Coordinate Spaces & Transformations

in InDesign CS4 – CC | Chapters 1-3

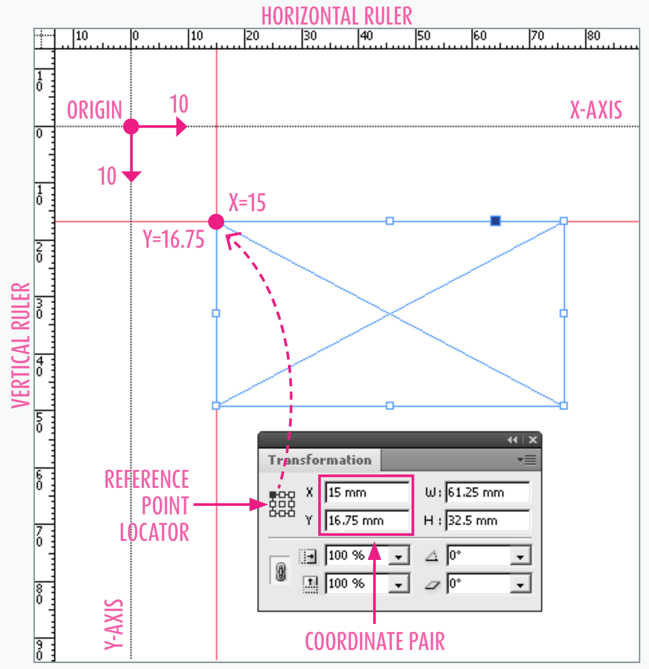
SCRIPTING SECRETS

1. Key Concepts

Dealing with coordinate spaces and transformation matrices is one of the most obscure and underappreciated exercises in InDesign scripting and programming. The fault mainly lies with Adobe documentation, especially the Scripting DOM reference, which does not clearly explain the topic and some of its essential keys. This document attempts to shed some light on the beast.

2D Coordinate Systems

Within InDesign, the geometric location of a point is defined in terms of coordinates within a two-dimensional space. A coordinate is a pair of numbers (usually denoted x and y) that locate a point



X 15 mm W: 61.25 mm
Y 16.75 mm H: 32.5 mm
100 % 0°
100 % 0°

Figure 1. The ruler coordinate system makes it easy to check locations and measurements from the application interface.

relative to a given *origin*, according to the orientation of two given *axes* and with respect to the length of some *units* along each axis. These three parameters form a 2D COORDINATE SYSTEM.

InDesign handles multiple coordinate systems. A given location in the layout can be expressed by different coordinate pairs depending on the system. Users can easily experience how coordinates and measurements vary when playing with on-screen rulers, changing measurement units, moving the origin or the Reference Point (cf. **Figure 1**). The actual position and size of layout elements do not change, but both the Control panel, the Info panel and the Transform panel accordingly update the coordinates of the objects and other related values such as width and height of page items. Special display settings (e.g. *Show Content Offset* and *Dimensions Include Stroke Weight*) also affect how measurements display in the application interface.

Affine Maps

Every coordinate system is somewhat *arbitrary*. Whatever the origin, the units and the orientations of

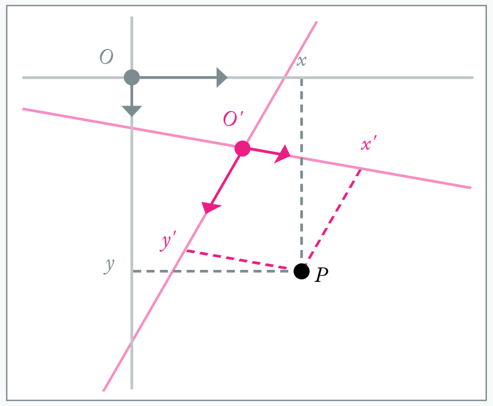


Figure 2. The same location **P** can be expressed by different coordinate pairs — (x, y) vs. (x′, y′) — depending on the coordinate system we consider.

the axes, we can point out to the same geometric point, or path, by simply adjusting the coordinates to the desired coordinate system (cf. **Figure 2**). In other words, we can *convert* any coordinate pair from one system into another.

Fortunately such *conversion* is easy to describe in mathematical terms (no matter what coordinate systems or points we are considering). The functional relationship between two coordinate systems is known as an AFFINE MAP. An important property of any affine map is that it can always be entirely defined by an array of six real numbers. (We'll talk more about these a bit later.)

Relative Locations & Inner Space

When rendering graphics, paths and frames, InDesign needs to address their final locations according to various parameters. Some are *extrinsic* (e.g. screen resolution, parent window size, zoom factor, scrolling state), other are *intrinsic* in that they specify the inner geometry of the layout items and their relationships within the publication.

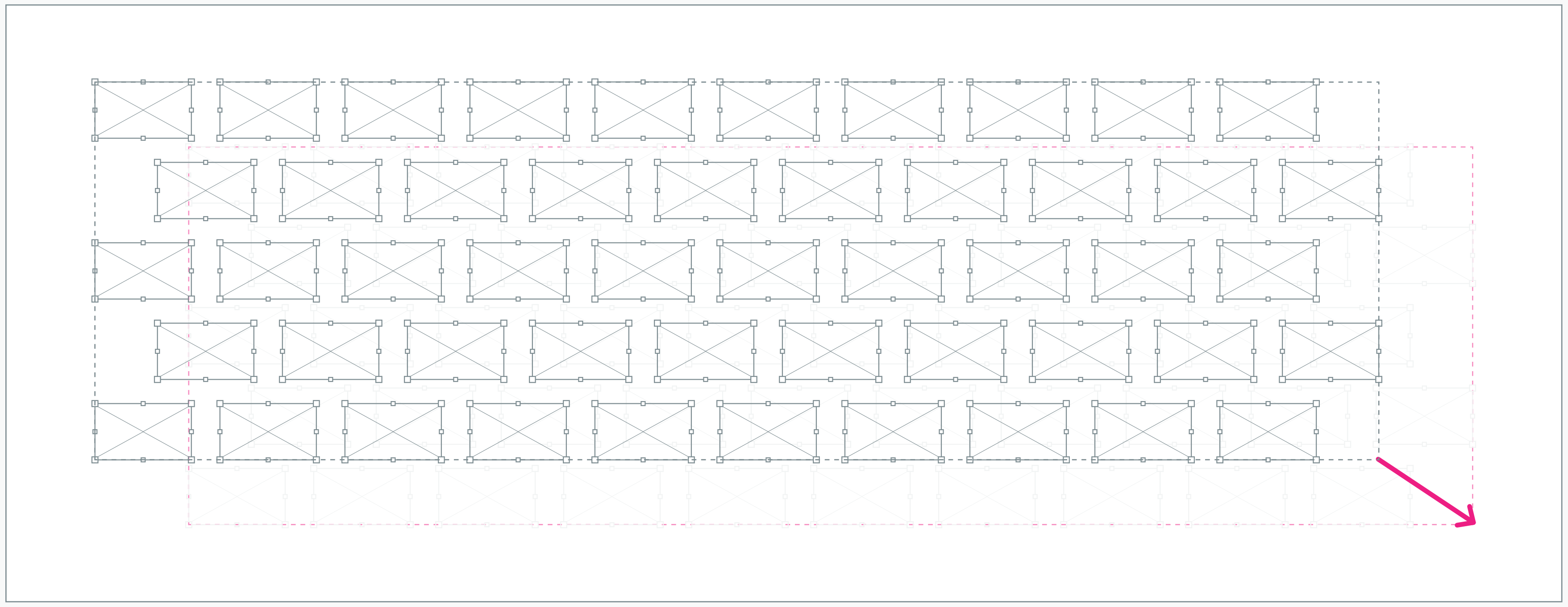


Figure 3. When a group of 50 rectangles is moving, InDesign doesn't need to update the location and the inner path points of every child item. Instead the group tells its parent—typically, a spread container—that its location has changed. Technically, this is done by simply adjusting the affine map attributes that connect the group's inner space to its parent space. This way the child objects are not modified at all (as their respective position relative to the group remains unchanged).

As the whole document relies on a hierarchical structure that involves a lot of dependencies, geometric constraints and nested elements, any change made at any level is likely to affect the location of *every* child object. Consider what is happening when the user moves a group formed by 50 rectangles (cf. **Figure 3**). Does this mean that every individual rectangle is in some way rewritten so that its inner path fits the new location? Of course not!

To accurately manage such operations, InDesign stores locations and geometric data through a hierarchical model that exactly reflects how layout objects are nested or linked. In this model, each component (including groups, pages, spreads, and even on-screen views) has a virtual coordinate system (usually referred to as its INNER COORDINATE SPACE) which is associated to an array of six numbers that specify *how to convert any coordinate pair from that inner system into the parent's coordinate system* (see **Affine Maps** above).

That's it! Now when the user is moving a group of page items, InDesign only has to change the map attributes that connect the group to its parent spread (in terms of coordinate systems). So there is no need to update children' locations.

Transformations Only Re-Map Coordinates

From the user's perspective, all goes as if page items themselves were *transformed*. We can make them smaller or wider, we can rotate them, shear them, etc. Anyway, the most important rule to learn regarding transformations is that *a transformation never alters the actual geometry of graphics objects.*

In other words, whatever the transformations we apply, every path point that underlies the target object will keep its intrinsic location *in the inner coordinate space* of that object.

In InDesign a transformation *only* affects the relationship between two coordinate systems. "Transforming an object" should be understood as changing in some way the affine map that *translates* the inner coordinate space of this object into its parent's coordinate space.

This definition may seem quite abstract, so let me take an example. Say you want to integrate a heart shape vector in your layout. At some point a page item is created storing *only* the geometry of this object within its inner space (cf. **Figure 4**). Note that the object

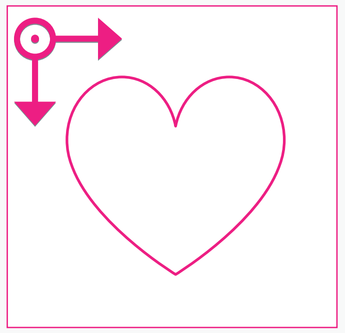


Figure 4. Inner space of a basic page item. The heart shape vector represents the object's geometry, made up by a set of path points. Here the location of each point is expressed relative to the inner coordinate system.

is not *visible* yet, as we didn't specify how and where it is supposed to take place in the publication. To render the page item, InDesign needs to target a DEVICE SPACE, that is, an imageable area where layout contents ultimately appear, such as an on-screen window or a printed page.

Let's not go into details and just assume that a device space is in turn a coordinate system which has the ability to draw graphics. Now, suppose that the heart shape is a direct child of the device along the object hierarchy. The child then can *convert* any coordinate pair from its inner coordinate system to the device coordinate system—since this is the purpose of the affine map associated to the child.

Figure 5 shows how the device and the page item interact via the affine map (M). As you can see, the device can draw the entire shape in its own coordinate space without altering the inner geometry of the page item: any required coordinate pair is just *re-mapped through* M.

Now remember that M is somehow encoded as a sequence of six numeric attributes. It is easy to understand that changing these attributes will cause the device to redraw the heart shape as if taking place in a different coordinate system. **Figure 6** shows the result of such "transformation"—making the shape look smaller in the device space. This is just an example of *scaling* the page item.

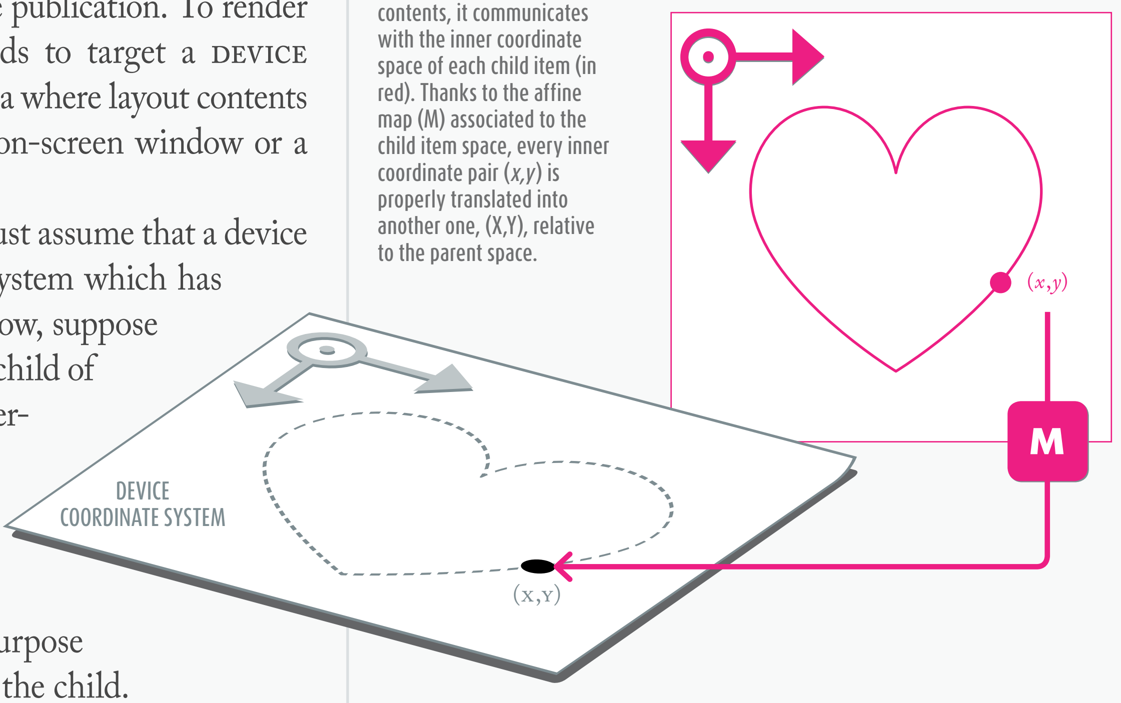


Figure 5. When the device (in gray) needs to draw its contents, it communicates with the inner coordinate space of each child item (in red). Thanks to the affine map (M) associated to the child item space, every inner coordinate pair (x,y) is properly translated into another one, (X,Y), relative to the parent space.

