.LineWidth = 50
End Sub
```

***Figure 8**. The PolyPolygonShape produces a closed-shape version of the PolyLineShape.*

RectangleShape and TextShape

Externally, the RectangleShape and the TextShape are virtually identical. The two shape types support the same set of services (except for the defining service, of course) and they can be configured to produce the same output. The primary difference between the two shape types is their default values while producing output. In principle, properties can be adjusted from default values, so that each type could produce either output. The macro in **Listing 19** creates a rectangle shape and a text shape next to each other (see **Figure 9**).

***Listing 19**. SimpleRectangleShape is found in the Graphic module in this chapter's source code files as SC15.sxi.*

```
Sub SimpleRectangleShape
  Dim oPage     'Page on which to draw
  Dim oShape    'Shape to insert

  oPage = createDrawPage(ThisComponent, "Test Draw", True)
  oShape = ThisComponent.createInstance("com.sun.star.drawing.RectangleShape")
  oPage.add(oShape)

  oShape.setPosition(createPoint(1000, 1000))
  oShape.setSize(createSize(6000, 1000))
  oShape.setString("rectangle")
  oShape.Shadow = True
  oShape = ThisComponent.createInstance("com.sun.star.drawing.TextShape")
  oPage.add(oShape)

  oShape.setPosition(createPoint(8000, 1000))
  oShape.setSize(createSize(10000, 1000))
  oShape.setString("text")
```

```
  oShape.Shadow = True
End Sub
```

***Figure 9**. Two shapes that are drawn in the same way—but with different, nearly identical shape types that have different default values—produce different output.*

The RectangleShape and the TextShape types both support the CornerRadius property. The corner radius is a Long Integer that indicates the radius of the circle used to produce the corners. This is demonstrated in Figure 5 as produced by Listing 12.

EllipseShape

A mathematician would say that an ellipse is a closed curve that is formed from two points (called the foci) in which the sum of the distances from any point on the curve to the two points is a constant. If the two foci are at the same point, the ellipse is a circle. In simpler terms, an ellipse is a circle or a squashed circle.

While drawing a rectangle, the position identifies the upper-left corner of the rectangle and then the size defines the width and height. If the same point and size is used to draw an ellipse, the ellipse will be contained inside the rectangle and will just barely touch the four sides of the rectangle. Mathematically, the sides of the rectangle are tangent to the ellipse at its principal axes, the maximum and minimum distances across the ellipse. The macro in **Listing 20** starts by drawing four ellipse shapes. The final ellipse is rotated 30 degrees. The macro then draws a rectangle using the same position, size, and rotation as the last ellipse (see **Figure 10**). The final rectangle helps to illustrate the relationship between a rectangle and an ellipse.

***Listing 20**. SimpleEllipseShapes is found in the Graphic module in this chapter's source code files as SC15.sxi.*

```
Sub SimpleEllipseShapes
  Dim oPage     'Page on which to draw
  Dim oShape    'Shape to insert
  Dim i%
  Dim x
  Dim nLocs
  nLocs = Array(_
    Array(CreatePoint(1000, 1000), createSize(1000, 1000)),_
    Array(CreatePoint(3000, 1000), createSize(1000, 1500)),_
    Array(CreatePoint(5000, 1000), createSize(1500, 1000)),_
    Array(CreatePoint(7000, 1000), createSize(1500, 1000))_
  )

  oPage = createDrawPage(ThisComponent, "Test Draw", True)
  For i = LBound(nLocs) To UBound(nLocs)
    oShape = ThisComponent.createInstance("com.sun.star.drawing.EllipseShape")
    oPage.add(oShape)
    x = nLocs(i)
    oShape.setPosition(x(0))
    oShape.setSize(x(1))
    oShape.setString(i)
  Next
  oShape.RotateAngle = 3000

  REM Now draw a rectangle the same size as the last ellipse.
  oShape = ThisComponent.createInstance("com.sun.star.drawing.RectangleShape")
  oPage.add(oShape)
  oShape.setPosition(x(0))
  oShape.setSize(x(1))
  oShape.RotateAngle = 3000
  oShape.FillStyle = com.sun.star.drawing.FillStyle.NONE
End Sub
```

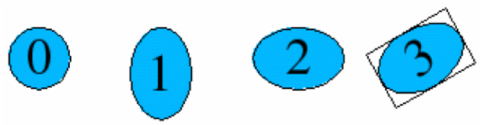

***Figure 10**. The size parameters determine shape; other parameters set position and orientation.*

The EllipseShape service contains a property of type CircleKind that determines whether the entire ellipse should be drawn, or only a portion of it (see **Table 13**). In other words, you can draw an arc. The properties CircleStartAngle and CircleEndAngle define where the arc starts and ends. Each ellipse in Figure 10 uses a FULL CircleKind.

***Table 13**. Values defined by the com.sun.star.drawing.CircleKind enumeration.*

Value	Description
com.sun.star.drawing.CircleKind.FULL	A full ellipse.
com.sun.star.drawing.CircleKind.SECTION	An ellipse with a cut connected by a line.
com.sun.star.drawing.CircleKind.CUT	An ellipse with a cut connected by two lines.
com.sun.star.drawing.CircleKind.ARC	An ellipse with an open cut.

The four different circle kinds are drawn by **Listing 21** and shown in **Figure 11**.

***Listing 21**. ArcEllipseShapes is found in the Graphic module in this chapter's source code files as SC15.sxi.*

