

## Advanced Whiteboarding 4: Containers & Restrictors

Containers and Restrictors are settings made in the Property Browser . See **Advanced Whiteboarding 3: Introduction to Properties** if you haven't already.


**Note:** Since the upgrade to version 1.5, I've noticed the **Block** ability described below is occasionally unreliable – now and again, moving objects slip past a blockable object. I've enquired on the Promethean website, but I'm told it's always been a bit fickle. Strange that I haven't noticed it before.


### Restrictors


The **Restrictor** group of properties mainly controls how an object moves, or interacts with other moving objects.

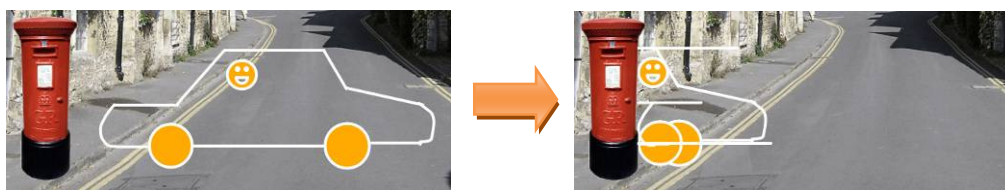
#### Can Block

If set to true, no other object may pass through it (i.e. they cannot overlap). Note that this restriction is one-way; the blocking object may still be placed over another object, unless that object is also set to block.

Blocking objects are ignored if an object is moved using Freely Move , or moves along a path (see **Can Move** below).

Restrictors	
Can Block	False
Can Snap	True
Snap Point x	0
Snap Point y	0
Snap To	Bottom Left
Can Move	Freely
Move Path	
Can Size	Freely
Interaction Mode	Any

Grouped objects and blocking objects don't mix. If a group of objects (specifically, a set of objects connected using Group ) hit a blocking object, the grouped objects will “crumple”, messing up their positions:




(I thought this issue might have been resolved in ActivInspire version 1.5 – it appears not.)

#### Can Snap

Use this in conjunction with Grids. By default, objects are set to snap. **Snap To** determines which part of the object snaps to the nearest grid point. By default, this is set to **Bottom Left**, but in cases such as plotting points or positioning the centre of a circle, you'll want this switched to **Centre**. (Despite reading the manual, I don't understand what **Snap Point x** and **Snap Point y** are meant to accomplish - sorry!)

#### Can Move

By default, this is set to **Freely**. Even when another value is set, Freely Move  overrides the restriction (unless set to **No**, which hides Freely Move).

**Horizontally** and **Vertically** restricts the object so it can only move in those directions.

**Along path** will do nothing until you select an object for the **Move Path** property. If the object used for the path is an annotation or shape, the moving object will follow its outline. Otherwise, it will follow the rectangular boundary of the path object.


One disappointing aspect of paths is that objects can “jump” from one part of a path to another. For example, if an object is restricted to a U-shaped path, it can jump from the left arm to the right arm without needing to go round the curve at the bottom.

Also disappointing is that objects moving on a path cannot be Blocked.

If you don’t want an object to move at all, set to **No**. However, trying to move such a restricted object will make it rotate instead. To stop this side effect, change its **Rotate**: **Can Rotate** property to **No** as well.

### Can Size

Switch to **No** to prevent resizing. You’ll notice the sizing handles do not appear the next time you select the object.

**Hint:** The difference between an object with properties set so it cannot move, rotate or be resized, and an object that is Locked , is that the former can still be selected, giving you access to its marquee handle, etc. But Locking an object is quicker than setting all three properties.

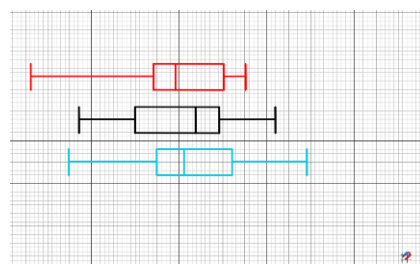
**Interaction Mode** is new in version 1.5, and is only relevant to the new types of Promethean boards that allow both pen and touch control.