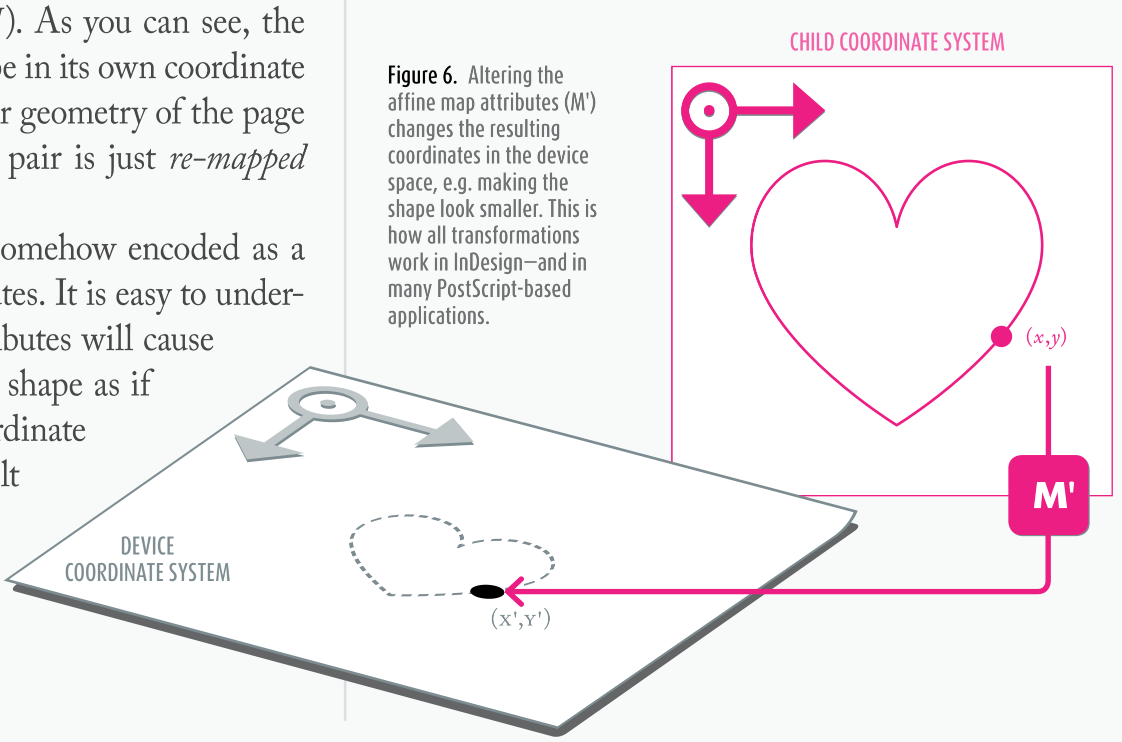


Figure 6. Altering the affine map attributes (M') changes the resulting coordinates in the device space, e.g. making the shape look smaller. This is how all transformations work in InDesign—and in many PostScript-based applications.

Maps, Transformations and Matrices

Before we go any further, there is an important point to highlight: the only *internal* difference between **Figure 5** and **Figure 6** above, is the change from M to M'. Not only the inner geometry of the heart shape but also the respective coordinate systems are preserved[1]. This means that all about transformations regards affine maps, and only affine maps … until the output device space is reached.

As said earlier, an affine map M is based on a sequence of six numbers. Although it is not vital to understand how these attributes operate behind the scenes, an essential key is that, in InDesign, any transformation T is encoded through a sequence of six numbers too. To put it differently: transformations and affine maps are substantially encoded the same way and can play the same role. In mathematical terms, applying T to M amounts to calculate a kind of *product*:

$$M' = M \times T$$

where M' refers to the resulting map (once the transformation is done). Of course the above terms are not real numbers. Each in fact is a 3-by-3 MATRIX that encapsulates the corresponding sequence of attributes.

1. Some authors take a different approach and consider that affine transformations actually affect coordinate systems; other argue that graphics objects themselves undergo transformations (rather than coordinate systems). Either point of view may be self-consistent, depending on how the underlying concepts are defined and developed. Anyway, my personal approach is that a transformation does only change affine maps, and that a affine map connects two coordinate systems.

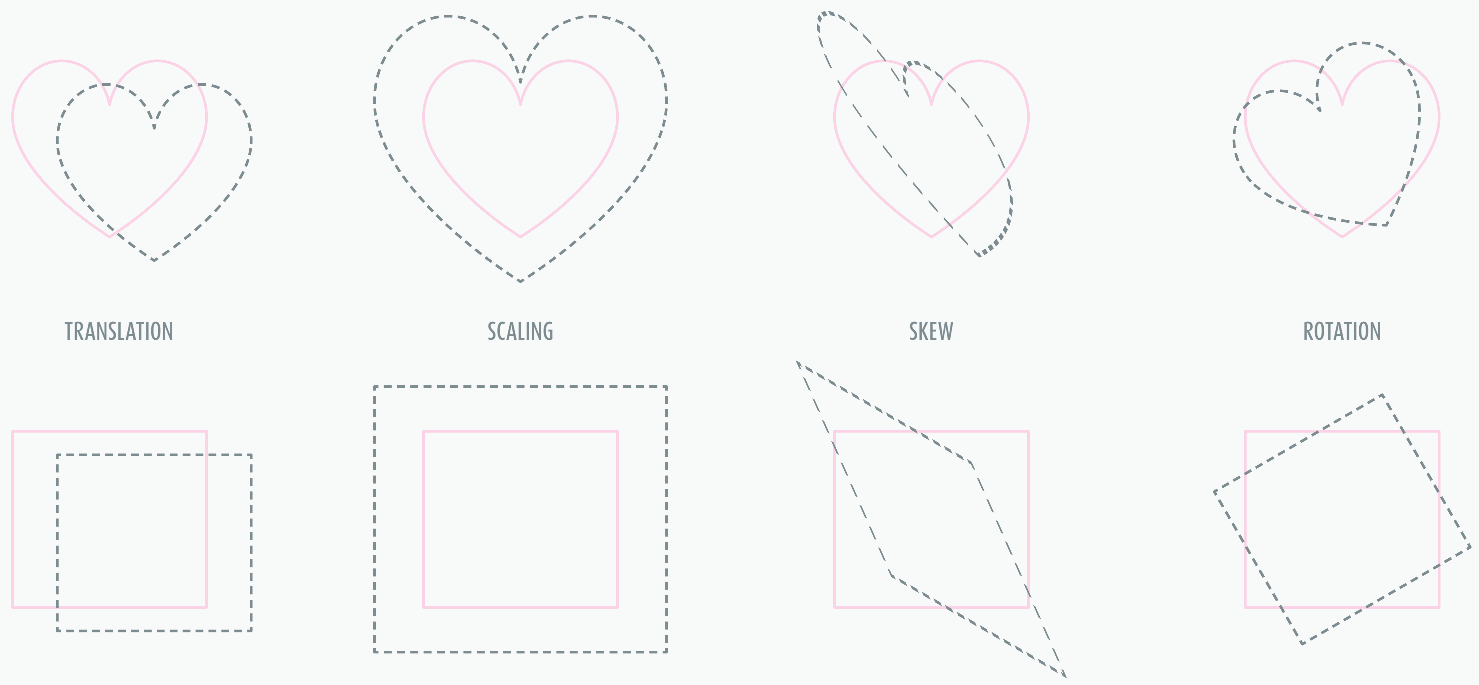


Figure 7. Examples of basic transformations applied to an heart shape (top) and to a rectangle (bottom). The initial geometry is shown in light red; the resulting shapes are shown dotted. (If you mentally replace rectangles with coordinate system bases, you get the same picture in terms of affine mapping.)

The reason why transformations can be encoded as affine maps, and vice-versa, is that InDesign only supports *affine transformations* of the plane, a group of geometric transformations that both preserve collinearity, ratios of distance *and* parallel lines[2]. These are: TRANSLATION, SCALING, ROTATION, REFLECTION, SHEAR, and any combination thereof. **Figure 7** shows basic examples.

In the InDesign SDK, scripting DOM and IDML terminologies, affine maps are often referred to as page item transform(ation) states:

— `ITransform` is the *"transformation matrix that maps from the inner coordinate space to the parent coordinate space."* (SDK)

— `PageItem.transformValuesOf`: *"After an object is transformed, you can get the transformation matrix that was applied to it, using the transformValuesOf() method."* (Scripting DOM)

— `ItemTransform`: *"The relationship of the inner coordinates of the child element to the coordinate system of the <Spread> element (or other parent element) is defined by the ItemTransform attribute of the child element."* (IDML specification)

These definitions all refer to the *current* affine map from a coordinate space to its parent's space, which at a given time is stored as a property of the component. Transformations themselves are *temporary* operands, used in methods that cause an affine map to change. In all cases, however, attributes or arguments are implemented, stored and/or processed as matrix stuctures or similar. For this reason, the term TRANSFORMATION MATRIX in Adobe documentation may refer to either an affine map (the *state*) or a transformation (the *action*).

2. An affine transformation always takes a parallelogram to a parallelogram; and, given two parallelograms P and P', there is always an affine transformation that takes P to P'. This strictly equates to the concept of affine mapping. By contrast, perspective projections are not affine transformations.

Matrix Patterns

By convention every transformation matrix is written:

$$\begin{bmatrix} a & b & 0 \\ c & d & 0 \\ e & f & 1 \end{bmatrix}$$

where a, b, c, d, e, f are the numeric attributes of the affine transformation, or map. To *apply* the matrix to a given (x, y) coordinate pair, we calculate the following product:

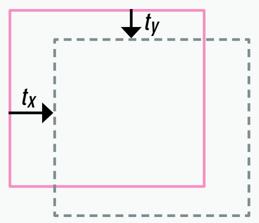
$$[\ x \ \ y \ \ 1\ ] \times \begin{bmatrix} a & b & 0 \\ c & d & 0 \\ e & f & 1 \end{bmatrix}$$

which leads to

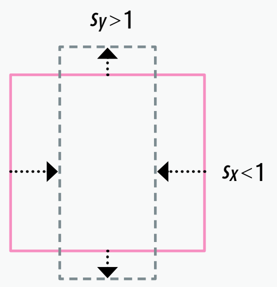
$$[\ xa{+}yc{+}e \quad xb{+}yd{+}f \quad 1\ ].$$

1. Key Concepts

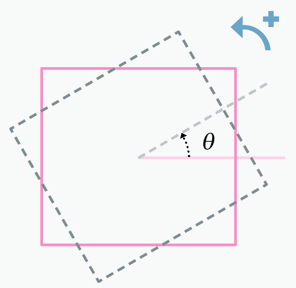
IDENTITY. Strictly identical mapping from the source space to the destination space.



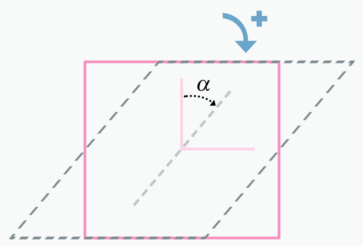
TRANSLATION. Takes any (x, y) pair to $(x+t_x, y+t_y)$. Allows to 'reposition' the object in the destination space.



SCALING. Takes any (x, y) pair to $(x \times s_x, y \times s_y)$. If $s_x = s_y$, this results in a uniform (i.e. homothetic) scaling.



ROTATION. Takes any (x, y) pair to: $(x \times \cos\theta + y \times \sin\theta,$ $-x \times \sin\theta + y \times \cos\theta)$ where θ is the counter-clockwise rotation angle.



SHEAR (along the *x*-axis). Takes any (x, y) pair to: $(x - y \times \tan\alpha,\ y)$ where α is the clockwise shear angle relative to the *y*-axis.

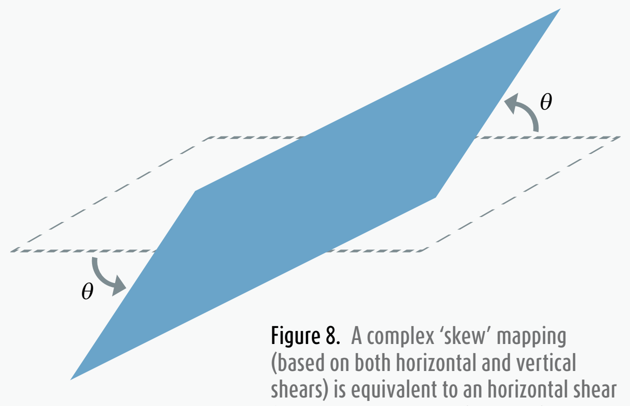


Figure 8. A complex 'skew' mapping (based on both horizontal and vertical shears) is equivalent to an horizontal shear followed by the appropriate rotation.

Ignoring the third dimension, the resulting coordinate pair is finally defined by:

$$(x', y') = (xa + yc + e,\ xb + yd + f).$$

But the above presentation is somewhat artificial. We can see that the 3D matrix only allows to compound a 2D *linear* transformation, based on the attributes a, b, c, d, with a 2D translation, based on the $[\,e\ \ f\,]$ vector. In two-dimensional terms, we would have as well:

$$[\,x'\ \ y'\,] = \underbrace{[\,x\ \ y\,] \times \begin{bmatrix} a & b \\ c & d \end{bmatrix}}_{\text{LINEAR TRANSFORMATION}} + \underbrace{[\,e\ \ f\,]}_{\text{TRANSLATION}}$$

Again, this evidences that any transformation matrix is made up to perform an affine mapping, i.e. a linear transformation *modulo* a translation.

The linear component—a, b, c, d—typically allows to apply scaling, flipping, rotation, or skew, around the origin of the input coordinate system, while the translation component—e, f—allows to reposition the result within the destination space. For this reason, the e and f attributes are often written t_x and t_y instead.

Here are the most common matrix patterns:

IDENTITY: $\begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$

TRANSLATION: $\begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ t_x & t_y & 1 \end{bmatrix}$

SCALING: $\begin{bmatrix} s_x & 0 & 0 \\ 0 & s_y & 0 \\ 0 & 0 & 1 \end{bmatrix}$

ROTATION: $\begin{bmatrix} \cos\theta & -\sin\theta & 0 \\ \sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix}$

SHEAR: $\begin{bmatrix} 1 & 0 & 0 \\ -\tan\alpha & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$

Note that the above *rotation* and *shear* formulas fit the default orientation of the *y*-axis in InDesign (i.e. values increasing from up to bottom) and with respect to the sign of either the 'rotation angle' or the 'shear angle' as shown in the GUI as well as in transformation settings. These may differ from academic patterns.

In particular, for skew mapping parallel to the *y*-axis, InDesign will use something like:

$$\begin{bmatrix} 0 & -1 & 0 \\ 1 & \tan\beta & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

where β denotes the shear angle relative to the *x*-axis. It is not difficult to see that this matrix results from applying a 90° rotation after the SHEAR pattern.[3] Indeed, InDesign treats any skew mapping as a combination of a *shear-along-x* and a *rotation* (see **Figure 8**).

3. Viz: $\begin{bmatrix} 1 & 0 \\ -\tan\beta & 1 \end{bmatrix} \times \begin{bmatrix} 0 & -1 \\ 1 & 0 \end{bmatrix}$ (having $\cos 90° = 0$ and $\sin 90° = 1$).

