ADOBE INDESIGN CS3 SCRIPTING:
WORKING WITH TRANSFORMATIONS IN JAVASCRIPT
Adobe InDesign CS3 Scripting: Working with Transformations in JavaScript

Operations that change the geometry of items on an InDesign page are called **transformations**. Transformations include scaling, rotation, shearing (skewing), and movement (or translation). In Adobe InDesign CS3 scripting, you apply transformations using the `transform` method. This one method replaces the `resize`, `rotate`, and `shear` methods used in previous versions of InDesign.

This document shows you how to transform objects in InDesign CS3 and discusses some of the technical details behind the new transformation architecture.

This document assumes you have a basic working knowledge of InDesign scripting.

Using the Transform Method

The `transform` method requires a **transformation matrix** (`transformationMatrix`) object that defines the transformation or series of transformations to apply to the object. A transformation matrix can contain any combination of scale, rotate, shear, or translate operations, in any order.

The order in which transformations are applied to an object is important. Applying transformations in differing orders can produce very different results.

To transform an object, you follow two steps:

1. Create a transformation matrix.
2. Apply the transformation matrix to the object using the `transform` method. When you do this, you also specify the coordinate system in which the transformation is to take place. For more on coordinate systems, see “Coordinate Spaces” on page 5. In addition, you specify the center of transformation, or transformation origin. For more on specifying the transformation origin, see "Transformation Origin" on page 6.

The following scripting example demonstrates the basic process of transforming a page item. (For the complete script, see TransformExamples.)

```javascript
//Rotate a rectangle "myRectangle" around its center point.
var myRotateMatrix = app.transformationMatrices.add({counterclockwiseRotationAngle:27});
myRectangle.transform(CoordinateSpaces.pasteboardCoordinates, AnchorPoint.centerAnchor, myRotateMatrix);

//Scale a rectangle "myRectangle" around its center point.
var myScaleMatrix = app.transformationMatrices.add({horizontalScaleFactor:.5, verticalScaleFactor:.5});
myRectangle.transform(CoordinateSpaces.pasteboardCoordinates, AnchorPoint.centerAnchor, myScaleMatrix);

//Shear a rectangle "myRectangle" around its center point.
var myShearMatrix = app.transformationMatrices.add({clockwiseShearAngle:30});
myRectangle.transform(CoordinateSpaces.pasteboardCoordinates, AnchorPoint.centerAnchor, myShearMatrix);

//Rotate a rectangle "myRectangle" around a specified ruler point ([72, 72]).
var myRotateMatrix = app.transformationMatrices.add({counterclockwiseRotationAngle:27});
myRectangle.transform(CoordinateSpaces.pasteboardCoordinates, [72, 72], AnchorPoint.topLeftAnchor, myRotateMatrix, undefined, true);
```
Working with Transformation Matrices

A transformation matrix cannot be changed once it has been created, but a variety of methods can interact with the transformation matrix to create a new transformation matrix based on the existing transformation matrix. In the following examples, we show how to apply transformations to a transformation matrix and replace the original matrix. (For the complete script, see TransformMatrix.)

//Scale a transformation matrix by 50% in both horizontal and vertical dimensions.
var myTransformationMatrix = myTransformationMatrix.scaleMatrix(.5, .5);

//Rotate a transformation matrix by 45 degrees.
myTransformationMatrix = myTransformationMatrix.rotateMatrix(45);

//Shear a transformation matrix by 15 degrees.
myTransformationMatrix = myTransformationMatrix.shearMatrix(15);

When you use the rotateMatrix method, you can use a sine or cosine value to transform the matrix, rather than an angle in degrees, as shown in the RotateMatrix script.

//The following statements are equivalent
//(0.25881904510252 is the sine of 15 degrees; 0.96592582628907, the cosine).
myTransformationMatrix = myTransformationMatrix.rotateMatrix(15);
myTransformationMatrix = myTransformationMatrix.rotateMatrix(undefined, 0.96592582628907);
myTransformationMatrix = myTransformationMatrix.rotateMatrix(undefined, undefined, 0.25881904510252);

When you use the shearMatrix method, you can provide a slope, rather than an angle in degrees, as shown in the ShearMatrix script.