```
Sub ArcEllipseShapes
  Dim oPage     'Page on which to draw
  Dim oShape    'Shape to insert
  Dim i%
  Dim x
  Dim nLocs
  nLocs = Array(_
    com.sun.star.drawing.CircleKind.FULL,_
    com.sun.star.drawing.CircleKind.SECTION,_
    com.sun.star.drawing.CircleKind.CUT,_
    com.sun.star.drawing.CircleKind.ARC,_
  )

  oPage = createDrawPage(ThisComponent, "Test Draw", True)
  For i = LBound(nLocs) To UBound(nLocs)
    oShape = ThisComponent.createInstance("com.sun.star.drawing.EllipseShape")
    oPage.add(oShape)
    oShape.setPosition(CreatePoint((i+1)*2000, 1000))
    oShape.setSize(CreateSize(1000,700))
    oShape.setString(i)
    oShape.CircleStartAngle = 9000
    oShape.CircleEndAngle = 36000
    oShape.CircleKind = nLocs(i)
  Next
End Sub
```

***Figure 11**. Each supported CircleKind drawn in order.*

Bezier curves

A Bezier curve is a smooth curve controlled by multiple points. Bezier curves connect the first and last points, but are only influenced by the other points. Mathematicians like Bezier curves because they are invariant under any affine mapping (any arbitrary combination of translation or rotation). Computer graphics professionals like Bezier curves because they are easy to manipulate and transform.

Bezier curves are controlled by a PolyPolygonBezierDescriptor (see **Table 14**), which is almost identical to the PolyPolygonDescriptor described in Table 11. The difference between the two descriptors is that each point in the Bezier curve is categorized based on how the point affects the curve.

***Table 14**. Properties in the com.sun.star.drawing.PolyPolygonBezierDescriptor service.*

Property	Description
PolygonKind	This read-only property identifies the polygon type (see Table 12).
PolyPolygonBezier	Reference points for this Bezier curve. This is a PolyPolygonBezierCoords structure. The structure contains an array of points and an array of flags to categorize each point as to its function in the curve.
Geometry	This is the PolyPolygonBezierCoords with no transformations.

The PolygonBezier property (see Table 14) is a PolyPolygonBezierCoords structure that contains two properties, Coordinates and Flags. The Coordinates property is an array of arrays of points that represent the control points for the Bezier curve. The Flags property is an array of arrays of PolygonFlags (see **Table 15**) that identifies how the corresponding point affects the curve.

***Table 15**. Values in the com.sun.star.drawing.PolygonFlags enumeration.*

Value	Description
NORMAL	The curve travels through normal points.
SMOOTH	The point is smooth through the point.
CONTROL	Influence the curve.
SYMMETRIC	The point is symmetric through the point.

The macro in **Listing 22** draws a small circle at each point in the Coordinates array. Drawing each point helps to visualize and understand how the different points affect the shape of a Bezier curve.

***Listing 22**. DrawControlPoints is found in the Graphic module in this chapter's source code files as SC15.sxi.*

```
Sub DrawControlPoints(oCoords, oPage, oDoc, nWidth As Long)
  Dim oPoints  'One subarray of points
  Dim oPoint   'One point
  Dim oFlags   'One subarray of flags
  Dim oShape   'The circle to draw
  Dim nShape%  'Index into the oCoords arrays
  Dim i%       'General index variable

  For nShape = LBound(oCoords.Coordinates) To UBound (oCoords.Coordinates)
    oPoints = oCoords.Coordinates(nShape)
    oFlags  = oCoords.Flags(nShape)
    For i = LBound(oPoints) To UBound(oPoints)
      oShape = oDoc.createInstance("com.sun.star.drawing.EllipseShape")
      oPage.add(oShape)
      oPoint = oPoints(i)
      REM To center the circle, I need to set the position
      REM as half width back and half width up.
      oShape.setPosition(CreatePoint(oPoint.X-nWidth/2, oPoint.Y-nWidth/2))
      oShape.setSize(CreateSize(nWidth, nWidth)
    Next
  Next
End Sub
```

Listing 23 draws two disconnected Bezier curves (see **Figure 12**). The second curve places two control points at the same location. The DrawControlPoints macro in Listing 22 is used to draw the control points along with the Bezier curve.

***Listing 23**. DrawOpenBezierCurves is found in the Graphic module in this chapter's source code files as SC15.sxi.*

```
Sub DrawOpenBezierCurves()
  Dim oPage     'Page on which to draw
  Dim oShape    'Shape to insert
  Dim oDoc
  Dim i%
  Dim oCoords 'Coordinates of the polygon to insert

  oCoords = createUnoStruct("com.sun.star.drawing.PolyPolygonBezierCoords")