Matrix Product

Let M and M' be two matrices. No matter their respective attributes, the product $M \times M'$ always results in a new matrix. Technically:

$$\begin{bmatrix} a & b & 0 \\ c & d & 0 \\ e & f & 1 \end{bmatrix} \times \begin{bmatrix} a' & b' & 0 \\ c' & d' & 0 \\ e' & f' & 1 \end{bmatrix} = \begin{bmatrix} aa'+bc' & ab'+bd' & 0 \\ ca'+dc' & cb'+dd' & 0 \\ ea'+fc'+e' & eb'+fd'+f' & 1 \end{bmatrix}$$

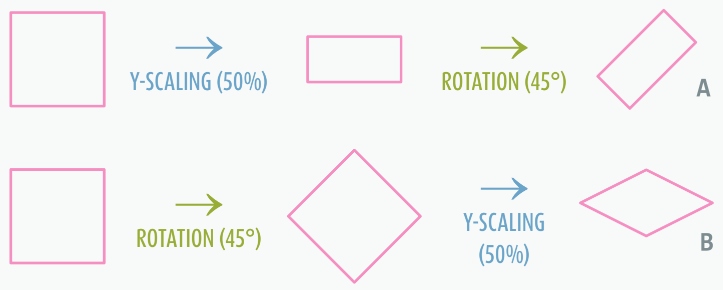
where a, b, c, d, e, f (resp. a', b', c', d', e', f') are the numeric attributes of M (resp. M'). These complex calculations are of little interest though. What is important is to get the *meaning* of the product: $M \times M'$ reflects the combination of the two transformations, that is, the M-transformation *followed* by the M'-transformation.

For example, suppose that M represents some SCALING and M' represents some ROTATION. Then, $M \times M'$ is the matrix that encodes the global transformation (SCALING, *then* ROTATION).

At any level, whenever InDesign *applies* a transformation, it simply computes the product of an existing matrix (map or transformation) by an incoming matrix. This way, successive transformations applied to an object—in fact, to its affine map—have not to be stored by themselves. The map is simply updated as the result of a matrix product, and its new attributes represent the whole effect of all transformations it has undergone from its creation.

No matter how, and how much, you multiply transformations on an object, the result is always as simple as a unique transformation, entirely described by six numbers, which finally encodes the resulting affine map.

However, during the computation, the order of transformations does matter, as $M \times M'$ is ordinary not equivalent to $M' \times M$. It is easy to check visually that scaling first, then rotating, is not the same as performing rotation before scaling:



In other words:

SCALING × ROTATION ≠ ROTATION × SCALING

This inequality can be generalized to most matrix products.

InDesign's Canonical Transformation Order (S×H×R×T)

But, as an experienced user, you may have noticed that the shape **A** (above) is easier to obtain than the shape **B**. Indeed, atomic transformations (TRANSLATION, SCALING, ROTATION, and SHEAR) cannot be applied in an arbitrary order to an object *via* the GUI—although this could be done through scripting, as we shall see later.

We must highlight here that every transformation matrix M can be decomposed as a product of four atomic matrices, in the following order:

$$M = S \times H \times R \times T,$$

where S is a SCALING matrix, H a SHEAR matrix, R a ROTATION matrix and T a TRANSLATION matrix (*see the previous page for the related patterns*). This canonical decomposition is unique, and InDesign uses it as an internal mechanism to link any transformation matrix to a set of user-friendly attributes in the interface. Developers will also access those attributes from the `TranformationMatrix` object; namely: horizontal and vertical scale factor (s_x, s_y), clockwise shear angle (α), counterclockwise rotation angle (θ), horizontal and vertical translation (t_x, t_y).

While this fact is generally overshadowed in the literature, it is of the utmost importance to understand the underlying principle before you deal with transformations. Given the matrix values (a, b, c, d, e, f), InDesign automatically resolves and maintains the correlated S×H×R×T scheme so that one *always* has the relation:

$$\begin{array}{ccccccccc} \begin{bmatrix} a & b & 0 \\ c & d & 0 \\ e & f & 1 \end{bmatrix} & = & \begin{bmatrix} s_x & 0 & 0 \\ 0 & s_y & 0 \\ 0 & 0 & 1 \end{bmatrix} & \times & \begin{bmatrix} 1 & 0 & 0 \\ -\tan\alpha & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} & \times & \begin{bmatrix} \cos\theta & -\sin\theta & 0 \\ \sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix} & \times & \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ t_x & t_y & 1 \end{bmatrix} \\ M & = & \text{SCALING} & & \text{SHEAR} & & \text{ROTATION} & & \text{TRANSL.} \end{array}$$

This decomposition has many advantages. First, since the translation is the final term of the product, the (t_x, t_y) components are not involved with previous calculations and remains independent, so $(e, f) = (t_x, t_y)$. The remaining equation has a pure 2D-linear form:

$$\begin{bmatrix} a & b \\ c & d \end{bmatrix} = \begin{bmatrix} s_x & 0 \\ 0 & s_y \end{bmatrix} \times \begin{bmatrix} 1 & 0 \\ -\tan\alpha & 1 \end{bmatrix} \times \begin{bmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{bmatrix}$$

Here we can see that the *determinant* of both the shear and the rotation matrices is 1 (which reflects the fact that these transformations preserves the area). So the determinant of the entire matrix is simply $s_x \times s_y$ ($= a \times d - b \times c$). This signed value represents the area scale factor, noting that a negative number indicates whether the shape is mirrored.

Also, we can distinguish the neutral parameters, viz. the IDENTITY matrix for each term: $(s_x, s_y) = (1,1)$ [no scaling]; $\alpha = \emptyset$ [no shear]; $\theta = \emptyset$ [no rotation].

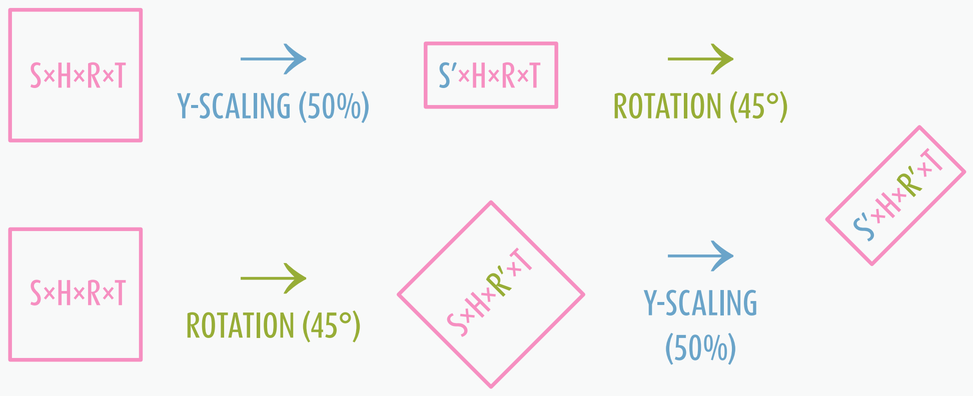
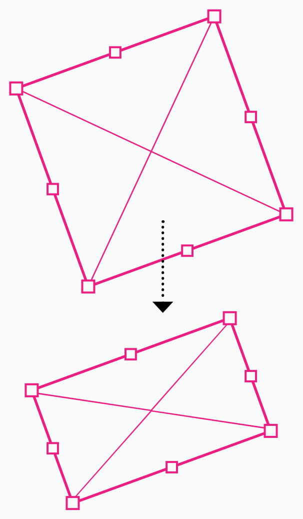


Figure 9.
Object transformations specified from the GUI are order-insensitive, because at each step InDesign only has to update a single matrix component without altering the canonical decomposition order (S×H×R×T).



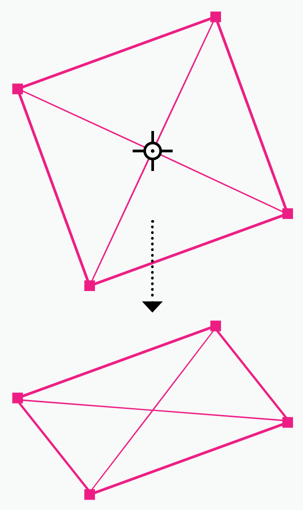


Figure 10a.
Changing the Y-scale of the selected object simply results in changing the SCALING component of the affine map. This is a TRANSFORMATION.

Figure 10b.
Manually selecting all path points with the Direct Selection tool then changing the Y-scale results in actually moving the points "along the transformation." This is a DEFORMATION.

A matrix is *invertible* iff its determinant is not zero, i.e. $s_x \times s_y \neq 0$—here you see why InDesign does not allow you to scale anything at 0%! In InDesign, any (valid) transformation matrix is invertible.

Now, let's consider a document page item. Its affine map (M), in its current state, can be expressed as $M = S \times H \times R \times T$ (keeping the above notation). Suppose that the user changes the shear angle from the Transform panel. What does this mean in terms of re-mapping?

You could think that some matrix is created, say U, *encoding the shear transformation specified by the user*, and that M is changed to $M \times U$.

That would make perfect sense, but that's not what happens. InDesign does *not* change M to $M \times U$, because it does *not* apply any shear matrix to the existing map! Instead, it only updates the existing shear component (H) so that it now reflects the desired shear angle. In other words, H just becomes H', and M therefore becomes $S \times H' \times R \times T$.[4]

The same thing happens when you change the scaling of a page item on which a rotation is already applied. The application doesn't compute $M \times U_{scaling}$, it just changes the map to $S' \times H \times R \times T$, where S' denotes the new scaling component. This is the reason why either the order *scaling-then-rotation*, or *rotation-then-scaling*, leads to the same result in the GUI (see **Figure 9**).

Transformation vs. Deformation

Since a transformation in itself only affects an affine map and does not impact the actual geometry of the shape, it is relevant for developers to draw a formal distinction between TRANSFORMING and DEFORMING an object. The former only deals with re-mapping coordinates *via* transformation matrices, the latter regards actual moves of path points relative to the inner space.

InDesign provides various ways to *deform* a path, that is, the geometry of a spline item. In this regard, an interesting operation is to select a set of path points using the Direct Selection tool (A) and to "apply a transformation" of whatever kind. What does happen under the hood? InDesign *does not transform* the object in the sense of updating its affine map. Instead, it moves the points along the transformation, as shown in **Figure 10b**. In this very specific case, although the system internally performs a transformation on the set of selected path points, no trace of this operation is stored in the transformation matrix.

Hierarchical Mapping

As said earlier, every new object along the document hierarchy—including groups, pages and spreads—has its own coordinate space bound to the coordinate space of the parent object *via* an affine map. Technically, each of those affine maps is encoded in a single transformation matrix. This is the way all graphic objects are positioned relative to each other.

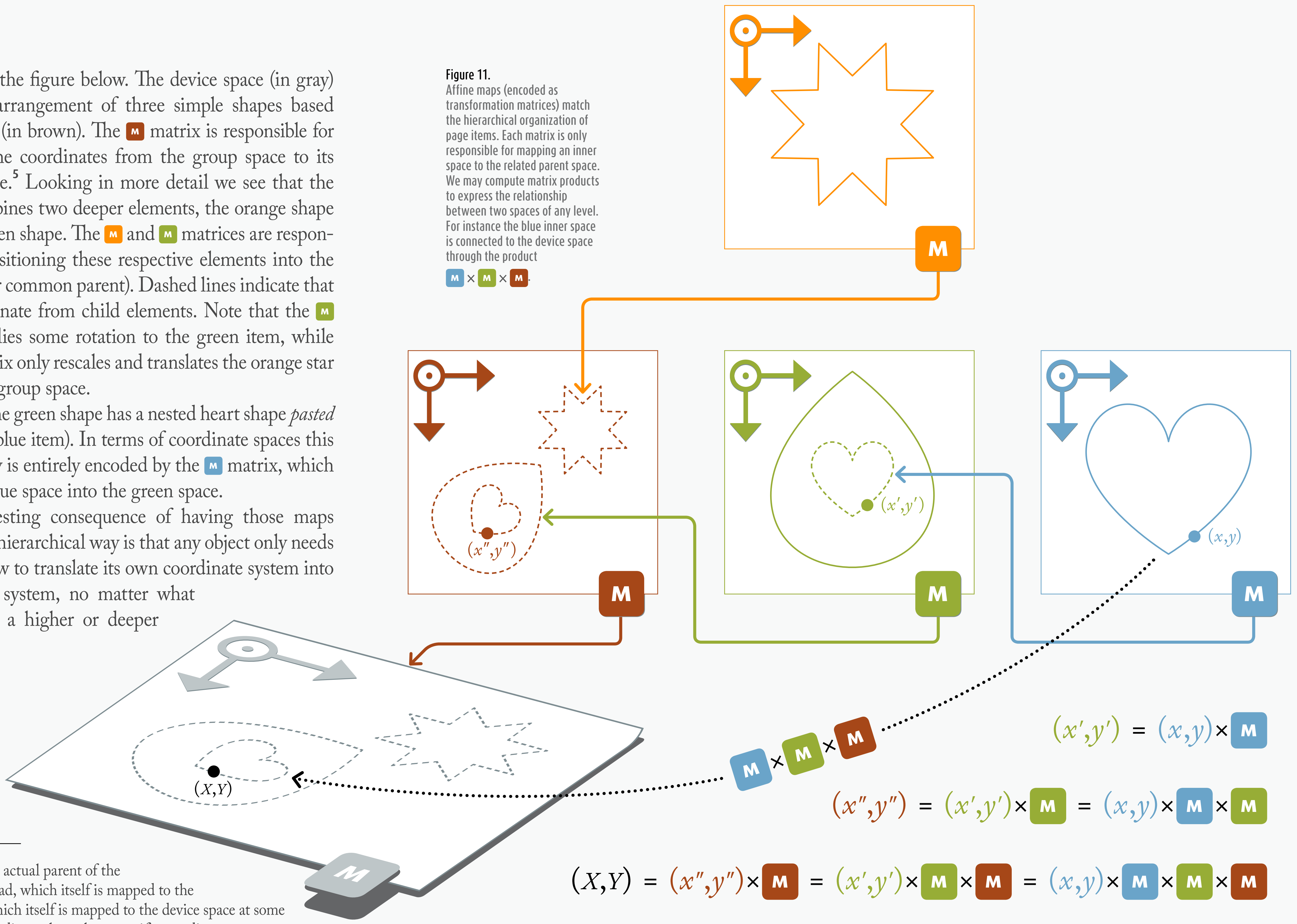
4. Of course the six matrix values (a, b, c, d, tx, ty) are recalculated accordingly.

Consider the figure below. The device space (in gray) shows an arrangement of three simple shapes based on a group (in brown). The M matrix is responsible for mapping the coordinates from the group space to its parent space.[5] Looking in more detail we see that the group combines two deeper elements, the orange shape and the green shape. The M and M matrices are responsible for positioning these respective elements into the group (their common parent). Dashed lines indicate that shapes emanate from child elements. Note that the M matrix applies some rotation to the green item, while the M matrix only rescales and translates the orange star within the group space.

Finally, the green shape has a nested heart shape *pasted into* it (the blue item). In terms of coordinate spaces this dependency is entirely encoded by the M matrix, which maps the blue space into the green space.