//The following statements are equivalent. slope = rise/run--so
//the slope of 45 degrees is 1.
myTransformationMatrix = myTransformationMatrix.shearMatrix(45);
myTransformationMatrix = myTransformationMatrix.shearMatrix(undefined, 1);

You can get the inverse of a transformation matrix using the invertMatrix method, as shown in the following example. (For the complete script, see InvertMatrix.) You can use the inverted transformation matrix to undo the effect of the matrix.

var myRectangle = app.documents.item(0).pages.item(0).rectangles.item(0);
var myTransformationMatrix = app.transformationMatrices.add({counterclockwiseRotationAngle:30, horizontalTranslation:12, verticalTranslation:12});
myRectangle.transform(CoordinateSpaces.pasteboardCoordinates, AnchorPoint.centerAnchor, myTransformationMatrix);
var myNewRectangle = myRectangle.duplicate();
//Move the duplicated rectangle to the location of the original
//rectangle by inverting, then applying the transformation matrix.
myTransformationMatrix = myTransformationMatrix.invertMatrix();
myRectangle.transform(CoordinateSpaces.pasteboardCoordinates, AnchorPoint.centerAnchor, myTransformationMatrix);
You can add transformation matrices using the `catenateMatrix` method, as shown in the following example. (For the complete script, see CatenateMatrix.)

```javascript
var myTransformationMatrixA = app.transformationMatrices.add({counterclockwiseRotationAngle:30});
var myTransformationMatrixB = app.transformationMatrices.add({horizontalTranslation:12, verticalTranslation:12});
var myRectangle = app.documents.item(0).pages.item(0).rectangles.item(-1);
var myNewRectangle = myRectangle.duplicate();
//Rotate the duplicated rectangle.
myNewRectangle.transform(CoordinateSpaces.pasteboardCoordinates, AnchorPoint.centerAnchor, myTransformationMatrixA);
myNewRectangle = myRectangle.duplicate();
//Move the duplicate (unrotated) rectangle.
myNewRectangle.transform(CoordinateSpaces.pasteboardCoordinates, AnchorPoint.centerAnchor, myTransformationMatrixB);
//Merge the two transformation matrices.
myTransformationMatrix = myTransformationMatrixA.catenateMatrix(myTransformationMatrixB);
myNewRectangle = myRectangle.duplicate();
//The duplicated rectangle will be both moved and rotated.
myNewRectangle.transform(CoordinateSpaces.pasteboardCoordinates, AnchorPoint.centerAnchor, myTransformationMatrix);
```

When an object is transformed, you can get the transformation matrix that was applied to it, using the `transformValuesOf` method, as shown in the following script fragment. (For the complete script, see TransformValuesOf.)

```javascript
//Note that transformValuesOf() always returns an array
//containing a single transformationMatrix.
var myTransformArray = myRectangle.transformValuesOf(CoordinateSpaces.parentCoordinates);
var myTransformationMatrix = myTransformArray[0];
var myRotationAngle = myTransformationMatrix.counterclockwiseRotationAngle;
var myShearAngle = myTransformationMatrix.clockwiseShearAngle;
var myXScale = myTransformationMatrix.horizontalScaleFactor;
var myYScale = myTransformationMatrix.verticalScaleFactor;
var myXTranslate = myTransformationMatrix.horizontalTranslation;
var myYTranslate = myTransformationMatrix.verticalTranslation;
var myString = "Rotation Angle: " + myRotationAngle + "\r";
myString += "Shear Angle: " + myShearAngle + "\r";
myString += "Horizontal Scale Factor: " + myXScale + "\r";
myString += "Vertical Scale Factor: " + myYScale + "\r";
myString += "Horizontal Translation: " + myXTranslate + "\r";
myString += "Vertical Translation: " + myYTranslate + "\r";
alert(myString);
```

**Note:** The values in the horizontal- and vertical-translation fields of the transformation matrix returned by this method are the location of the upper-left anchor of the object, in pasteboard coordinates.

---

### Coordinate Spaces

In the transformation scripts we presented earlier, you might have noticed the `CoordinateSpaces.pasteboardCoordinates` enumeration provided as a parameter for the transform.
method. This parameter determines the system of coordinates, or coordinate space, in which the transform operation occurs. The coordinate space can be one of three values:

- **CoordinateSpaces.pasteboardCoordinates** is a coordinate space that uses points as units and extends across all spreads in a document. It does not correspond to InDesign’s rulers or zero point. Transformations applied to objects have no effect on this coordinate space (e.g., the angle of the horizontal and vertical axes do not change).

- **CoordinateSpaces.parentCoordinates** is the coordinate space of the parent of the object. Any transformations applied to the parent affect the parent coordinates; for example, rotating the parent object changes the angle of the horizontal and vertical axes of this coordinate space. In this case, the parent object refers to the group or page item containing the object; if the parent of the object is a page or spread, parent coordinates are the same as pasteboard coordinates.

- **CoordinateSpaces.innerCoordinates** is a coordinate space based on the object itself.