  REM Fill in the actual coordinates. The first and last points
  REM are normal points and the middle points are Bezier control points.
  oCoords.Coordinates = Array(_
    Array(_
      CreatePoint( 1000, 1000 ),_
      CreatePoint( 2000, 3000 ),_
      CreatePoint( 3000, 0500 ),_
      CreatePoint( 4000, 1000 ),_
    ),_
    Array(_
      CreatePoint( 5000, 1000 ),_
      CreatePoint( 6500, 0200 ),_
      CreatePoint( 6500, 0200 ),_
      CreatePoint( 8000, 1000 ),_
    ),_
  )
  oCoords.Flags = Array(_
    Array(_
      com.sun.star.drawing.PolygonFlags.NORMAL,_
      com.sun.star.drawing.PolygonFlags.CONTROL,_
      com.sun.star.drawing.PolygonFlags.CONTROL,_
      com.sun.star.drawing.PolygonFlags.NORMAL,_
    ),_
    Array(_
      com.sun.star.drawing.PolygonFlags.NORMAL,_
      com.sun.star.drawing.PolygonFlags.CONTROL,_
      com.sun.star.drawing.PolygonFlags.CONTROL,_
      com.sun.star.drawing.PolygonFlags.NORMAL,_
    )_
  )

  oDoc = ThisComponent
  oPage = createDrawPage(ThisComponent, "Test Draw", True)
  oShape = oDoc.createInstance("com.sun.star.drawing.OpenBezierShape")
  oPage.add(oShape)
  oShape.PolyPolygonBezier = oCoords
  DrawControlPoints(oCoords, oPage, oDoc, 100)
End Sub
```

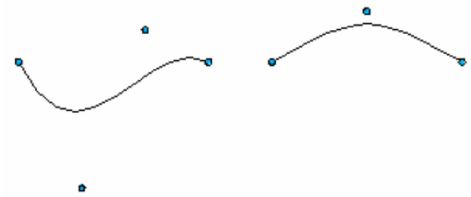

***Figure 12**. Notice how the control points influence the curve.*

Not all combinations of points and flags are valid. A complete discussion of what constitutes a valid combination of points and flags is beyond the scope of this book. A run-time error occurs if you use an incorrect number of points or an unsupported sequence of control flags.

ConnectorShape

Use the ConnectorShape to provide a connection between two shapes. A “glue point” is a position inside a shape where the endpoint of a ConnectorShape can connect. Each glue point is defined by the GluePoint2 structure (see **Table 16**).

***Table 16**. Properties in the com.sun.star.drawing.GluePoint2 structure.*

Property	Description
Position	Glue-point position as a point structure.
IsRelative	If True, Position is given in 1/100 percent.
PositionAlignment	com.sun.star.drawing.Alignment enumerated value that specifies how the point is moved if the shape is resized. Valid values include: TOP_LEFT, TOP, TOP_RIGHT, LEFT, CENTER, RIGHT, BOTTOM_LEFT, BOTTOM, and BOTTOM_RIGHT.
Escape	com.sun.star.drawing.EscapeDirection enumerated value that specifies the escape direction for a glue point. Valid values include: SMART, LEFT, RIGHT, UP, DOWN, HORIZONTAL, and VERTICAL.
IsUserDefined	If False, this is a default glue point.

Each shape contains a default glue point at the top, right, bottom, and left of the shape. You can access a shape's glue points by using the getGluePoints() method (see Table 4). The index of the default glue points are 0 (top), 1 (right), 2 (bottom), and 3 (left). You also can add new glue points to a shape's default glue points (see Listing 25).

Connector shapes contain StartPosition and EndPosition properties (see **Table 17**), which identify the connector's start and end positions. The start and end positions are used only if the corresponding properties StartShape and EndShape are empty. If the StartShape and EndShape properties are not empty, the connector shape connects to a glue point in the corresponding shape. Connector shapes reference other shapes' glue points by index using the StartGluePointIndex and EndGluePointIndex properties.

***Table 17**. Properties in the com.sun.star.drawing.ConnectorShape service.*

Property	Description
StartShape	Start shape, or empty if the start point is not connected to a shape.
StartGluePointIndex	Index of the glue point in the start shape.
StartPosition	Start point position in 1/100 mm. You can set the position only if the start point is not connected, but you can always read the point.
EndShape	End shape, or empty if the start point is not connected to a shape.
EndPosition	End point position in 1/100 mm. You can set the position only if the end point is not connected, but you can always read the point.
EndGluePointIndex	Index of the glue point in the end shape.
EdgeLine1Delta	Distance of line 1.
EdgeLine2Delta	Distance of line 2.
EdgeLine3Delta	Distance of line 3.
EdgeKind	Type of connector (see Table 18).

Four connector types are supported (see **Table 18**). The connector type determines how the line is drawn between the two points. The STANDARD type prefers to use three lines to connect the shapes, but it will use more lines if required.

***Table 18**. Properties in the com.sun.star.drawing.ConnectorType enumeraiton.*

Value	Description
STANDARD	The ConnectorShape is drawn with three lines, with the middle line perpendicular to the other two.
CURVE	The ConnectorShape is drawn as a curve.
LINE	The ConnectorShape is drawn as one straight line.
LINES	The ConnectorShape is drawn with three lines.

As of OOo 1.1.1, the LINE connector type is not available by name; you must use the corresponding integer value of 2. This is scheduled to be fixed by OOo 2.0.

The macro in **Listing 24** draws four rectangles and then connects the rectangles using a ConnectorShape. Although the macro specifies the initial glue point, the end glue point is automatically chosen by OOo. If the macro did not explicitly set the initial glue point, it also would be automatically chosen. When a glue point is automatically chosen, it is done intelligently, as you can see in **Figure 13**.

***Listing 24**. DrawConnectorShape is found in the Graphic module in this chapter's source code files as SC15.sxi.*