An interesting consequence of having those maps linked in a hierarchical way is that any object only needs to *know* how to translate its own coordinate system into the parent system, no matter what happens at a higher or deeper level.

Figure 11.
Affine maps (encoded as transformation matrices) match the hierarchical organization of page items. Each matrix is only responsible for mapping an inner space to the related parent space. We may compute matrix products to express the relationship between two spaces of any level. For instance the blue inner space is connected to the device space through the product
M × M × M.



$$(x',y') = (x,y) \times \text{M}$$

$$(x'',y'') = (x',y') \times \text{M} = (x,y) \times \text{M} \times \text{M}$$

$$(X,Y) = (x'',y'') \times \text{M} = (x',y') \times \text{M} \times \text{M} = (x,y) \times \text{M} \times \text{M} \times \text{M}$$

5. In fact, the actual parent of the group is a spread, which itself is mapped to the pasteboard, which itself is mapped to the device space at some point. We will discuss later these specific coordinate systems.

SUMMARY

A 2D coordinate is a pair of numbers (x, y) that locate a point relative to a given system of axes, origin, and units—referred to as a "coordinate system."

In InDesign every layout component (including pages and spreads) is bound to its own coordinate system, also known as its "inner coordinate space."

The functional relationship between two coordinate systems is called an "affine map." Any affine map is determined by a set of six real numbers, conventionally arranged in a matrix—a "transformation matrix."

The way we operate on such matrices might be purely described in terms of geometrical transformations. They address *"any linear mapping of two-dimensional coordinates, including translation, scaling, rotation, reflection, and skewing."* (InDesign SDK)[1]

InDesign internally reduces any transformation matrix to a combination of four atomic transformations: SCALING, SHEAR, ROTATION, and TRANSLATION, in that order. This "canonical decomposition" is unique and allows to treat separately the underlying parameters (scaling factors, shear angle, etc.).

1. Regarding transformation matrices, InDesign complies with the rules of the PDF Specification: *"A transformation matrix specifies the relationship between two coordinate spaces. By modifying a transformation matrix, objects can be scaled, rotated, translated, or transformed in other ways."* (PDF 32000-1:2008, p. 117.)

When one "applies" a transformation onto an object—say a rotation—InDesign does not really modify the geometry of the underlying shape. Instead, the application updates the affine map that links the coordinate space of that object to the coordinate space of its parent. Therefore, what is said a transformed object is nothing but the *same* object seen from a different perspective and/or location.

However, under specific circumstances InDesign may allow to use transformation tools in a way that actually impacts the inner geometry of the target object, rather than its affine map. This case will be referred to as a "deformation."[2]

Affine maps are chained according to the layout hierarchy. Thanks to this mechanism transformations that occur at any level may be described in the perspective of any other coordinate space.

EXERCISES

001. Let *`Obj`* be a **`PageItem`** and `(1,2,-1,0,3,1)` the matrix values of its affine map. Express in *`Obj`*'s parent space the coordinate pair of its inner space origin.

002. Explain why a shear angle cannot amount to 90°.

003. Let *`G`* be a **`Group`** having two rectangles *`R1`* and *`R2`* as direct children. (None of those page items has been scaled, skewed, or rotated yet.) Assume that the user then select *`G`* and apply a `45°` rotation to it. How do the transformation matrices of *`G`*, *`R1`*, and *`R2`* now look like?

004. Suppose that the area of some shape, measured in its inner space, is `8pt²`. Let `(2,3,3,6,-7,5)` be the matrix values of its affine map. What is the shape area measured in the parent space?

005. Let `S = (`s_x`,0,0,`s_y`,0,0)` be a valid SCALING matrix. What is the *inverse* of `S`? [The inverse of a matrix `M` is a matrix `M'` such that `M × M' = M'× M =` IDENTITY.]

006. Can an InDesign user change the location of an object relative to its parent **`Spread`** without changing at all its affine map?

2. During a deformation, the transformation parameters have only a temporary existence.

Object locations and transformations cannot be understood without a clear comprehension of InDesign-specific coordinate spaces. This section presents those fundamental frames to programmers before they fiddle with geometry.

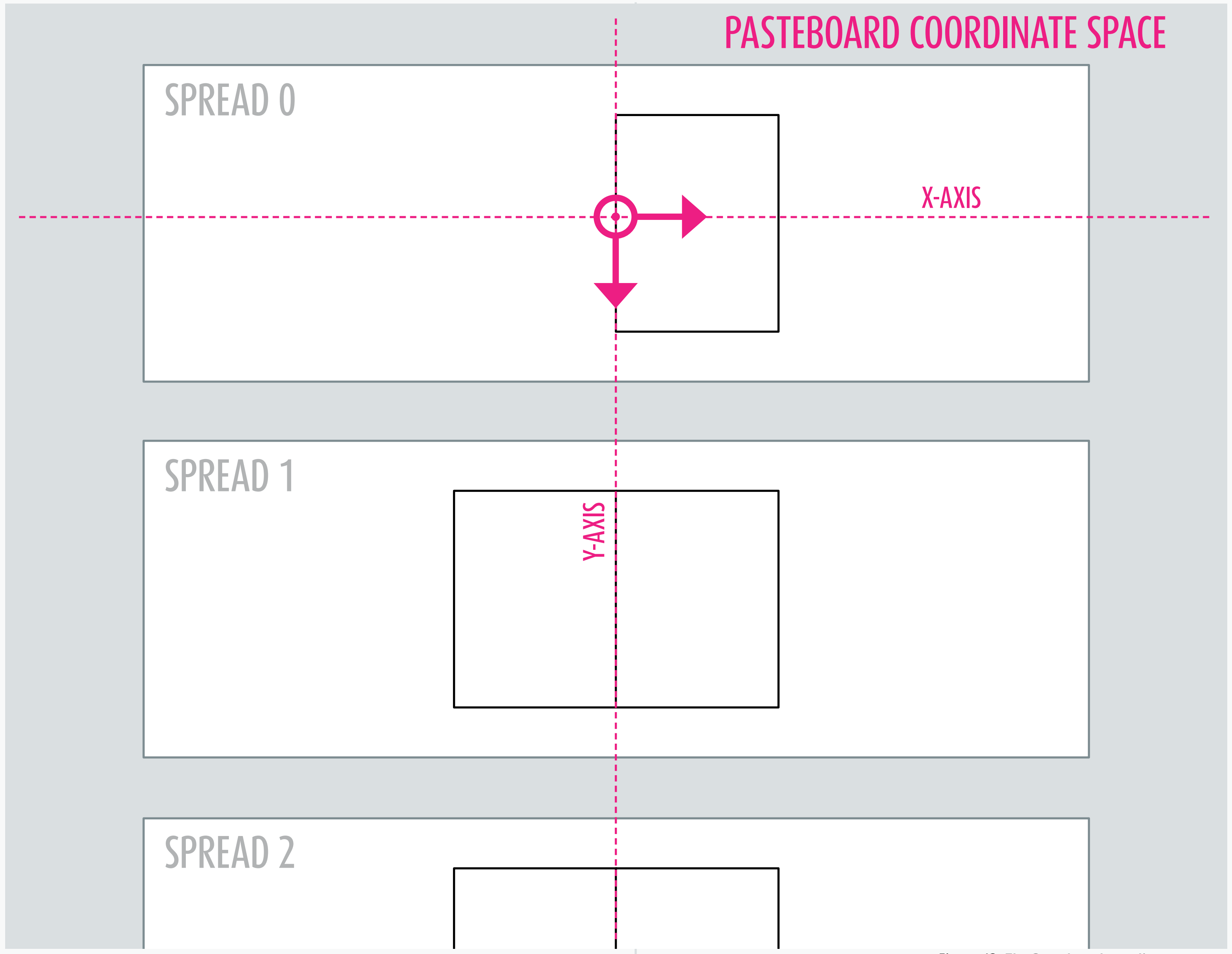


Figure 12. The Pasteboard coordinate space.

Pasteboard Coordinate Space

At the very top level is the PASTEBOARD COORDINATE SPACE, a kind of Galilean reference frame. It is the global, absolute coordinate system that surrounds the whole document.

The pasteboard[1] space encompasses the entire workspace, including interstitial areas where no object can be laid out at all. This root entity might be seen as the virtual parent of every document spread. We will consider it a *device space* for the on-screen layout. InDesign uses in fact higher coordinate systems that reflect how the layout is shown in different *views* (based on windows, scrolling, magnification…) but these do not regard the imageable document.

Any location can be easily and univocally expressed in the pasteboard coordinate space, whose origin is the center of the first spread of the document. The *x*-axis is horizontal with values increasing from left to right; the *y*-axis is vertical with values increasing from up to bottom (see Figure 12).

1. Note that the pasteboard is not represented as an object in the InDesign Scripting DOM, which may cause some confusion. The word 'pasteboard' is commonly associated with the outer region of a page (white background) which still contains or may contain layout items. In fact, this extra region would be rather described in terms of spread margins, because anything that lives there is in the scope of the corresponding spread and actually belongs to it. Due to this confusion some settings that regard spreads are referred to as *pasteboard* things in the DOM. For example, the `PasteboardPreference` object (available under `Document` and `Application`) exposes a `pasteboardMargins` property (array of two measurement units) which controls the width and the height of the spread margins. In InDesign CS4 only the height of the spread margin was addressable, *via* the `minimumSpaceAboveAndBelow` property.

The length along each axis is measured in points and there is no way to change this. InDesign documents and all basic coordinate spaces handle measurements in PostScript points, although both Scripting DOM and ExtendScript's core provide tools to convert measurements into other units, as this is done in the application GUI.

Since the pasteboard is not an actual DOM object, it has no parent and therefore no transformation matrix bound to it—at least, nothing that we could reach through scripting.[2]

Given a document, you can refer to the pasteboard coordinate space using the enumerated value `CoordinateSpaces.pasteboardCoordinates` in any method that handles coordinates. We will see later various uses for this key.

Spread Coordinate Space

"Each spread has its own coordinate space, also known as the inner coordinate space for a spread. The origin of the spread coordinate space is the center of the spread. The parent coordinate space is pasteboard coordinate space." (InDesign SDK, see **Figure 13**.)

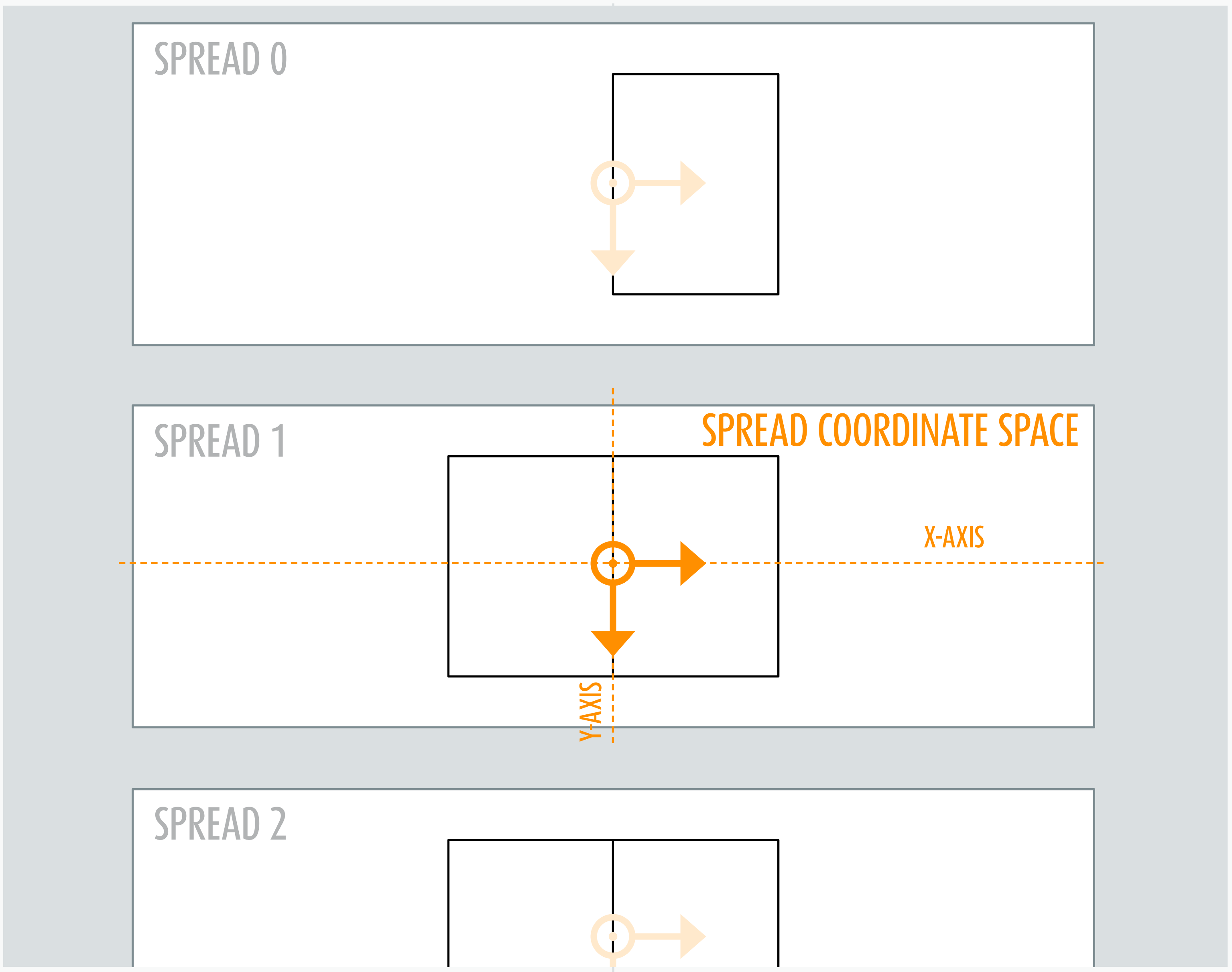
A `Spread` object being known, you can directly refer to its specific coordinate space using `CoordinateSpaces.spreadCoordinates` in every method that handles coordinates. Be aware that the origin of a spread coordinate space does not coincide with the default zero point in Ruler Per Spread mode. Also, unlike the rulers coordinate system (which we will study later), spread coordinate spaces only address measurements in points.

As a general rule, the transformation matrix of a spread—read: the affine map that connects this spread space to the pasteboard space—will specify a TRANSLATION in the form:

$$\begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & t_y & 1 \end{bmatrix}$$

where t_y represents the offset along the y-axis relative to the pasteboard space origin. Indeed, although

Figure 13. A typical spread coordinate space.



2. It is worth noting, however, that the pasteboard space origin directly depends on the size and state of the *first* document spread, meaning it may *move* in various circumstances, such as changing the page size or transforming a spread. Also, facing-pages vs. non-facing-pages documents follow distinct rules on positioning spreads.

