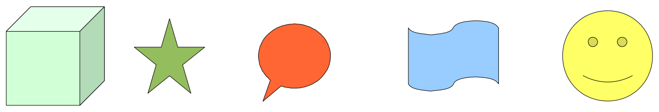
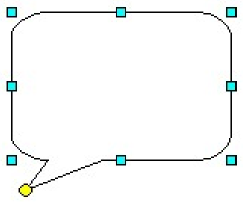
Custom Shapes in OpenOffice.org

One of the great features of OpenOffice.org is the ability to insert complex shapes such as cubes, stars, word balloons, flowchart symbols, and even smiley faces.



These shapes can have a “handle” which lets you change the shape of part of the drawing. In the following figure of the word balloon, the yellow dot is the handle.

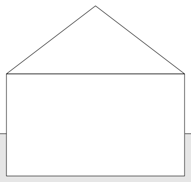


Because of this interactivity, most of the custom shapes are wired in to OpenOffice.org. If you know how they are represented in the OpenDocument XML, though, you can create your own shapes for use in your documents. You can’t easily add them to the drawing toolbar; that would require rebuilding the OpenOffice.org application from source. You have to take the long way around instead:

a) Unzip an existing drawing document into its constituent XML files.
b) Edit the `content.xml` file to add the XML for your custom shape.
c) Re-zip the files to create a new drawing document.
d) Copy and paste the custom shapes from the new document to your other documents.

A Simple Custom Shape

A custom shape can be a fixed set of lines. For our first shape, let’s construct a simple picture of a house:
This will involve putting the following markup into the `content.xml` file:



```
<draw:custom-shape
 draw:style-name="gr2" draw:text-style-name="P1"
 draw:layer="layout"
 svg:width="2.354cm" svg:height="2.249cm"
 svg:x="1.00cm" svg:y="2.50cm">
  <text:p/>
  <draw:enhanced-geometry
    svg:viewBox="0 0 1000 1000"
    draw:text-areas="0 400 1000 1000"
    draw:enhanced-path="M 0 400 L 0 1000 1000 1000 1000 400 Z N
    M 0 400 L 500 0 1000 400 Z N"/>
</draw:custom-shape>
```

Example 1: XML for a simple custom shape

The opening `<draw:custom-shape>` element has these attributes:

`draw:style-name`
The graphic style for the custom shape. In the minimal file, this specifies that the area is to be filled in white. Any attached text is to be centered horizontally and vertically, and the height and width do not grow or shrink to match any added text.

`draw:text-style-name`
The text style for attached text; centered text with no margin or indent.

`draw:layer="layout"`
The drawing goes in the main drawing area (as opposed to controls or dimension lines).

`svg:x` and `svg:y`
The location of the upper left corner of the object.

`svg:width` and `svg:height`
The width and height of the object's "bounding box," which is the rectangular area that completely encloses the object.

The content of the `<draw:custom-shape>` element starts with the attached text; normally a custom shape you create should not have any text, so put in an empty `<text:p>` element.

The text is followed by the main event—the `<draw:enhanced-geometry>` element, which describes how the custom shape is to be drawn.

`svg:viewBox`
This attribute sets the coordinate system for the object. It consists of four integers giving the origin (x and y) and width and height of the coordinate system. You should always set the x and y values to zero. OpenOffice.org sets the width and height to 21600 for custom objects.[1] In this example, we are setting the coordinate system to 1000 units to make the arithmetic easier.

`draw:text-areas`
This attribute gives the left, top, right, and bottom coordinates where text is attached to the object. These coordinates are in terms of the system established by `svg:viewBox`. If you specify a second text area, it is used for vertical text. In this case, we set the text area to the "walls" of the house so that text doesn't appear in the roof area.

`draw:extended-path`
The extended path is based on the path attribute in the Scalable Vector Graphics (SVG) specification. It consists of a series of commands and parameters that describe the lines and curves to be drawing. An extended path consists of one or more sub-paths, consisting of a "move to" command followed by one or more line or curve commands. Here are the commands used in this path:

1 This gives you a coordinate system corresponding to a width and height of 15 inches in twips, where there are 1440 twips per inch.

Command	*Meaning*	*Parameters*
M	Move to a point	*x* and *y* coordinates
L	Line (series of connected lines)	pairs of *x* and *y* coordinates
Z	Close sub-path by drawing a line to the beginning point of sub-path.	*none*
N	End sub-path	*none*

You Try It!