```
Sub DrawConnectorShape
  PrintConTypes()
  Dim oPage     'Page on which to draw
  Dim oShapes   'Shape to insert
  Dim oShape    'Shape to insert
  Dim nConTypes
  Dim oDoc
  Dim i%

  oDoc = ThisComponent

  REM The value com.sun.star.drawing.ConnectorType.LINE does not work!
  nConTypes = Array(_
    com.sun.star.drawing.ConnectorType.STANDARD,_
    com.sun.star.drawing.ConnectorType.CURVE,_
    2,_
    com.sun.star.drawing.ConnectorType.LINES,_
    )

  oShapes = Array(_
    oDoc.createInstance("com.sun.star.drawing.RectangleShape"),_
    oDoc.createInstance("com.sun.star.drawing.RectangleShape"),_
    oDoc.createInstance("com.sun.star.drawing.RectangleShape"),_
    oDoc.createInstance("com.sun.star.drawing.RectangleShape"),_
    )

  REM Create the draw page and then add the shapes before manipulating them.
  oPage = createDrawPage(oDoc, "Test Draw", True)
  For i = 0 To 3
    oPage.add(oShapes(i))
    oShapes(i).setSize(createSize(1300, 1000))
  Next
  oShapes(0).setPosition(createPoint(1000, 1500))
  oShapes(1).setPosition(createPoint(4000, 1000))
  oShapes(2).setPosition(createPoint(7000,  500))
  oShapes(3).setPosition(createPoint(4000, 2500))

  For i = 0 To 3
    oShapes(i).setString(i)
    oShape = oDoc.createInstance("com.sun.star.drawing.ConnectorShape")
    oPage.add(oShape)
    oShape.StartShape = oShapes(i)
    oShape.StartGluePointIndex = i
    oShape.EndShape = oShapes((i + 1) MOD 4)
    oShape.EdgeKind = nConTypes(i)
  Next
End Sub
```

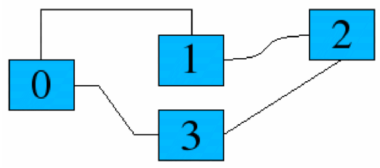

***Figure 13**. Notice the different connector types.*

Many of a shape's properties are reset when the shape is added to a draw page. Therefore, you should set most properties after adding the shape to the draw page. It is also important to set properties in the correct order because setting some properties resets other properties. For example, setting a connector's StartShape resets the StartGluePointIndex.

Creating your own glue points

If you want to attach a connector to a shape at a location of your choosing, you must create a GluePoint2 structure and add it to the shape. Modify Listing 24 by adding the code from **Listing 25** immediately after setting the EdgeKind property. Listing 25 creates a glue point located at the center of the rectangle (see **Figure 14**) and then uses this point as the start point. Table 16 contains a description of the GluePoint2 structure.

***Listing 25**. DrawConnectorShape_Glue is found in the Graphic module in this chapter's source code files as SC15.sxi.*

```
Rem Now create a glue point in the center of the shape.
oGlue = createUnoStruct("com.sun.star.drawing.GluePoint2")
oGlue.IsRelative = False
oGlue.Escape = com.sun.star.drawing.EscapeDirection.SMART
oGlue.PositionAlignment = com.sun.star.drawing.Alignment.CENTER
oGlue.IsUserDefined = True
oGlue.Position.X = oShapes(i).getPosition().X + 650
oGlue.Position.Y = oShapes(i).getPosition().Y + 500
oShape.StartGluePointIndex = oShapes(i).getGluePoints().insert(oGlue)
```

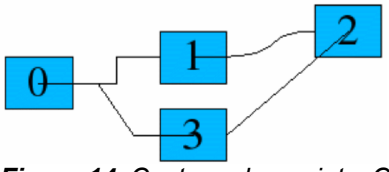

***Figure 14**. Custom glue points: Connectors start in the middle of the rectangles.*

Adding arrows by using styles

You can set many properties for a shape by creating a style. If you frequently use certain fill styles and shadow styles, you should create a special style so you can quickly change the objects if required. **Listing 26** displays the graphics styles supported by an Impress document (shown in **Figure 15**).

You can add arrows to a shape by setting the shape's style to "objectwitharrow". If you don't like the default values in the "objectwitharrow" style, which produces very wide lines (see **Figure 16**), you can create your own style and use that instead. Modify Listing 24 by adding the code from **Listing 27** immediately after setting the EdgeKind property.

***Listing 26**. PrintStyles is found in the Graphic module in this chapter's source code files as SC15.sxi.*