## Restrictor Examples

### Box and Whisker Diagrams



This example can be seen in the flipchart that accompanies these notes.

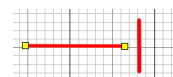
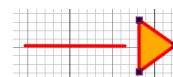
These box and whisker diagrams (aka box plots) use restrictors and a snapping grid to make them easy to adjust.






The vertical bars are restricted to move horizontally, and snap to the grid. The horizontal lines are connectors, that stretch as required as the bars are moved (no need to set any properties for the connectors).

The vertical bars are actually made from flattened triangles, which allows a connector to “anchor” on to the halfway point of the line. Here’s how it’s done:

1. Start with a connector and a triangle. Rotate the triangle 90° (for accuracy, you can use properties. Set **Position**: **Angle** to 90).
2. Select the triangle, then click the Edit Shape Points  marquee button.
3. Grab the rightmost point and move it halfway between the other two, forming a vertical line.
4. Select the connector and click Edit Shape Points .
5. Move the rightmost point onto the middle point of the triangle (it will appear when you’re close enough) to connect the two together.



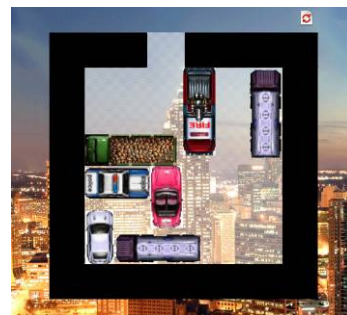
If you set the flattened triangle’s properties first, then Duplicate  it for the rest of the box and whisker diagram, the copies will have the properties already set. Join them up with connectors, and it should be complete. You’ll probably need to use **Freely Move**  or switch **Design Mode** to orange  to override the restrictors until you get everything correctly positioned.

## Rush Hour Puzzle

*This example can be seen in the flipchart that accompanies these notes.*

These sliding puzzles can be found in various forms on the internet, and also come in “real world” versions. So an ActivInspire version may seem a little pointless, but it does demonstrate the use of blocking and movement restriction.

The object of the puzzle is to move the pink car off the playing area via the gap in the border.



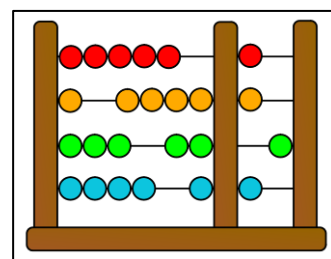
The black border of the playing area is made up of rectangles that each have **Can Block** set to **True**. They also cannot be moved, rotated or resized.

Each vehicle also has **Can Block** set to **True**, and has **Can Move** set to either **Horizontally** or **Vertically**, depending on the direction they face.

*The example included in the flipchart plays a sound effect when the pink car successfully escapes the traffic jam. This is done using a **container** – see below for information about containers.*

## Abacus

The goal here is to create circles that slide along the rails without passing through each other, or moving off the ends. Therefore, the brown stand objects and the circles must all be set to block.



However, if you set the circles to move along a path (with the paths being the relevant horizontal line), the blocking will not work, as blocking is ineffectual when paths are used. Instead, set the circles to move horizontally, once they're correctly positioned on the lines.

**Tip:** If you really want them all lined up perfectly on the lines, set them to move along a path (the line) first – this places each one centrally – then change them all to horizontal movement.

**Note:** ActivInspire doesn't have any sense of physics, so you have to move each circle in turn – it's not possible to move the end one and push the others along with it. Shame.

## Rotate

Although separate from the [Restrictors](#) property group, the [Rotate](#) property group offers settings that are also restrictive, and can even affect / override movement restrictions. The most obvious one is that rotating objects are never stopped by blocking objects.

**Hint:** If you want an object to rotate when you click and drag it, set its [Restrictors](#): [Can Move](#) property to **No**.

Rotate	
Can Rotate	Freely
Rotate Step	45
Rotate About	Centre
Rotate Object	
Rotate Point x	50
Rotate Point y	50

### Can Rotate

Set whether the object can rotate **Freely**, **Clockwise**, **Anticlockwise** or not at all (**No**).