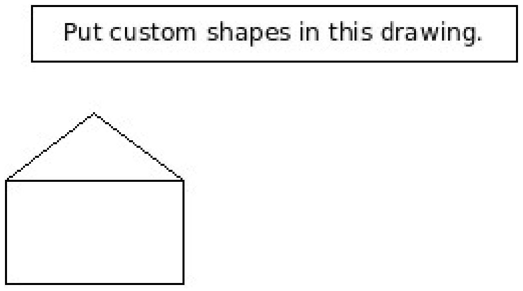
Download the example files that came with this article, downloadable at http://books.evc-cit.info/odbook/custom_shapes_article.zip. Go into the `custom_shapes_odg` directory and insert the XML from Example 1 into file `content.xml` at the place where the markup says:

```
<!-- Your custom shapes go here -->
```

Now zip the files in the folder. On Linux, you use a command like this:

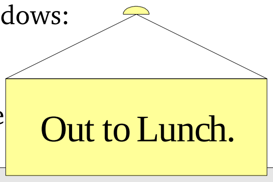
```
zip -r ../test.odg *
```

The ".." makes sure that the resulting file ends up in a directory *other* than the one that hold the unzipped files. Open file `test.odg` in OpenOffice.org. If you've done everything correctly, you should see something like this:



Handles and Modifiers

The shape we just made also looks a lot like those signs you see hanging in store windows:



This example will put a handle on the top of the "string" that lets you adjust its height. Here is the XML for the `<draw:enhanced-geometry element>`, with the new information in red and bold:

```
<draw:enhanced-geometry
  svg:viewBox="0 0 1000 1000"
  draw:text-areas="0 400 1000 1000"
  draw:modifiers="0"
  draw:enhanced-path="M 0 400 L 0 1000 1000 1000 1000 400 Z N
  M 0 400 L 500 $0 1000 400 F Z N
  U 500 $0 50 50 0 180 Z N">
  <draw:handle draw:handle-position="500 $0"
    draw:handle-range-y-minimum="0"
    draw:handle-range-y-maximum="350"/>
</draw:enhanced-geometry>
```

Example 2: Simple shape with handle

The first change is the `draw:modifiers` attribute. Its value is a whitespace-separated list of floating-point numbers. A modifier acts very much like a variable in a programming language. The first entry in the list of numbers is referred to as `$0`, the second as `$1`, the third as `$2`, and so on.

The command `M 0 400 L 500 $0 1000 400` refers to that first variable; instead of always drawing the triangle to the top of the shape area, it draws to whatever value is in `$0`. (When the object first appears, that value is zero, so it draws to the top.)

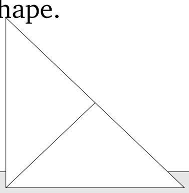
The second change is a minor one; by adding the `F` command before closing a path, the application that displays the object will *not* fill the path. This leaves the triangular area under the “string” empty.

Finally, the enhanced-path uses a new command: `U`, which draws a circle or ellipse. The first two parameters after the command are the *x*- and *y*-coordinates of the center of the circle/ellipse. The next two parameters are the width and height of the circle/ellipse's axes. (In this case, they are equal, so it draws a circle.) The last two parameters are the starting and ending angle in degrees. Notice that the *y* coordinate of the circle’s center is drawn at $0, which, at the beginning, is zero.

The key to interaction is in the `<draw:handle>` element. The `draw:handle-position` gives the initial *x*- and *y*-coordinates of a handle. Because the position uses modifier `$0`, this “links” the handle position with that modifier. As the handle is moved, the value of `$0` will contain the *y*-coordinate of the handle. The `draw:handle-range-y-minimum` and `draw:handle-range-y-maximum` attributes do exactly what their names say: limit the range through which the handle may be moved. Because no *x* range is specified, the handle cannot move in the *x* direction at all.

Equations

The next example is something you might want to create if you were writing a geometry book. This custom object is a triangle whose top vertex can be dragged both horizontally and vertically. As you reshape the triangle, a line will always connect the lower left vertex to the center of the opposite side. A little algebra tells you how to draw that line. If the top vertex, left base, and right base vertices are at coordinates (x_1,y_1), (x_2,y_2), and (x_3, y_3), then the line goes from (x_2,y_2) to $((x_1+ x_3)/2, (y_1+ y_3)/2)$. Here’s the enhanced geometry for this shape.



```
<draw:enhanced-geometry
 svg:viewBox="0 0 1000 1000"
 draw:modifiers="0 0"
 draw:enhanced-path="M 0 1000 L $0 $1 1000 1000 Z N
  M 0 1000 L ?f0 ?f1 F N">
 <draw:equation draw:name="f0" draw:formula="($0 + right) / 2"/>
 <draw:equation draw:name="f1" draw:formula="($1 + bottom) / 2"/>
 <draw:equation draw:name="f2" draw:formula="2*(bottom - top) / 3"/>
 <draw:handle draw:handle-position="$0 $1"
  draw:handle-range-x-minimum="0"
  draw:handle-range-x-maximum="right"
  draw:handle-range-y-minimum="0"
  draw:handle-range-y-maximum="?f2"/>
</draw:enhanced-geometry>
```

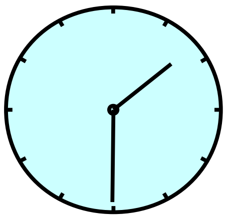
Example 3: Shape using formulas

In addition to the use of modifiers, (`$0` and `$1`), the enhanced path draws the line from coordinate (0,1000) to (`?f0`, `?f1`) with the command `M 0 1000 L ?f0 ?f1`. The leading question mark indicates that the application should use the result of the `<draw:equation>` with the corresponding `draw:name` attribute. If you look at the first `<draw:equation>` element, you see that the result of equation `f0` is the sum of the current *x*-coordinate, in modifier `$0`, ands the right coordinate of the coordinate system[2] divided by two. The result of equation `f1` is the sum of the current *y*-cordinate, in modifier `$1`, and the top of the bounding box, divided by two. Equation `f2` is used in the attribute `draw:handle-range-y-maximum="?f2"` to restrict the *y* movement of the handle so it can go no further than two-thirds of the way down the object area.