```
Sub PrintStyles
  Dim oStyles
  Dim oStyleFamilies
  oStyleFamilies = ThisComponent.getStyleFamilies()
  MsgBox Join(oStyleFamilies.getElementNames(), CHR$(10)), 0, "Families"

  oStyles = ThisComponent.getStyleFamilies().getByName("graphics")
  MsgBox Join(oStyles.getElementNames(), CHR$(10)), 0, "Graphics Styles"
End Sub
```

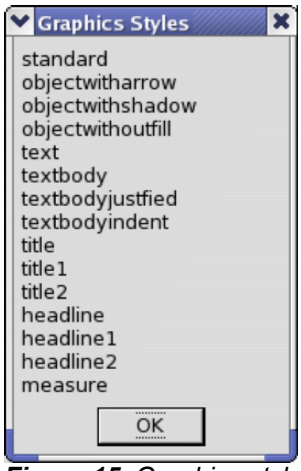

***Figure 15**. Graphics styles supported by presentation documents.*

***Listing 27**. DrawConnectorShape_Arrows is found in the Graphic module in this chapter's source code files as SC15.sxi.*

```
oStyles = oDoc.getStyleFamilies().getByName("graphics")
oShape.Style = oStyles.getByName("objectwitharrow")
```

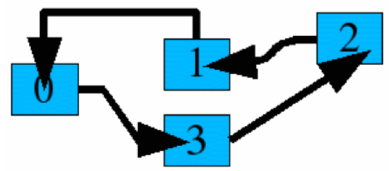

***Figure 16**. The default arrow style draws an arrow from the end shape to the start shape.*

Forms

When a control is inserted into a document, it is stored in a form. Forms are contained in draw pages, which implement the com.sun.star.form.XFormsSupplier interface. The visible portion of the control—what you see—is stored in a draw page and represented by a ControlShape. The data model for the control is stored in a form and referenced by the ControlShape. The method getForms() returns an object that contains the forms for the draw page (see **Table 19**).

***Table 19**. Some methods supported by the com.sun.star.form.Forms service.*

Method	Description
createEnumeration()	Create an enumeration of the forms.
getByIndex(Long)	Get a form by index.
getByName(String)	Get a form by name.
getCount()	Get the number of forms.
hasByName(String)	Return True if the form with the specified name exists.
hasElements()	Return True if the page contains at least one form.
insertByIndex(Long, Form)	Insert a form by index.
insertByName(String, Form)	Insert a form by name.
removeByIndex(Long)	Remove a form by index.
removeByName(String)	Remove the named form.
replaceByIndex(Long, Form)	Replace a form by index.
replaceByName(String, Form)	Replace a form by name.

The purpose of **Listing 28** is to demonstrate how to add a form to a draw page. Forms are not interesting unless they contain a control, so Listing 28 adds a drop-down list box with some values to select.

***Listing 28**. AddAForm is found in the Graphic module in this chapter's source code files as SC15.sxi.*

```
Sub AddAForm
  Dim oPage            'Page on which to draw
  Dim oShape           'Shape to insert
  Dim oDoc             'ThisComponent
  Dim oForm            'Individual form
  Dim oControlModel    'Model for a control
  Dim s (0 To 5) As String

  REM Data for the combo box!
  s(0) = "Zero"   : s(1) = "One"   : s(2) = "Two"
  s(3) = "Three"  : s(4) = "Four"  : s(5) = "Five"

  oDoc = ThisComponent
  oPage = createDrawPage(oDoc, "Test Draw", True)

  REM Create a shape for the control.
  oShape = oDoc.createInstance("com.sun.star.drawing.ControlShape")
  oShape.Position = createPoint(1000, 1500)
  oShape.Size = createSize(2500, 800)

  REM Create a combo box model.
  oControlModel = oDoc.createInstance("com.sun.star.form.component.ComboBox")
  oControlModel.Name = "NumberSelection"
  oControlModel.Text = "Zero"
  oControlModel.Dropdown = True
  oControlModel.StringItemList = s()

  REM Set the shape's control model!
  oShape.Control = oControlModel

  oForm = oDoc.createInstance("com.sun.star.form.component.Form")
  oForm.Name = "NumberForm"
  oPage.Forms.insertByIndex( 0, oForm )