The following script shows the differences between the three coordinate spaces. (For the complete script, see CoordinateSpaces.)

```javascript
var myRectangle = 
app.documents.item(0).pages.item(0).groups.item(-1).rectangles.item(0);
alert("The page contains a group which has been rotated 45 degrees (counterclockwise). The rectangle inside the group was rotated 45 degrees counterclockwise before it was added to the group.\r\nWatch as we apply a series of scaling operations in different coordinate spaces.");
var myTransformationMatrix = 
app.transformationMatrices.add({horizontalScaleFactor:2});
//Transform the rectangle using inner coordinates.
myRectangle.transform(CoordinateSpaces.innerCoordinates, 
AnchorPoint.centerAnchor, myTransformationMatrix);
//Select the rectangle and display an alert.
app.select(myRectangle);
alert("Transformed by inner coordinates.");
//Undo the transformation.
app.documents.item(0).undo();
//Transform using parent coordinates.
myRectangle.transform(CoordinateSpaces.parentCoordinates, 
AnchorPoint.centerAnchor, myTransformationMatrix);
app.select(myRectangle);
alert("Transformed by parent coordinates.");
app.documents.item(0).undo();
//Transform using pasteboard coordinates.
myRectangle.transform(CoordinateSpaces.pasteboardCoordinates, 
AnchorPoint.centerAnchor, myTransformationMatrix);
app.select(myRectangle);
alert("Transformed by pasteboard coordinates.");
```

**Transformation Origin**

The transformation origin is the center point of the transformation. The transformation origin can be specified in several ways:

- **Bounds space:**
  - anchor — An anchor point on the object itself.
    
    `AnchorPoint.centerAnchor`
anchor, bounds type — An anchor point specified relative to the geometric bounds of the object (BoundingBoxLimits.geometricPathBounds) or the visible bounds of the object (BoundingBoxLimits.outerStrokeBounds).

[AnchorPoint.bottomLeftAnchor, BoundingBoxLimits.outerStrokeBounds]

anchor, bounds type, coordinate system — An anchor point specified as the geometric bounds of the object (BoundingBoxLimits.geometricPathBounds) or the visible bounds of the object (BoundingBoxLimits.outerStrokeBounds) in a given coordinate space.

[AnchorPoint.bottomLeftAnchor, BoundingBoxLimits.outerStrokeBounds, CoordinateSpaces.pasteboardCoordinates]

(x, y), bounds type — A point specified relative to the geometric bounds of the object (BoundingBoxLimits.geometricPathBounds) or the visible bounds of the object (BoundingBoxLimits.outerStrokeBounds). In this case, the top-left corner of the bounding box is (0, 0); the bottom-right corner, (1, 1). The center anchor is located at (.5, .5).

[[.5, .5], BoundingBoxLimits.outerStrokeBounds]

(x, y), bounds type, coordinate space — A point specified relative to the geometric bounds of the object (BoundingBoxLimits.geometricPathBounds) or the visible bounds of the object (BoundingBoxLimits.outerStrokeBounds) in a given coordinate space. In this case, the top-left corner of the bounding box is (0, 0); the bottom-right corner, (1, 1). The center anchor is located at (.5, .5).

[[.5, .5], BoundingBoxLimits.outerStrokeBounds, CoordinateSpaces.pasteboardCoordinates]

Ruler space:

(x, y), page index — A point, relative to the ruler origin on a specified page of a spread.

[[72, 144], 0]

(x, y), location — A point, relative to the parent page of the specified location of the object. Location can be specified as an anchor point or a coordinate pair. It can be specified relative to the object's geometric or visible bounds, and it can be specified in a given coordinate space.

[[72, 144], AnchorPoint.centerAnchor]

Transform space:

(x, y) — A point in the pasteboard coordinate space.

[72, 72]

(x, y), coordinate system — A point in the specified coordinate space.

[[72, 72], CoordinateSpaces.parentCoordinates]

((x, y)) — A point in the coordinate space given as the in parameter of the transform method.

[[72, 72]]

The following script example shows how to use some of the transformation origin options. (For the complete script, see TransformationOrigin.)

//Rotate around the duplicated rectangle's center point.
myNewRectangle.transform(CoordinateSpaces.pasteboardCoordinates, AnchorPoint.centerAnchor, myTransformationMatrix);
//Rotate the rectangle around the ruler location [-100, -100].
//Note that the anchor point specified here specifies the page
//containing the point—**not** that transformation point itself.
//The transformation gets the ruler coordinate [-100, -100] based
//on that page. Setting the considerRulerUnits parameter to true makes
Resolving Locations

Sometimes, you need to get the location of a point specified in one coordinate space in the context of another coordinate space. To do this, you use the `resolve` method, as shown in the following script example. (For the complete script, see ResolveLocation.)

```javascript
var myPageLocation = myRectangle.resolve([[72, 72], AnchorPoint.topRightAnchor], CoordinateSpaces.pasteboardCoordinates, true);
```