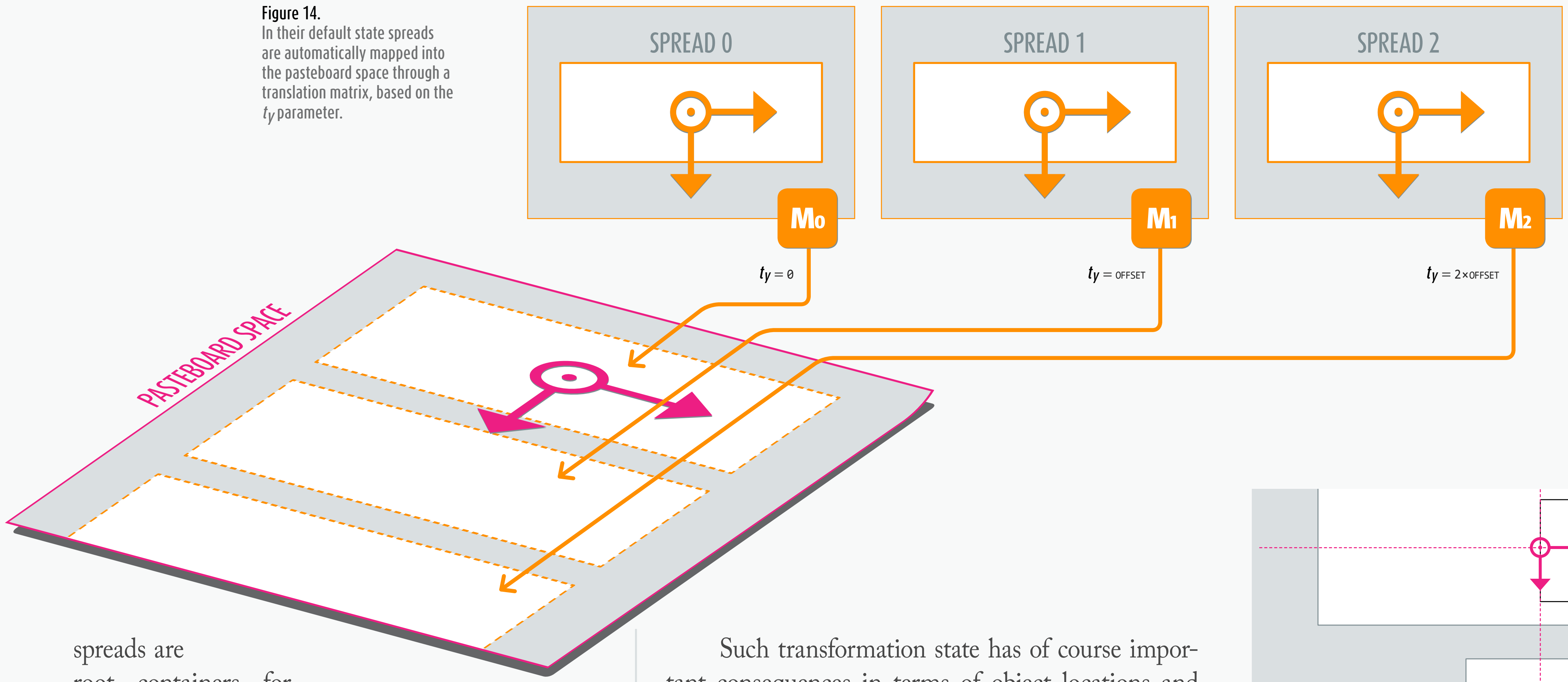


Figure 14.
In their default state spreads are automatically mapped into the pasteboard space through a translation matrix, based on the t_y parameter.

spreads are root containers for pages and page items, they can usually be treated from the pasteboard perspective as simple rectangular regions ordered along the vertical axis (see **Figure 14**).

But in InDesign CS4 and later spreads support transformations, with a few restrictions though. That is, the affine map of a spread may contain non-default SCALING, SHEARING and/or ROTATION parameters.[3] As a basic example let's apply a 90° Rotated Spread View on a spread. The pasteboard perspective will then look like in **Figure 15** below.

3. The only component that you cannot control is the TRANSLATION part. The (t_x, t_y) parameters of a spread matrix entirely depend on automatic positioning, which in turn depends on respective spread areas and orientation, facing-pages option, etc.

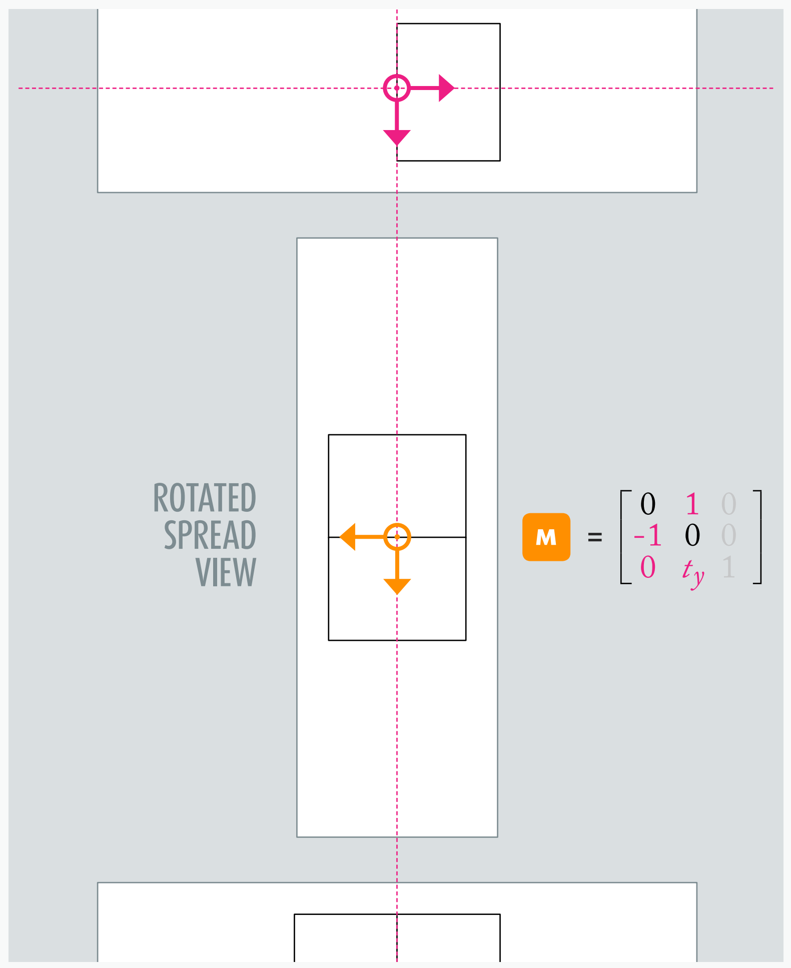
Such transformation state has of course important consequences in terms of object locations and metrics.

How to get details on rotated views from your scripts? You need first to retrieve the affine map of the spread, as follows:

```
// 01. Get the affine map of a spread
const CS = CoordinateSpaces,
      CS_PARENT = CS.parentCoordinates;
var mx = mySpread.transformValuesOf(CS_PARENT)[0];
alert(mx.matrixValues);
```

The method `transformValuesOf(`*anySpace*`)` returns a singleton array whose unique element is a matrix that maps the caller inner space to *anySpace*. Therefore, the following code:

Figure 15.
When a "Rotated Spread View" is applied (from the Pages panel), the transformation matrix of the underlying spread space is accordingly changed to reflect the rotation.


ROTATED SPREAD VIEW
$M = \begin{bmatrix} 0 & 1 & 0 \\ -1 & 0 & 0 \\ 0 & t_y & 1 \end{bmatrix}$

```
anyObj.transformValuesOf(CoordinateSpaces.
parentCoordinates)[0]
```

always returns the affine map attached to *anyObj*, as the enum value **CoordinateSpaces.parentCoordinates** points out to the parent coordinate space associated to *anyObj* along the hierarchy.[4]

We store the result, a **TransformationMatrix**, in the variable **mx** so that we can study the underlying components. The property **mx.matrixValues** reveals the matrix parameters in the form [a, b, c, d, tx, ty] (array of six numbers, keeping the notations used in the previous chapter.)

From then it's easy to get the *meaning* of these data:

Matrix Values	Spread Rotation State
[1, 0, 0, 1, 0, *ty*]	Default. (No rotation applied.)
[0, 1,-1, 0, 0, *ty*]	90° clockwise (CW).
[0,-1, 1, 0, 0, *ty*]	90° counterclockwise (CCW).
[-1, 0, 0,-1, 0, *ty*]	180°.

Alternately the **TransformationMatrix** object exposes a property, **counterclockwiseRotationAngle**, which indicates the ccw rotation angle in degrees.

```
// 02. Display the rotation angle of a spread
const CS = CoordinateSpaces,
      CS_PARENT = CS.parentCoordinates;
var mx = mySpread.transformValuesOf(CS_PARENT)[0];
alert(mx.counterclockwiseRotationAngle);
```

4. As we are considering a **Spread** object, **CoordinateSpaces.parentCoordinates** is, in fact, equivalent to **CoordinateSpaces.pasteboardCoordinates**. The former syntax is just more generic.

Page Coordinate Space

"Each page has its own coordinate space, also known as the inner coordinate space for a page. The parent coordinate space for page coordinate space is spread coordinate space. The origin of page coordinate space is the top-left corner of the page." (InDesign SDK, see Figure 16.) A **Page** being known, you can refer to its specific space using **CoordinateSpaces.pageCoordinates** in CS6 and later.

Here again, note that the origin of a page coordinate space is in no way determined by the zero point in Ruler Per Page mode—users can move the zero point anywhere in the page area. Also, unlike rulers' coordinate system, a page coordinate space only handles measurements in points.

As a general rule, the transformation matrix of a page—read: the affine map that connects this page space to the parent spread space—will specify a TRANSLATION in the form:

$$\begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ t_x & T_y & 1 \end{bmatrix}$$

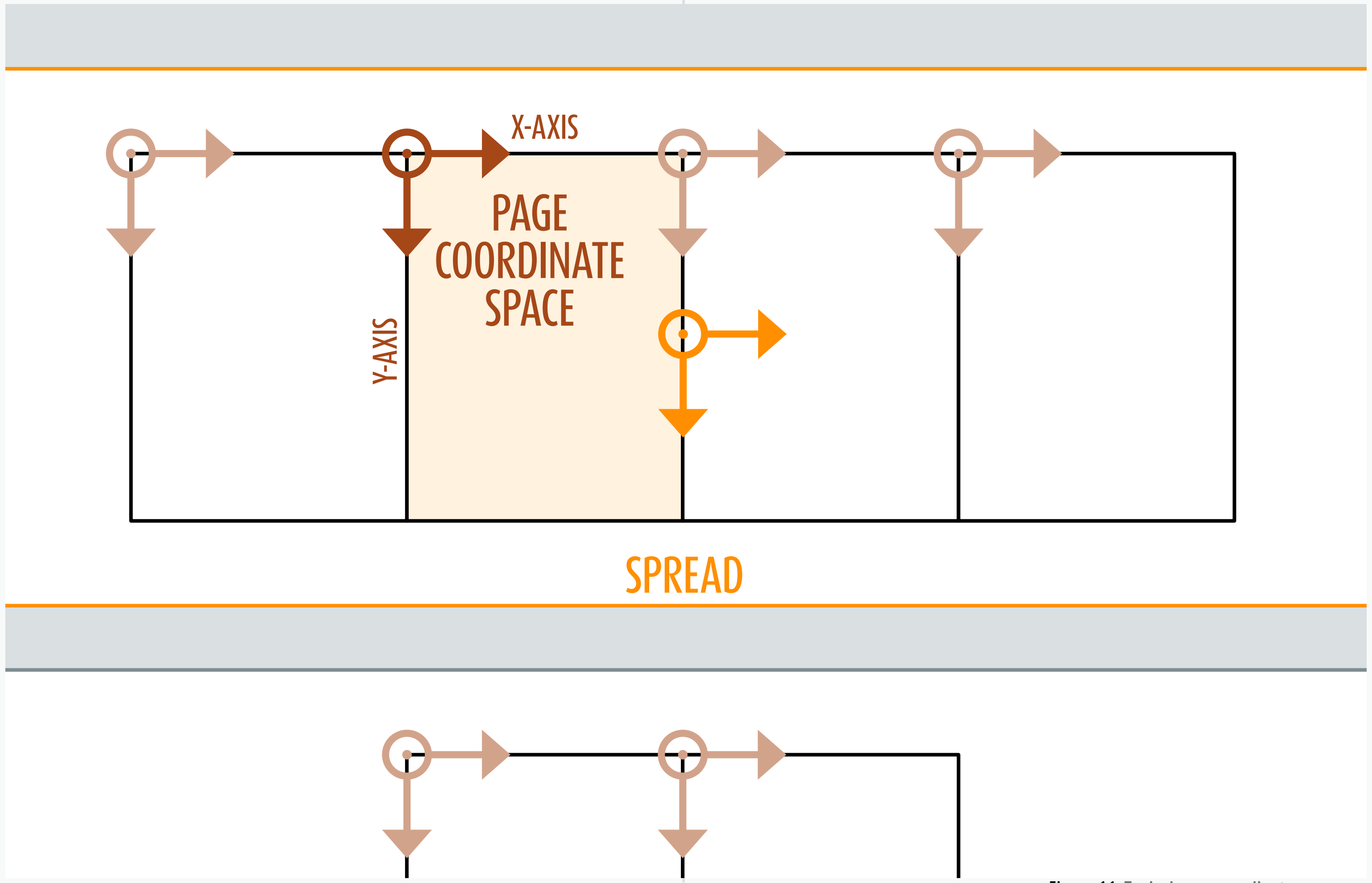


Figure 16. Typical page coordinate space.