  REM Add the control model to the first form in the collection.
  oForm.insertByIndex( 0, oControlModel )
  oPage.add( oShape )
End Sub
```

Regular forms, as created by Listing 28, group form components (described in **Table 20**) together. A DataForm, however, can connect to a database and display the results of SQL queries. An HTMLForm, on the other hand, contains controls specific to HTML pages.

***Table 20**. Control components that can be added to forms.*

Component	Description
CheckBox	Check box control.
ComboBox	Provides text input or selection from a list of text values.
CommandButton	A clickable button.
CurrencyField	An edit field with a currency value.
DatabaseCheckBox	A data-aware check box that can be bound to a database field.
DatabaseComboBox	A data-aware combo box that can be bound to a database field.
DatabaseCurrencyField	A data-aware edit field with a currency value that can be bound to a database field.
DatabaseDateField	A data-aware date field that can be bound to a database field.
DatabaseFormattedField	A data-aware formatted field that can be bound to a database field.
DatabaseImageControl	A field for displaying images stored in a database.
DatabaseListBox	A data-aware list box that can be bound to a database field.
DatabaseNumericField	A data-aware numeric field that can be bound to a database field.
DatabasePatternField	A data-aware pattern field that can be bound to a database field.
DatabaseRadioButton	A data-aware radio button that can be bound to a database field.

Component	Description
DatabaseTextField	A data-aware text field that can be bound to a database field.
DatabaseTimeField	A data-aware time field that can be bound to a database field.
DateField	An edit field with a date value.
FileControl	An edit field for a file name.
FixedText	Display text that cannot be edited by the user.
FormattedField	An edit field that contains formatted text.
GridControl	Display data in a table-like way.
GroupBox	A control that can visually group controls.
HiddenControl	A control that is hidden.
ImageButton	A clickable button which is represented by an image.
ListBox	A control with multiple values from which to choose.
NumericField	An edit field with a numeric value.
PatternField	An edit field with text that matches a pattern.
RadioButton	A radio button.
TextField	A text-edit field that supports single-line and multi-line data.
TimeField	An edit field with a time value.

Presentations

The Presentation service contains properties (see **Table 21**) and methods that control a specific presentation. You can create multiple presentation objects for different types of presentations. For example, you might have one presentation that runs continuously at a trade show and one that requires manual intervention—for example, for a client sales visit. The document's getPresentation() method returns a new presentation object. After setting a presentation's properties as shown in Table 21, the methods start() and end() are used to start and stop a presentation. The method rehearseTimings() starts a presentation while displaying a running clock to help you determine the running time of your presentation. See **Listing 29**.

***Table 21**. Properties defined by the the com.sun.star.presentation.Presentation service.*

Property	Description
AllowAnimations	If True, animations are enabled.
CustomShow	Name of a customized show to use for this presentation; an empty value is allowed.
FirstPage	Name of the first page in the presentation; an empty value is allowed.
IsAlwaysOnTop	If True, the presentation window is always the top window.
IsAutomatic	If True, page changes happen automatically.
IsEndless	If True, the presentation repeats endlessly.
IsFullScreen	If True, the presentation runs in full-screen mode.
IsLivePresentation	If True, the presentation runs in live mode.
IsMouseVisible	If True, the mouse is visible during the presentation.
Pause	Long Integer duration that the black screen is displayed after the presentation is finished.
StartWithNavigator	If True, the Navigator opens at the start of the presentation.
UsePen	If True, a pen appears during the presentation so that you can draw on the screen.

***Listing 29**. SimplePresentation is found in the Graphic module in this chapter's source code files as SC15.sxi.*

```
Sub SimplePresentation()
  Dim oPres
  oPres = ThisComponent.getPresentation()
```

```
  oPres.UsePen = True
  REM This will start the presentation.
  REM Be ready to press the space bar to move through the slides.
  oPres.Start()
End Sub
```

A custom presentation can show the presentation's pages in any order. Pages can be shown multiple times or not at all. The getCustomPresentations() method returns a custom presentations object that contains all custom presentations (see **Table 22**).

***Table 22**. Some methods supported by the XCustomPresentationSupplier interface.*

Method	Description
createInstance()	Create a custom presentation.
getByName(String)	Get a custom presentation by name.
getElementNames()	Array of custom presentation names.
hasByName(String)	Return True if the custom presentation with the specified name exists.
hasElements()	Return True if the page contains at least one custom presentation.
insertByName(String, CustomPresentation)	Insert a custom presentation by name.
removeByName(String)	Remove the named custom presentation.
replaceByName(String, CustomPresentation)	Replace a custom presentation by name.

A custom presentation (shown in **Listing 30**) is a container for draw pages that supports the XNamed and XIndexedAccess interfaces. Create the custom presentation, add the draw pages in the order that you want them to appear, and then save the custom presentation. A custom presentation is displayed in exactly the same way that regular presentations are displayed, using a Presentation object, but the CustomShow attribute is set to reference the custom presentation.

***Listing 30**. CustomPresentation is found in the Graphic module in this chapter's source code files as SC15.sxi.*

```
Sub CustomPresentation()
  Dim oPres   'Presentations, both customer and regular
  Dim oPages  'Draw pages

  oPres  = ThisComponent.getCustomPresentations().createInstance()
  If NOT ThisComponent.getCustomPresentations().hasByName("custom") Then
    oPages = ThisComponent.getDrawPages()

    REM Display pages 0, 2, 1, 0
    oPres.insertByIndex(0, oPages.getByIndex(0))
    oPres.insertByIndex(1, oPages.getByIndex(2))
    oPres.insertByIndex(2, oPages.getByIndex(1))
    oPres.insertByIndex(3, oPages.getByIndex(0))
    ThisComponent.getCustomPresentations().insertByName("custom", oPres)
  End If