Equations may use all the typical arithmetic operations: `+`, `-`, `*`, and `/`. Equations also may include function calls to `abs`, `sqrt`, `sin`, `cos`, `tan`, `atan`, `atan2`, `max`, and `min`. The last two of these functions require two arguments. There's even an `if` function that takes three arguments. If the first argument's value is greater than or equal to zero, then the result of the `if` is the value of the second argument, otherwise it is the value of the third argument.

A Complex Example

Most shops that put up an "Out to Lunch" sign also have a clock telling when they'll be back. The following custom shape has two handles that let you set the hour and minute hand of a clock face:



Here's a quick overview of the trigonometry involved to find out the coordinates for the hour hand, given that the coordinate system is 1000 by 1000 units:

- Get the *x* and *y* distances by subtracting 500 from the handle's coordinates
- The angle of the hour hand is the arctangent of *y/x*
- Calculate the cosine and sine of the angle
- Draw a line from (500, 500) to (500 + length of hour hand * cosine, 500 + length of hour hand * sine)

The calculations for the minute hand differ only in that the minute hand is longer than the hour hand. Here's the XML It starts with the outer circle, the small circle in the center of the clock, and then 12 tick marks around the inner face of the clock.[3] The last part of the path draws the hour hand and minute hand.

```
<draw:enhanced-geometry
 svg:viewBox="0 0 1000 1000"
 draw:text-areas="0 0 1000 1000"
 draw:modifiers="500 150 500 50"
 draw:enhanced-path="U 500 500 500 500 0 360 Z N
 U 500 500 20 20 0 360 F N
 M 1000 500 L 970 500 N M 933 750 L 907 735 N
 M 750 933 L 735 907 N M 500 1000 L 500 970 N
 M 250 933 L 265 907 N M 66 750 L 92 735 N
 M 0 500 L 30 500 N M 66 249 L 92 264 N
 M 249 66 L 264 92 N M 499 0 L 499 30 N
 M 750 66 L 735 92 N M 933 249 L 907 264 N
 M 500 500 L ?hourX ?hourY N M 500 500 L ?minuteX ?minuteY N">
```

Example 4: Custom Shape Clock Path

2 OpenDocument lets you use the reserved words `right`, `left`, `bottom`, and `top`, which stand for the "corners" of the coordinate system. That way, you don't have to remember and copy coordinate numbers.

3 I used a Perl program to generate the coordinates rather than trying to guess them or figure them out by hand.

The equations follow, as described in the preceding pseudocode, finishing up with the handle specifications. The ranges for the hour and minute hand ensure that the hour hand's handle cannot be exteded out as far as the minute hand's, so they don't have to overlap.

```
  <!-- calculations for hour hand -->
  <draw:equation draw:name="cosHour"
    draw:formula="cos(atan2($1-500,$0-500))"/>
  <draw:equation draw:name="sinHour"
    draw:formula="sin(atan2($1-500,$0-500))"/>
  <draw:equation draw:name="hourX"
    draw:formula="500 + 350 * ?cosHour"/>
  <draw:equation draw:name="hourY"
    draw:formula="500 + 350 * ?sinHour"/>

  <!-- calculations for minute hand -->
  <draw:equation draw:name="cosMinute"
    draw:formula="cos(atan2($3-500,$2-500))"/>
  <draw:equation draw:name="sinMinute"
    draw:formula="sin(atan2($3-500,$2-500))"/>
  <draw:equation draw:name="minuteX"
    draw:formula="500 + 450 * ?cosMinute"/>
  <draw:equation draw:name="minuteY"
    draw:formula="500 + 450 * ?sinMinute"/>

  <!-- handle for hour hand -->
  <draw:handle draw:handle-position="$0 $1"
    draw:handle-range-x-minimum="150"
    draw:handle-range-x-maximum="850"
    draw:handle-range-y-minimum="150"
    draw:handle-range-y-maximum="850"/>
  <!-- handle for minute hand -->
  <draw:handle draw:handle-position="$2 $3"
    draw:handle-range-x-minimum="50"
    draw:handle-range-x-maximum="950"
    draw:handle-range-y-minimum="50"
    draw:handle-range-y-maximum="950"/>
</draw:enhanced-geometry>
```

Example 5: Custom-shape clock: Equations and Handles

Further Information

This article has given you the basic information that you need to start creating and using your own custom shapes. There are many other path commands that let you draw arcs, cubic and quadratic Bézier curves, and quarter-ellipses. For a full list, see the OpenDocument specification at http://www.oasis-open.org/committees/documents.php?wg_abbrev=office.

This article is © 2005 by J. David Eisenberg. The content is currently licensed under a Creative Commons License

SOME RIGHTS RESERVED creative commons