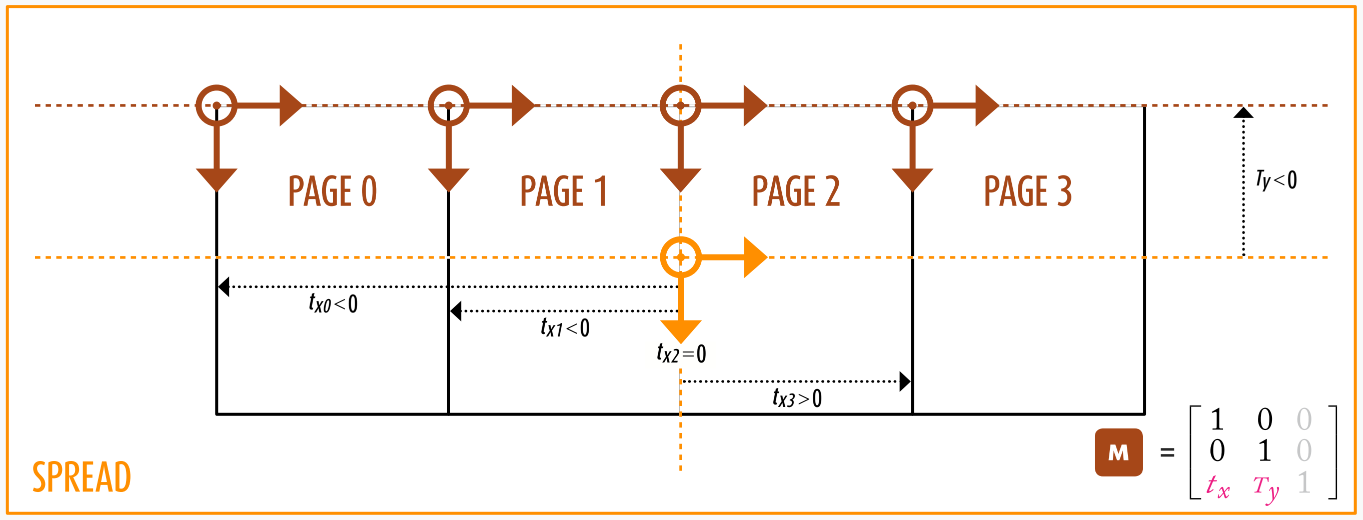


Figure 17. A typical four-page spread in its default state. As long as pages remain untransformed, their affine map boils down to a simple (tx,Ty)-translation. Note that rotating the spread view (that is, changing the affine map of the spread itself) wouldn't have any effect on those page-to-spread matrices (M).

where t_x represents an offset along the x-axis relative to the spread coordinate space, while T_y stands for some constant y-offset, as illustrated in Figure 17.

Let's pause for a moment and try to clarify why T_y is a negative offset. A primary reflex is to think that the TRANSLATION encoded in **M** should *move* the page origin *to* the spread origin, which would lead to $T_y>0$. That's a misrepresentation of what the TRANSLATION is. As said earlier the purpose of the map **M** is to convert page-relative coordinates into spread-relative coordinates. In particular, applying the map to `(0,0)`—i. e., the origin of the page in its own coordinate space—must result in a coordinate pair (x_o, y_o) which *positions* that origin in the spread coordinate space. Considering PAGE 2 in the figure we clearly expect $x_o = 0$ and $y_o < 0$. Let's apply the affine map:

$$[\,x_o, y_o, 1\,] = [\,0, 0, 1\,] \times \mathbf{M} = [\,t_x, T_y, 1\,].$$

It comes $t_x = x_o (= 0)$, and $T_y = y_o (< 0)$, so we can express the rule as follows: *The affine map of a page is usually a simple* TRANSLATION *whose* (t_x, t_y) *parameters reflect the location of the page space origin relative to the parent spread origin (i.e. the center point of the spread).*

Let's reveal these translation parameters using `Page.transformValuesOf()`:

```
// 03.  Display the translation values of
//      all pages hosted by spreads[spdIndex]
const CS = CoordinateSpaces,
      CS_PARENT = CS.parentCoordinates;
var spd = app.activeDocument.spreads[spdIndex],
    pgs = spd.pages.everyItem(),
    a = [].concat(pgs.transformValuesOf(CS_PARENT)
        )[0],
    i = a.length;
while( i-- ) (a[i]=a[i].matrixValues).splice(0,4);
alert( a.join('\r') );

//  Typical result for a four-page spread
//  in facing-pages mode
//  ---
//      -1200,-425
//      -600,-425
//      0,-425
//      600,-425
```

Page Size and Location Issues

Prior to InDesign CS5 pages couldn't be *transformed* at all (that is, page affine maps couldn't be programmatically changed). Now `Page` objects support the `transform()` method, meaning that we can alter the underlying matrix so that a specific page appears transformed within its parent spread.

Page transformation is Pandora's box. It leads to both great possibilities and unexpected troubles regarding page coordinate spaces. First above all, you cannot assume that page sizes are uniform anymore. Document settings only specifies a *default page size*. Using the Page Tool, the user can change the dimensions of a specific page and/or its default location. Such effects depend on document facing-pages options, shuffling behavior between spreads, layout rules involving master page inheritance mechanism, and so on.[5]

5. In InDesign CS5 and later, you cannot even be sure that the origin of a page coordinate space will match the top-left corner of the corresponding page! It's easy to break rules playing with the Page Tool or applying custom transformations to the master-to-page matrix (`Page.masterPageTransform`). See next page for an example.

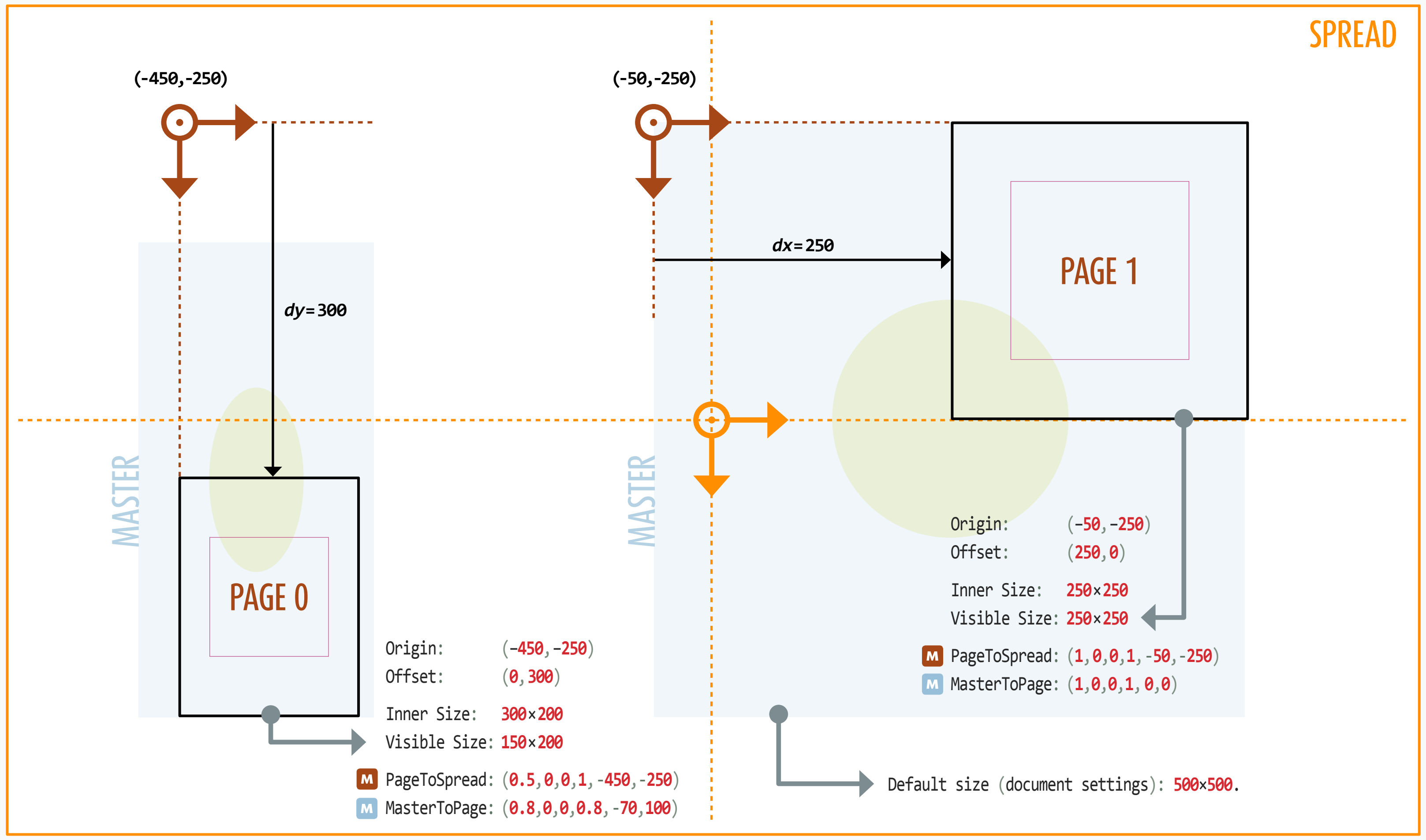


Figure 18.

Sample spread demoing various issues regarding page location and size.

The actual dimensions of PAGE 0 is 300×200 (inner size) but since its affine map *PageToSpread* specifies a 50% scaling along the *x*-axis, it appears reduced to 150×200 in the parent spread. In addition, a custom transformation *MasterToPage* (80% scaling + translation) is applied to its master page relative to the page space* before it undergoes the page-to-spread mapping. The result becomes pretty difficult to predict!

By contrast PAGE 1 has no scaling applied (relative to the spread) and its masterPageTransform matrix is transparent (identity). However it still has a custom size (250×250) relative to the document settings (500×500).

In both cases we can observe that the origin of each page coordinate space—as revealed by the translation values of the respective affine maps—does not coincide with page's top-left corner. A vertical offset (*dy*=300) appears for PAGE 0 and an horizontal offset (*dx*=250) appears for PAGE 1. So, you cannot blindly trust the "page coordinate space" system as defined in Adobe's documentation.

* From *IDML File Format Specification* (version 8.0, page 157): "Because the master page applied to each page can be of a different size than the page, InDesign provides a way to position the contents of the master page as they appear on the page. In IDML, this transformation appears as the MasterPageTransform attribute on the `<Page>` element. While this is a complete transformation matrix, **only translations are supported**." The last statement is wrong! `Page.masterPageTransform` is available in InDesign's DOM from CS5 and it behaves as a fully customizable matrix.

When `DocumentPreference.facingPages` is turned off, in particular, pages within a spread can be freely *repositioned* along both the *x*- and the *y*-axis. Depending on how this is done the user may shift the top-left corner of the page relative to the actual origin of its inner space (see Figure 18).

Another issue should be mentioned. The global preference `app.transformPreferences.whenScaling` has a critical impact on how scaling is performed on graphic components, including pages:

➔ `WhenScalingOptions.applyToContent` prevents any scaling operation from being registered as a transformation. In other words, scaling is treated as a *deformation*, meaning that the inner geometry of the target object is actually *resized*. In this context, applying some scaling transformation to a page does not update the scaling values of its affine map. Instead, the actual (inner) size of the page will change.[6]

➔ On the contrary, the option `WhenScalingOptions.adjustScalingPercentage` forces InDesign to manage scaling through transformation matrices, so that the inner geometry of the target is not resized. In other words, scaling a page will not change its actual inner size. What you see from the spread perspective ("visible size") is not what you get on printing or exporting that page—unless you output the spread itself.

For all these reasons, it is worth considering pages as just rectangular items—which they actually are, under the hood—and to compute coordinates for pages as well as for page items, that is, relative to the parent spread space or the pasteboard space (depending on your needs).

As for determining the real size of a page considered as a *device space*, best is to use the bounding box coordinate system, as we shall see.

6. By contrast, transforming a spread never results in a *deformation*. The printable size of a spread is determined by internal rules, disregarding whether that spread is transformed (e.g. *scaled*) and how it is rendered in the perspective of the pasteboard coordinate space.

Finally, remember that the actual parent of any top-level item is a **Spread**.[7] Thus, there is no rigid connection between a page and the objects which happen to stand on it.

Inner Coordinate Space of a Page Item

Adobe's documentation does not tell much about basic **PageItem**'s inner coordinate space: *"Each page item has its own coordinate space, known as its inner coordinate space. Each page item has an associated parent coordinate space (…)"* (InDesign SDK.)

Although we already know that any coordinate space handles measurements in points and that the associated transformation matrix describes the affine map that connects the inner coordinate space to the parent space, a question remains unanswered:

Where exactly is located the origin of a PageItem inner space relative to its own geometry? Can we extrapolate what we know about spread (or page) coordinate spaces?

One might intuitively assume that, given a **PageItem**, its inner space coincide with either the top-left corner, or maybe the center point, of some intrinsic bounding box. Unfortunately this is definitely not the case!

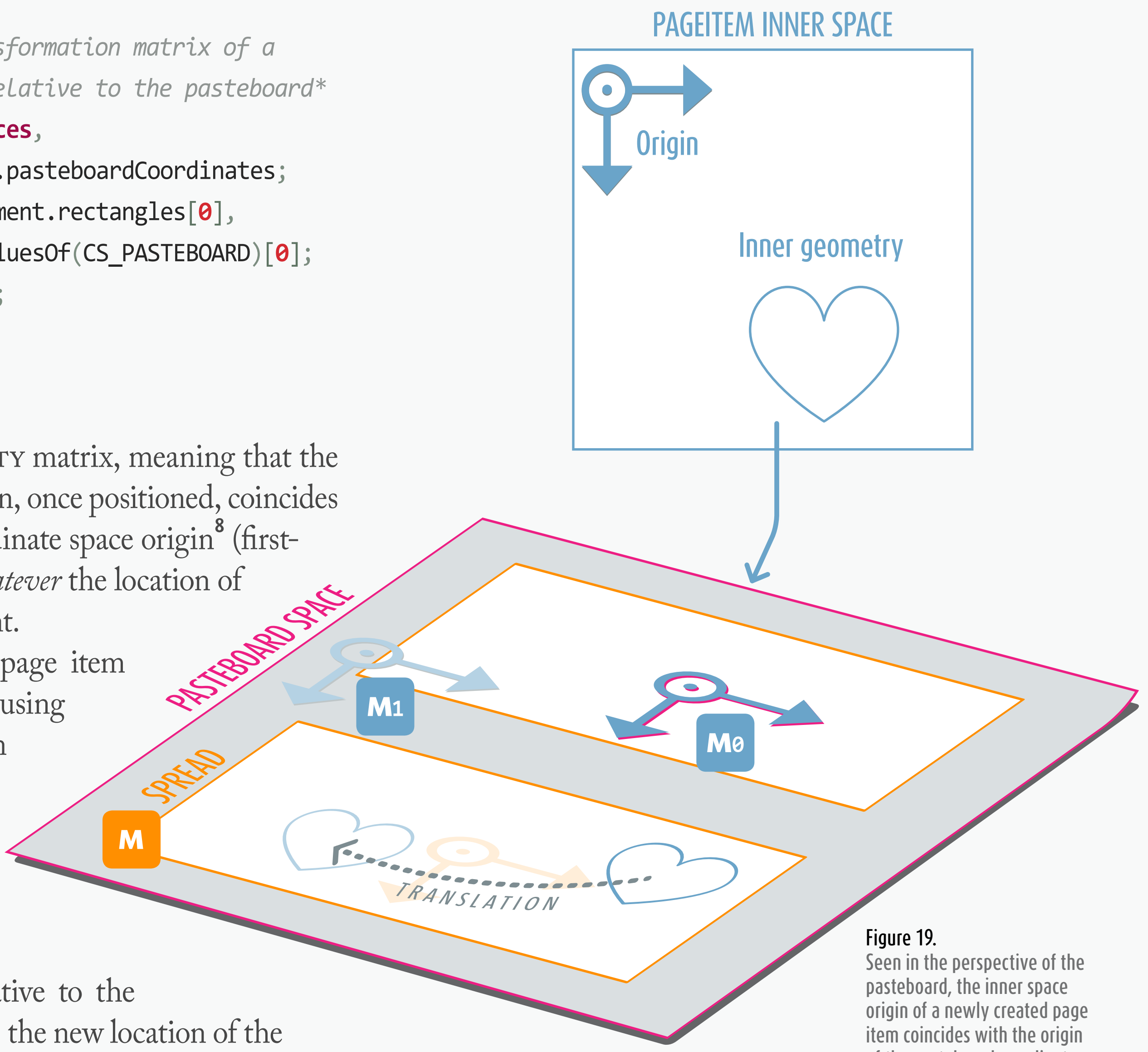
Let's set up a new InDesign document having several empty pages, then draw a basic rectangle on the last page, using the Rectangle tool. Do not move or transform the object, just run the following script:

```
// 04.  Display the transformation matrix of a
//      new rectangle *relative to the pasteboard*
const CS = CoordinateSpaces,
      CS_PASTEBOARD = CS.pasteboardCoordinates;
var rec = app.activeDocument.rectangles[0],
    mx = rec.transformValuesOf(CS_PASTEBOARD)[0];
alert( mx.matrixValues );
//  Result:
//  1,0,0,1,0,0
```

The result is the IDENTITY matrix, meaning that the rectangle inner space origin, once positioned, coincides with the pasteboard coordinate space origin[8] (first-spread's center point), *whatever* the location of the object in the document.