  REM Now, run the customer presentation.
  oPres = ThisComponent.getPresentation()
  oPres.CustomShow = "custom"
  oPres.Start()
End Sub
```

Presentation draw pages

Draw pages in a presentation document are slightly different from those in a drawing document. The properties in **Table 23** dictate how and when page transitions occur while showing presentations.

***Table 23**. Properties defined by the com.sun.star.presentation.DrawPage service.*

Property	Description
Change	Long Integer that specifies what causes a page change. • 0 – A mouse-click triggers the next animation or page change. • 1 – The page change is automatic. • 2 – Object effects run automatically, but the user must click on the page to change it.
Duration	Long Integer time in seconds the page is shown if the Change property is set to 1.
Effect	Effect used to fade in or out (see Table 24).
Layout	Index of the presentation layout page if this is not zero.
Speed	Speed of the fade-in effect using the com.sun.star.presentation.AnimationSpeed enumeration: SLOW, MEDIUM, or FAST.

Page transitions are governed by the Effect property of the presentation draw page (see **Table 24**). The macro in **Listing 31** sets the transitions on all draw pages to RANDOM.

***Table 24**. Values defined by the com.sun.star.presentation.FadeEffect enumeration.*

Values	Values	Values
NONE DISSOLVE RANDOM	VERTICAL_STRIPES VERTICAL_CHECKERBOARD VERTICAL_LINES	HORIZONTAL_STRIPES HORIZONTAL_CHECKERBOARD HORIZONTAL_LINES
FADE_FROM_LEFT FADE_FROM_TOP FADE_FROM_RIGHT FADE_FROM_BOTTOM FADE_FROM_UPPERLEFT FADE_FROM_UPPERRIGHT FADE_FROM_LOWERLEFT FADE_FROM_LOWERRIGHT	MOVE_FROM_LEFT MOVE_FROM_TOP MOVE_FROM_RIGHT MOVE_FROM_BOTTOM MOVE_FROM_UPPERLEFT MOVE_FROM_UPPERRIGHT MOVE_FROM_LOWERRIGHT MOVE_FROM_LOWERLEFT	UNCOVER_TO_LEFT UNCOVER_TO_UPPERLEFT UNCOVER_TO_TOP UNCOVER_TO_UPPERRIGHT UNCOVER_TO_RIGHT UNCOVER_TO_LOWERRIGHT UNCOVER_TO_BOTTOM UNCOVER_TO_LOWERLEFT
FADE_TO_CENTER FADE_FROM_CENTER CLOCKWISE COUNTERCLOCKWISE	ROLL_FROM_LEFT ROLL_FROM_TOP ROLL_FROM_RIGHT ROLL_FROM_BOTTOM	CLOSE_VERTICAL CLOSE_HORIZONTAL OPEN_VERTICAL OPEN_HORIZONTAL
STRETCH_FROM_LEFT STRETCH_FROM_TOP STRETCH_FROM_RIGHT STRETCH_FROM_BOTTOM	WAVYLINE_FROM_LEFT WAVYLINE_FROM_TOP WAVYLINE_FROM_RIGHT WAVYLINE_FROM_BOTTOM	SPIRALIN_LEFT SPIRALIN_RIGHT SPIRALOUT_LEFT SPIRALOUT_RIGHT

***Listing 31**. SetTransitionEffects is found in the Graphic module in this chapter's source code files as SC15.sxi.*