Transforming Points

You can transform points as well as objects, which means scripts can perform a variety of mathematical operations without having to include the calculations in the script itself. The ChangeCoordinates sample script shows how to draw a series of regular polygons using this approach:

```javascript
//General purpose routine for drawing regular polygons from their center point.
function myDrawPolygon(myParent, myCenterPoint, myNumberOfPoints, myRadius, myStarPolygon, myStarInset){
  var myTransformedPoint;
  var myPathPoints = new Array;
  var myPoint = [0,0];
  if(myStarPolygon == true){
    myNumberOfPoints = myNumberOfPoints * 2;
  }
  var myInnerRadius = myRadius * myStarInset;
  var myAngle = 360/myNumberOfPoints;
  var myRotateMatrix = app.transformationMatrices.add({
    counterclockwiseRotationAngle:myAngle});
  var myOuterTranslateMatrix = app.transformationMatrices.add({
    horizontalTranslation:myRadius});
  var myInnerTranslateMatrix = app.transformationMatrices.add({
    horizontalTranslation:myInnerRadius});
  for (var myPointCounter = 0; myPointCounter < myNumberOfPoints; myPointCounter ++){
    //Translate the point to the inner/outer radius.
    if ((myStarInset == 1)||(myIsEven(myPointCounter)==true)){
      myTransformedPoint = myOuterTranslateMatrix.changeCoordinates(myPoint);
    } else{
      myTransformedPoint = myInnerTranslateMatrix.changeCoordinates(myPoint);
    }
    myTransformedPoint = myRotateMatrix.changeCoordinates(myTransformedPoint);
    myPathPoints.push(myTransformedPoint);
    myRotateMatrix = myRotateMatrix.rotateMatrix(myAngle);
  }
  //Create a new polygon.
  var myPolygon = myParent.polygons.add();
  //Set the entire path of the polygon to the array we've created.
  myPolygon.paths.item(0).entirePath = myPathPoints;
```
// If the center point is somewhere other than [0,0],
// translate the polygon to the center point.
if((myCenterPoint[0] != 0)||((myCenterPoint[1] != 0))){
    var myTranslateMatrix = app.transformationMatrices.add({
        horizontalTranslation:myCenterPoint[0],
        verticalTranslation:myCenterPoint[1]});
    myPolygon.transform(CoordinateSpaces.pasteboardCoordinates,
        AnchorPoint.centerAnchor, myTranslateMatrix);
}

// This function returns true if myNumber is even, false if it is not.
function myIsEven(myNumber){
    var myResult = (myNumber%2)?false:true;
    return myResult;
}

You also can use the changeCoordinates method to change the positions of curve control points, as shown in the FunWithTransformations sample script.

Transforming Again

Just as you can apply a transformation or sequence of transformations again in the user interface, you can do so using scripting. There are four methods for applying transformations again:

- transformAgain
- transformAgainIndividually
- transformSequenceAgain
- transformSequenceAgainIndividually

The following script fragment shows how to use transformAgain. (For the complete script, see TransformAgain.)

var myRectangle = myPage.rectangles.item(0);
var myBounds = myRectangle.geometricBounds;
var myX1 = myBounds[1];
var myY1 = myBounds[0];
var myRectangleA = myPage.rectangles.add({geometricBounds:[myY1-12, myX1-12,
myY1+12, myX1+12]});
var myTransformationMatrix =
    app.transformationMatrices.add({counterclockwiseRotationAngle:45});
myRectangleA.transform(CoordinateSpaces.pasteboardCoordinates,
    AnchorPoint.centerAnchor, myTransformationMatrix);
var myRectangleB = myRectangleA.duplicate();
myRectangleB.transform(CoordinateSpaces.pasteboardCoordinates,[[0,0],
AnchorPoint.topLeftAnchor], myTransformationMatrix, undefined, true);
var myRectangleC = myRectangleB.duplicate();
myRectangleC.transformAgain();
var myRectangleD = myRectangleC.duplicate();
myRectangleD.transformAgain();
var myRectangleE = myRectangleD.duplicate();
myRectangleE.transformAgain();
var myRectangleF = myRectangleE.duplicate();
myRectangleF.transformAgain();
var myRectangleG = myRectangleF.duplicate();
myRectangleG.transformAgain();
var myRectangleH = myRectangleG.duplicate();
myRectangleH.transformAgain();
myRectangleB.transform(CoordinateSpaces.pasteboardCoordinates,
AnchorPoint.centerAnchor, myTransformationMatrix);
myRectangleD.transformAgain();
myRectangleF.transformAgain();
myRectangleH.transformAgain();