Now if you move the page item from its original location using the Selection tool and then re-run the script, you get of course another result in the form (1,0,0,1, t_x, t_y), where (t_x, t_y) are the TRANSLATION values relative to the pasteboard space—that is, the new location of the inner space origin within the pasteboard space.

Figure 19 illustrates the case of moving a shape in terms of matrix mapping. Let M_0 be the affine map of the page item in its original state, M the affine map of its parent spread. The product $M_0 \times M$ (that is, *ItemToSpread* × *SpreadToPasteboard*) results in the *ItemToPasteboard*



$M_0 \times M$ = (1,0,0,1,0,0)

$M_1 \times M$ = (1,0,0,1,t_x,t_y)

Figure 19. Seen in the perspective of the pasteboard, the inner space origin of a newly created page item coincides with the origin of the pasteboard coordinate space, whatever the parent spread and the location of the object. Then, when some transformation is applied (e.g. a TRANSLATION) the affine map is updated accordingly. Anyway, note that the "inner geometry" does not change at all. Page item's path points keep the same location relative to the inner coordinate space.

7. This statement is partially wrong in CS4, where a **PageItem** could still *belong* to a **Page**. However, even in CS4 the *parent coordinate space* of a page-child item is the related spread coordinate space.

8. Indeed the (0,0) location in the inner space is *mapped* onto the (0,0) location in the pasteboard space.

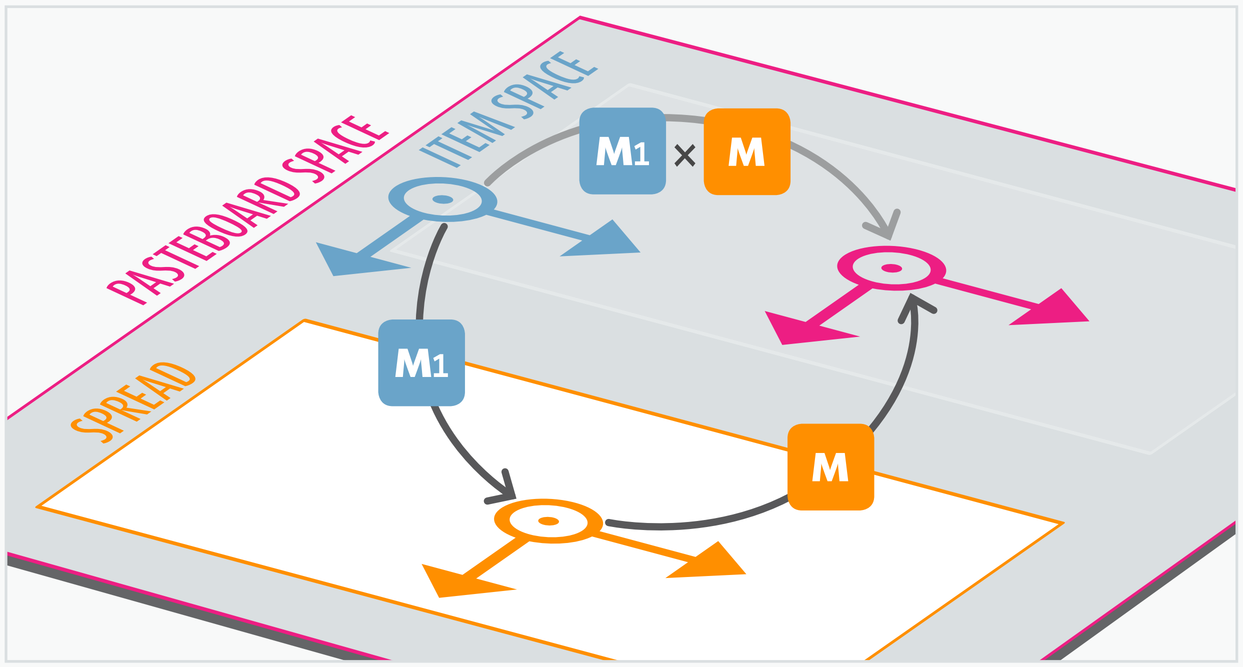


Figure 20.
Chasles' Relation between the *ItemToSpread* matrix $\boxed{M_1}$, the *SpreadToPasteboard* matrix $\boxed{M}$, and the resulting *ItemToPasteboard* matrix $\boxed{M_1} \times \boxed{M}$.
It is assumed here that the page item is a direct child of the spread. (See **figure 11** for a detailed visualization of matrix products.)

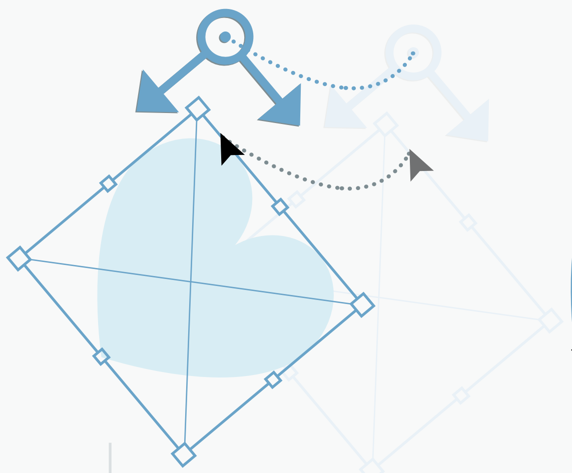


Figure 21a.
Moving a page item, the regular way (TRANSLATION).

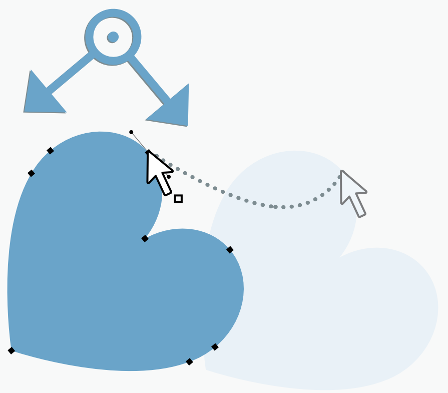


Figure 21b.
Displacing a path, relative to the inner origin.

matrix, which as we have just observed is the IDENTITY mapping. In short, $\boxed{M_0} \times \boxed{M} =$ IDENTITY. You can check by yourself that this equality remains true regardless of the transformation state of the spread at the time you create the item. From this we can derive an interesting property: *The affine map of a newly created page item is nothing but the inverse matrix of its parent spread affine map.*

As for $\boxed{M_1} \times \boxed{M}$, this product just reflects the transformation which the page item space globally undergoes *relative to the pasteboard space*, i.e. the TRANSLATION $(1, 0, 0, 1, t_x, t_y)$ in our example. More generally we have something of a Chasles' Relation between transformation matrices. Let

M_{AB} be the matrix that maps space A to space B,
M_{BC} be the matrix that maps space B to space C,
M_{AC} be the matrix that maps space A to space C,

then we have $M_{AB} \times M_{BC} = M_{AC}$.

Figure 20 helps you visualize this rule.

Moving vs. Displacing

When we *move* a page item using the Selection tool, or even when we cut-and-paste this object onto another page, the location of its path points do not change within the inner coordinate space. That the reason why we claim that "moving an object" actually means applying a TRANSLATION to its affine map—or, to put it equivalently, altering the TRANSLATION values of its affine map.

Therefore, most of the time *a move is a transformation*. However, as already observed, some InDesign tools allow the user to perform a *deformation* instead of a *transformation*.[9] In such cases we will rather talk about a DISPLACEMENT to avoid any confusion with regular moves processed through TRANSLATION. See **Figure 21a** vs. **21b**.

9. See Chapter 1, Key Concepts: "Transformation vs. Deformation."

	TRANSFORMATION	DEFORMATION
NAME	Translation (*regular move*)	Displacement (*fake move*)
HOW-TO	Usual means, e.g. select and drag the bounding box of the page item.	"Direct-select" path points and drag them all to another location.
INNER GEOMETRY	Unchanged.	Changed (relative to the origin).
LOCATION OF THE ORIGIN	Changed (relative to the parent space).	Unchanged.

2. InDesign Coordinate Spaces

SUMMARY

InDesign defines several coordinate spaces relative to which object coordinates are interpreted and transformations are processed. All "InDesign coordinate spaces" rely on a 2D ORTHOGONAL basis listed in CLOCKWISE order and using POSTSCRIPT POINT as canonical unit.[1]

➜ The PASTEBOARD COORDINATE SPACE surrounds the whole document. Its origin is the center point of the first spread. It has no affine map (as it represents the top of the hierarchy). In the Scripting DOM it is referred to as `CoordinateSpaces.pasteboardCoordinates`.

➜ A SPREAD COORDINATE SPACE reflects the inner space for a specific `Spread` object. Its affine map, usually a TRANSLATION, links it to the pasteboard space. Its origin is the center point of the underlying spread. In the Scripting DOM it is referred to as `CoordinateSpaces.spreadCoordinates`.

➜ A PAGE COORDINATE SPACE reflects the inner space for a specific `Page` object. Its affine map links it to the parent spread space. Its origin is, *in most cases*, the top-left corner of the page. In the Scripting DOM it is referred to as `CoordinateSpaces.pageCoordinates` in InDesign CS6 and later.[2]

1. Bounding box and rulers coordinate systems are not pure coordinate spaces. Those systems will be discussed in the next chapters.

2. Prior to CS6 a page coordinate space has no dedicated enum value but it already makes sense, in the scope of a `Page` object, as `CoordinateSpaces.innerCoordinates`.

➜ Any `PageItem` also has an associated inner coordinate space, whose affine map is connected to the parent coordinate space—which can be either a `Spread` space, or another `PageItem` space depending on the document hierarchy.[3] `PageItem`'s inner space ORIGIN coincides with the pasteboard origin when the item is created and positioned, then subsequent moves are usually processed by adjusting TRANSLATION values.

➜ In addition, the Scripting DOM provides a special keyword, `CoordinateSpaces.parentCoordinates`, that refers to the parent coordinate space of any *transformable* object. The generic code

```
myObj.transformValuesOf(CoordinateSpaces.
parentCoordinates)[0]
```

returns a `TransformationMatrix` which describes the affine map applied to `myObj`'s coordinate space.

Depending on InDesign versions, `Spread` and/or `Page` coordinate spaces can be transformed in unexpected ways—including ROTATION, SCALING, and SHEAR. Tranforming pages may lead to obscure artifacts: offset of the default origin location, difference between "inner size" and "visible size," etc.

CHASLES' RELATION applies to transformation matrix product. Given three coordinate spaces A, B, and C, if the matrix M_{AB} maps A to B, and the matrix M_{BC} maps B to C, then $M_{AB} \times M_{BC}$ is the matrix that maps A to C.

3. Fortunately (?) the parent coordinate space of a `PageItem` space cannot be a page coordinate space, despite the fact that *in CS4* `PageItem.parent` may refer to a `Page`. (See also page 17, note 7.)

EXERCISES

001. The Scripting DOM also exposes the enum value `CoordinateSpaces.innerCoordinates` (which we have not introduced yet.) Try to predict its usage and the result of

```
anyObj.transformValuesOf(CoordinateSpaces.
innerCoordinates)[0].matrixValues
```

where `anyObj` can refer as well to a `Spread`, a `Page`, or any `PageItem` in any transformation state.

002. Let's set up a new `Document` having a single `Page` (non-facing mode). In case *(a)* we draw an `Oval` at the center of the page, then we rotate the spread view `90°` clockwise. In case *(b)* we rotate the spread view `90°` clockwise, then we draw an `Oval` at the center of the page. Are there some differences between *(a)* and *(b)* in terms of transformation matrices? Provide a script that corroborates your answer.

003. Using the Selection tool, select the left edge of a `SplineItem`'s bounding box and drag it to the left so that the width of the underlying shape is increased. Does this change the associated affine map? Provide a script that corroborates your answer.

004. A document has a unique `Page` whose affine map is `(0.5,-0.25,0.25,0.5,-125,-125)`. The containing `Spread` has a `180°` "rotated view" applied. What is the transformation matrix of the page coordinate space relative to the pasteboard coordinate space?

A bounding box is *the smallest rectangle that encloses a geometric page item.** This definition intuitively corresponds to what we perceive as a selection frame in the interface. When the user is in drawing a vector shape and switches to the Selection tool s/he immediately sees a rectangle bordering the entire path. This sounds dead easy at first sight, but now it's time to open that black box.

A Bounding Box Depends on a Coordinate Space

You might think that a given page item has a single, uniquely determined BOUNDING BOX. But does it really make sense to talk about *the* smallest rectangle that encloses a shape? **Figure 22** shows that we can construct as many enclosing rectangles as desired, depending on a chosen orientation.

Here is a cleaner definition of a bounding box in InDesign's workspace: *Given a* COORDINATE SPACE *and given a geometric object, the associated* BOUNDING BOX *is the smallest rectangle that encloses the shape with respect to the specific axes and orientation of that coordinate space.*

Each coordinate space governs how to frame an object. Unlike InDesign's GUI, which always displays bounding boxes relative to the inner space,[1] a script can select any other reference frame (parent space, spread space, pasteboard space).