```
Sub SetTransitionEffects()
  Dim oPages  'Draw pages
  Dim i%

  oPages = ThisComponent.getDrawPages()
  For i = 0 To oPages.getCount() - 1
    With oPages.getByIndex(i)
      .Effect = com.sun.star.presentation.FadeEffect.RANDOM
      .Change = 1
      .Duration = 2
      .Speed = com.sun.star.presentation.AnimationSpeed.FAST
    End With
  Next
End Sub
```

Presentation shapes

Shapes contained in Impress documents differ from shapes in Draw documents in that they support the com.sun.star.presentation.Shape service. The presentation Shape service provides properties that define special behavior to enhance presentations (see **Table 25**).

***Table 25**. Properties defined by the com.sun.star.presentation.Shape service.*

Property	Description
Bookmark	Generic URL string used if the OnClick property requires a URL.
DimColor	Color for dimming this shape if DimPrevious is True and DimHide is False.
DimHide	If True and DimPrevious is True, the shape is hidden.
DimPrevious	If True, the shape is dimmed after executing its animation effect.
Effect	Animation effect for this shape (see Table 26).
IsEmptyPresentationObject	True if this is the default presentation shape and it is empty.
IsPresentationObject	True if this is a presentation object.
OnClick	Specify an action if the user clicks the shape (see Table 27).
PlayFull	If True, the sound of this shape is played in full.
PresentationOrder	Long Integer representing the order in which the shapes are animated.
Sound	URL string for a sound file that is played while the shape's animation is running.
SoundOn	If True, sound is played during the animation.
Speed	Speed of the fade-in effect using the com.sun.star.presentation.AnimationSpeed enumeration: SLOW, MEDIUM, or FAST.
TextEffect	Animation effect for the text inside this shape (see Table 26).
Verb	Long Integer "ole2" verb if the ClickAction is VERB.

The animation effects supported by shapes (see **Table 26**) are similar to, but more plentiful than, the animation effects supported by draw pages (see Table 21).

***Table 26**. Values defined by the com.sun.star.presentation.AnimationEffect enumeration.*

Property	Property	Property
NONE RANDOM PATH	DISSOLVE APPEAR HIDE	CLOCKWISE COUNTERCLOCKWISE
MOVE_FROM_LEFT MOVE_FROM_TOP MOVE_FROM_RIGHT MOVE_FROM_BOTTOM MOVE_FROM_UPPERLEFT MOVE_FROM_UPPERRIGHT MOVE_FROM_LOWERRIGHT MOVE_FROM_LOWERLEFT	MOVE_TO_LEFT MOVE_TO_TOP MOVE_TO_RIGHT MOVE_TO_BOTTOM MOVE_TO_UPPERLEFT MOVE_TO_UPPERRIGHT MOVE_TO_LOWERRIGHT MOVE_TO_LOWERLEFT	MOVE_SHORT_TO_LEFT MOVE_SHORT_TO_TOP MOVE_SHORT_TO_RIGHT MOVE_SHORT_TO_BOTTOM MOVE_SHORT_TO_UPPERLEFT MOVE_SHORT_TO_UPPERRIGHT MOVE_SHORT_TO_LOWERRIGHT MOVE_SHORT_TO_LOWERLEFT
MOVE_SHORT_FROM_LEFT MOVE_SHORT_FROM_TOP MOVE_SHORT_FROM_RIGHT MOVE_SHORT_FROM_BOTTOM MOVE_SHORT_FROM_UPPERLEFT MOVE_SHORT_FROM_UPPERRIGHT MOVE_SHORT_FROM_LOWERRIGHT MOVE_SHORT_FROM_LOWERLEFT	LASER_FROM_LEFT LASER_FROM_TOP LASER_FROM_RIGHT LASER_FROM_BOTTOM LASER_FROM_UPPERLEFT LASER_FROM_UPPERRIGHT LASER_FROM_LOWERLEFT LASER_FROM_LOWERRIGHT	STRETCH_FROM_LEFT STRETCH_FROM_UPPERLEFT STRETCH_FROM_TOP STRETCH_FROM_UPPERRIGHT STRETCH_FROM_RIGHT STRETCH_FROM_LOWERRIGHT STRETCH_FROM_BOTTOM STRETCH_FROM_LOWERLEFT
ZOOM_IN_FROM_LEFT ZOOM_IN_FROM_TOP ZOOM_IN_FROM_RIGHT ZOOM_IN_FROM_BOTTOM ZOOM_IN_FROM_CENTER ZOOM_IN_FROM_UPPERLEFT ZOOM_IN_FROM_UPPERRIGHT ZOOM_IN_FROM_LOWERRIGHT ZOOM_IN_FROM_LOWERLEFT	ZOOM_OUT_FROM_LEFT ZOOM_OUT_FROM_TOP ZOOM_OUT_FROM_RIGHT ZOOM_OUT_FROM_BOTTOM ZOOM_OUT_FROM_CENTER ZOOM_OUT_FROM_UPPERLEFT ZOOM_OUT_FROM_UPPERRIGHT ZOOM_OUT_FROM_LOWERRIGHT ZOOM_OUT_FROM_LOWERLEFT	FADE_FROM_LEFT FADE_FROM_TOP FADE_FROM_RIGHT FADE_FROM_BOTTOM FADE_FROM_CENTER FADE_FROM_UPPERLEFT FADE_FROM_UPPERRIGHT FADE_FROM_LOWERLEFT FADE_FROM_LOWERRIGHT FADE_TO_CENTER

Property	Property	Property
ZOOM_IN ZOOM_IN_SMALL ZOOM_IN_SPIRAL ZOOM_OUT ZOOM_OUT_SMALL ZOOM_OUT_SPIRAL	VERTICAL_CHECKERBOARD HORIZONTAL_CHECKERBOARD HORIZONTAL_ROTATE VERTICAL_ROTATE HORIZONTAL_STRETCH VERTICAL_STRETCH	VERTICAL_STRIPES HORIZONTAL_STRIPES VERTICAL_LINES HORIZONTAL_LINES
WAVYLINE_FROM_LEFT WAVYLINE_FROM_TOP WAVYLINE_FROM_RIGHT WAVYLINE_FROM_BOTTOM	SPIRALIN_LEFT SPIRALIN_RIGHT SPIRALOUT_LEFT SPIRALOUT_RIGHT	CLOSE_VERTICAL CLOSE_HORIZONTAL OPEN_VERTICAL OPEN_HORIZONTAL

Shapes contained in presentation documents support special actions (see **Table 27**) by setting the shape's OnClick property (see Table 25).

***Table 27**. Values defined by the com.sun.star.presentation.ClickAction enumeration.*

Property	Description
NONE	No action is performed.
PREVPAGE	Jump to the previous page.
NEXTPAGE	Jump to the next page.
FIRSTPAGE	Jump to the first page.
LASTPAGE	Jump to the last page.
BOOKMARK	Jump to a bookmark specified by the Bookmark property in Table 25.
DOCUMENT	Jump to another document specified by the Bookmark property in Table 25.
INVISIBLE	The object becomes invisible.
SOUND	Play a sound specified by the Bookmark property in Table 25.
VERB	An OLE verb is performed as specified by the Verb property in Table 25. An OLE object supports actions called verbs. An OLE object that displays a video clip might support the verb "play," for example.
VANISH	The object vanishes.
PROGRAM	Execute another program specified by the Bookmark property in Table 25.
MACRO	Execute a Star Basic macro specified by the Bookmark property in Table 25.
STOPPRESENTATION	Stop the presentation.

Conclusion

Impress and Draw documents contain numerous features supporting drawing and graphics in documents. Both types of documents have many drawing and graphical presentation features in common, and Impress documents facilitate the construction of graphic presentations with support for manual or automatic page presentation. Although these documents support the display of bitmapped images, their strength is vector drawings rather than photographic images. The results possible with Impress and Draw documents range from simple to quite complex. Consider this chapter as only a starting point for your explorations on the capabilities of these two document types.

Updates and corrections to this chapter can be found on Hentzenwerke's Web site, **www.hentzenwerke.com**. Click "Catalog" and navigate to the page for this book.