From then we can discern at least four distinct bounding boxes for the same page item:

➔ The INNER SPACE RELATIVE BOUNDING BOX (which we will abbreviate to "inner box" from now on). It is the enclosing rectangle aligned with the axes of the *inner* coordinate space.

➔ The PARENT SPACE RELATIVE BOUNDING BOX (in short, the "in-parent box"). It is the enclosing rectangle aligned with the axes of the *parent* coordinate space.

➔ The SPREAD RELATIVE BOUNDING BOX (in short, the "in-spread box"). It is the enclosing rectangle aligned with the axes of the *spread* coordinate space. Of course the in-spread box *is* the in-parent box if the page item has no intermediate owner along the hierarchy. Also, if the object we consider is the spread itself, its in-spread box is nothing but its inner box.

➔ The PASTEBOARD RELATIVE BOUNDING BOX (in short, the "in-board box"). It is the enclosing rectangle aligned with the axes of the *pasteboard* coordinate space, that is, the horizontal and vertical axes of your screen. As long as the containing spread is untransformed (no rotated view applied, etc.), the in-board box of a page item coincides with its in-spread box.

* *InDesign SDK*, "Bounding box and IGeometry."

1. Except when multiple items are selected; in such case InDesign shows the bounding box aligned with the pasteboard space.

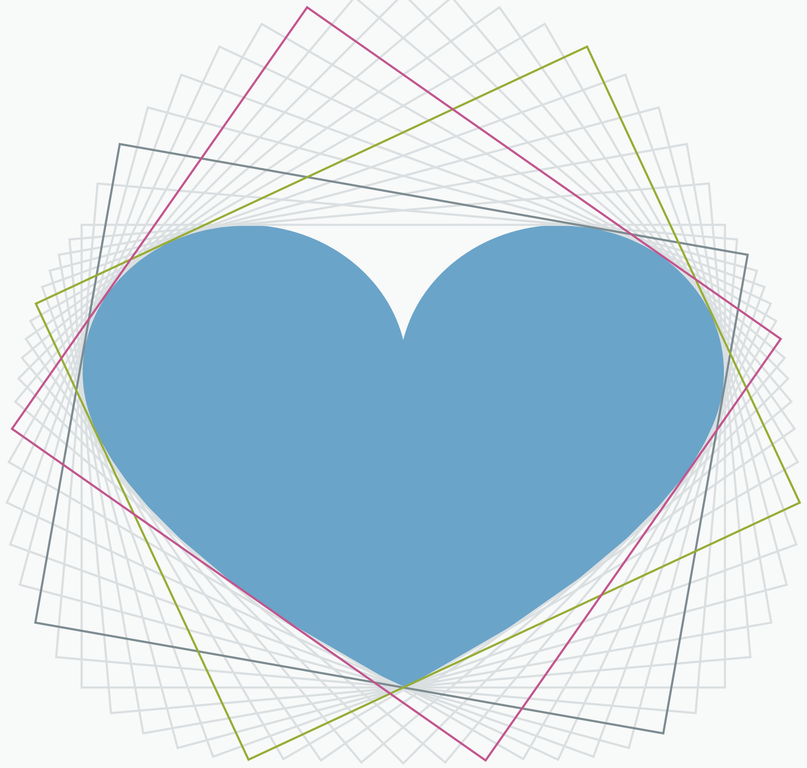


Figure 22.
In the absence of further instructions there are infinite ways of enclosing a shape in a rectangular region. In InDesign geometry, several bounding boxes can be defined for the same object depending on the coordinate space we consider.

In CS6 and later you could also consider the "in-page box" of an object, that is, the bounding box aligned with the associated page coordinate space, provided that the object actually meets a page. In practice it is discouraged to rely on in-page boxes because of the many inconsistencies already mentioned about page coordinate spaces.[2]

A Bounding Box Defines a Coordinate System

A bounding box must be understood as an oriented rectangle in that it defines a system of anchor points in clockwise order, TOP-LEFT, TOP-RIGHT, BOTTOM-RIGHT, BOTTOM-LEFT—with respect to the orientation of the associated coordinate space.

This set of anchor points makes bounding boxes similar to coordinate spaces although they do not use the same origin, nor the same units. To avoid any confusion we shall talk about *bounding box coordinate system*, or "bounds coordinates."[3]

Bounds coordinates use the top-left anchor as the origin. The distance from the top-left anchor to the top-right anchor provides the *horizontal unit*, while the distance from the top-left anchor to the bottom-left anchor provides the *vertical unit*. It follows in particular that the center anchor point of the bounding box has the coordinates `(0.5,0.5)` whatever the actual dimensions of the enclosing rectangle (see Figure 23). Using this system—or dedicated enumerators—a script can easily specify any anchor point location. It can also access to other relative locations, such as `(0.25,0.75)`, `(-1,0)`, `(2,3)` etc. Note that bounds coordinates allow to define locations outside the bounding box.

While a given shape has different bounding boxes (one for each connected space) and then different bounds coordinate systems, an interesting property is that the location of the center anchor point remains invariant.

2. The best you can do is to handle the inner box of a given page (that is, its own bounding box relative to its inner space) then to resolve associated anchor points into spread or pasteboard coordinates. Considering page bounding box is helpful when you have to fix page size and/or location issues (see previous chapter, page 15).

3. Adobe's documentation may refer as well to "bounds space," keeping in mind that such coordinate system is not strictly a coordinate space. (Coordinate spaces are discussed in Chapter 2.)

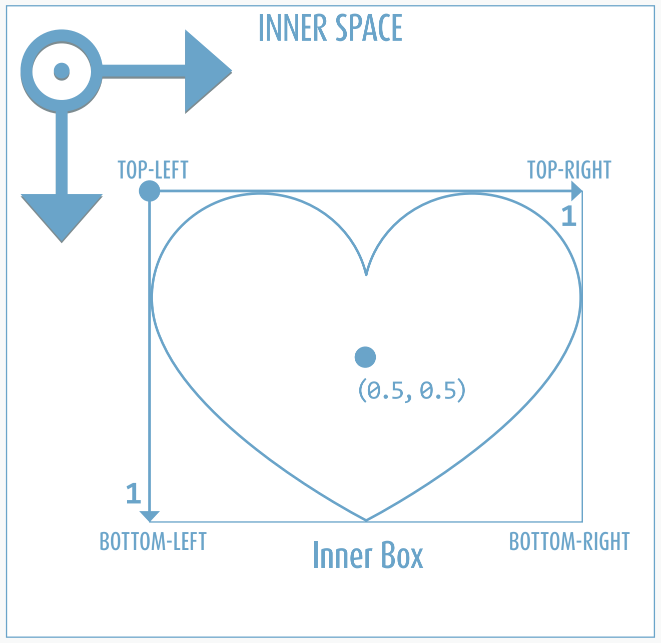
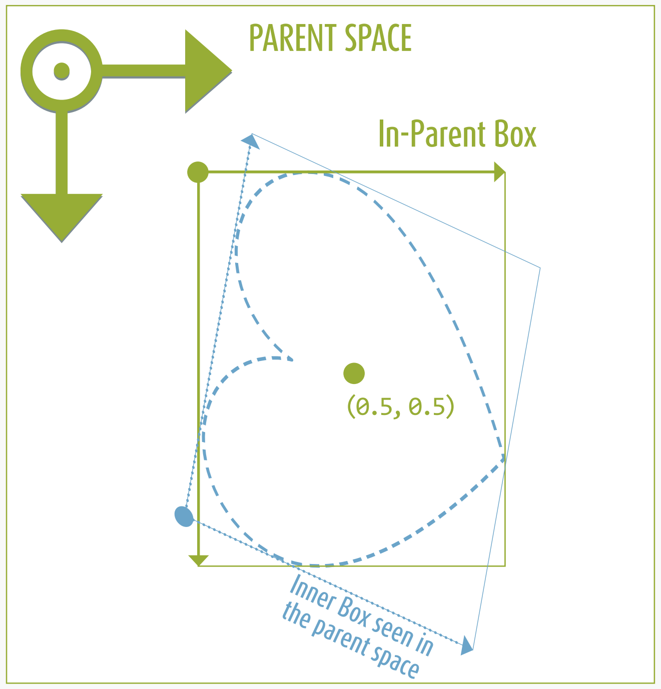


Figure 23.
Inner box (blue) and in-parent box (green) of a page item. Seen in its associated coordinate space a bounding box is always rectangular and oriented as the space basis. However these properties may be lost if the box is observed in the perspective of another space (as shown below).



Path Bounds vs. Visible Bounds

A bounding box may or may not fit the path stroke or other border effects that affect the *visible bounds* of a page item, such as rounded corners.

InDesign's GUI always renders visible-bounds-dependent bounding boxes, which the Scripting DOM refers to as `outerStrokeBounds`[4] or `visibleBounds`, in contrast with the inherent path bounds, referred to as `geometricPathBounds`[4] or `geometricBounds`.

At the scripting level one can always refer to either the "path box" or the "visible box," meaning that *two* coordinate systems are actually available for any spline item we consider. Selecting the right system is of primary importance when moving or resizing objects within a specific region of your layout.

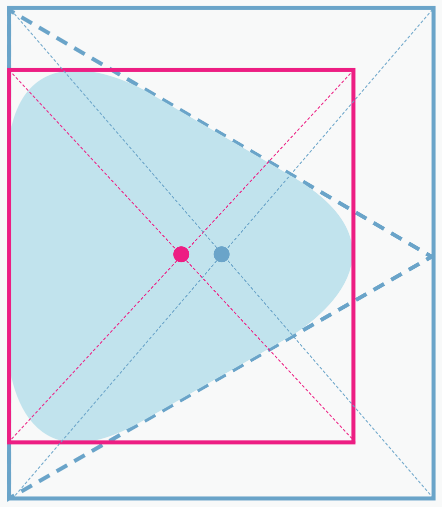
4. Those two values are exposed by the `BoundingBoxLimits` enumeration, which plays an important role in the `resolve` method, as we shall see later.

The figure below (see **Figure 24**) shows that a visible box can be larger, or narrower, than its corresponding path box. As an immediate consequence the associated coordinate systems do not match. For instance, the path box system origin (blue point) is not at the same location as the visible box system origin (magenta point.)[5]

However, relative to some coordinate space, the path box and the visible box of an object always have the same orientation. Therefore we could describe the relationship between these two systems by a transformation matrix that only involves, in the worst case, SCALING and TRANSLATION components.

Also, since **Spread** and **Page** objects never undergo stroke effects, their inner path[6] and visible boxes are always identical.

5. We can even show that the visible box and the path box do not necessarily share the same center point `(0.5,0.5)`:



6. In fact, talking about "path" boxes is something of a misnomer for spreads and pages, as those objects are not spline items at all.

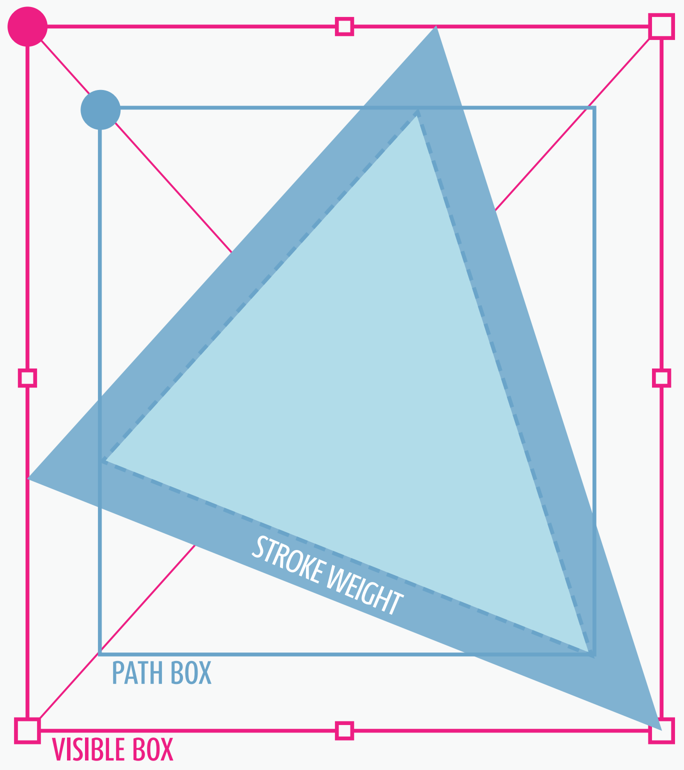
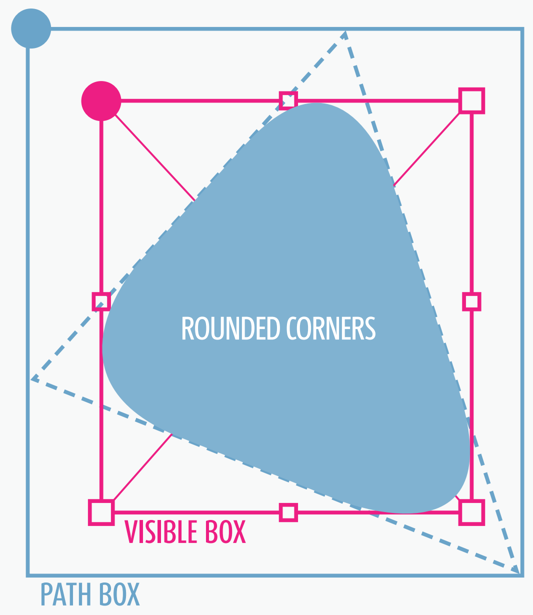


Figure 24.
Depending on applied stroke effetcs the visible box (magenta) can be larger or smaller than the path box (blue.) In the top figure the triangle has a solid border (outer stroke) that increases its visible box. In the bottom figure rounded corners have been applied to the triangle, so that the visible box is smaller than the path box.



SUMMARY

Given a coordinate space S and an object O being either a page item, a page, or a spread, the *visible bounding box* of O *relative to* S is the smallest rectangle in S that encloses O with respect to S's basis.

In addition, the *path bounding box* of O *relative to* S is the smallest rectangle in S that encloses the path of O (disregarding stroke effects) with respect to S's basis.

Each bounding box determines a coordinate system (not a coordinate space) having its top-left anchor point as the origin, the basis being defined by the top-left to to top-right vector (x-axis) and the top-left to bottom-left vector (y-axis.)

EXERCISES

001. Let T be an equilateral triangle having some side aligned with the bottom line of the associated path box. Express the location of the geometric center of T in the inner path box system.

002. Suppose that InDesign's GUI displays the bounding box of a page item as a non-rectangular frame. Explain why this frame is anyway a parallelogram. Show that the transformation matrix which maps the inner space of that page item to the pasteboard space necessarily contains some SHEAR component.

003. Can the *path* box and the *inner* box of a spline item coincide—in any given space—if this object has a nonzero outer stroke weight?