### Rotate Step

By setting a certain amount of degrees, you can make the object “jump” or “snap” to certain angles, like the second hand of a clock moving in steps of 6°.

### Rotate About



This determines how the object rotates, and links with the last three properties of this group. Normally, an object will rotate about its **Centre**. But if you wanted an indicator for something like an ammeter or petrol gauge, you probably want a line or arrow that rotates about its **Bottom**.

Additionally, you can have objects rotate around **Other Objects**, like the Moon rotating about the Earth. Set the object to rotate around using the [Rotate Object](#) property.

A couple of points on this:

1. You can't model a mini-solar system using this, as the Moon rotating round the Earth rotating round the Sun won't work as you think it will or would like it to. Try it and see! It's particularly fascinating to watch if the Moon and Earth are also Grouped together. Looks cool, but it ain't science...
2. If you set two otherwise immobile objects to rotate around each other, you can get them to break their movement restriction by effectively swinging around each other to get across the page, a bit like someone trying to walk with both feet nailed to a plank.

Finally, you can also set an object to rotate about a specific point (confusingly called **Other Place**). Set [Rotate Point x](#) and [y](#) for this. Unfortunately, the coordinates aren't Page coordinates; they appear to be some kind of point relative to the current location of the object. Better answers on a postcard, please.

**Note:** If the Maths Tool **XY Origin**  is being used, what happens to a rotating object becomes a little more ambiguous. If using the marquee handle [Rotate](#) , it will often rotate around the **XY Origin** point, whereas just dragging it when it is set to no movement will usually obey the [Rotate](#) property settings. It all depends on the circumstances, and possible the weather. Test thoroughly before using in the classroom...

## Containers

Containers add an extra layer of interactivity to your flipcharts in a fairly simple way.

*(Very simple containers were covered in the second session, **Building Blocks**. In that session, rectangular shapes were turned into “Notelets”, allowing annotations to be written on them and rearranged on the page. See that session for more details.)*

Effectively, any object can become a container but, regardless of the object used, the containing area will be the rectangular boundary of the object. So it usually looks best if containers are objects with a certain amount of interior space. Otherwise, it just looks wrong.

Unfortunately, there are a couple of odd choices in the way containers have been set up in ActivInspire that can cause a lot of headaches if you don't know about them. Read on carefully.

Container	
Can Contain	Keywords
Contain Object	
Contain Words	adjective
Contain Rule	Completely Contained
Reward Sound	False
Reward Sound Location	camera.wav
Return if not Contained	False

### Can Contain

If set to a value other than **Nothing**, this turns an object into a container. Set to **Anything** and, if an object is placed on top of it (and “fits” – see **Contain Rule** below), then the placed object will be contained and will move with the container when the container is moved. **Note:** *the container must be lower in the layer than the object to be contained, or on a lower layer altogether. Containing only works one way – you can't “scoop up” an object by sliding a container underneath it, the object must be placed onto the container.*

If set to **Specific Object**, then you need to set **Contain Object** to whichever object the container is meant for. Once set, this is the only object the container will hold. Such containers will only contain one object.

If set to **Keywords**, you need to enter the keywords in **Contain Words**, either using the **Keywords Editor** or just entering each word with a space between them (making sure you press Enter afterwards, or it won't remember!). See below for more information about Keywords. Such containers can contain multiple objects.

### Contain Rule

This determines how a container checks the object is fully contained. Usually, you want this left to **Completely Contained**, which means you need to make sure the container is big enough to encompass the whole rectangular boundary of the objects it's to contain.

If you need to contain an object bigger than the container, you should switch to **Centres Must Match**. Then the object is deemed to be in the container if their centres roughly align. (*Hint: the tolerance of this alignment can be adjusted in the **Settings** menu, under **Flipchart Objects** / **Container centre tolerance**.*)

### Reward Sound

If set to **True**, you need to set a sound file to be played when an object is correctly contained in the **Reward Sound Location** property.

### !!! Return if not Contained !!!

This is the annoying part of containers in ActivInspire, so make sure you take this information on board: this property, unlike the others in this group, applies to the objects to be contained, not the container object. Got that? Read it again just to be sure, because there isn't much around to give this little known fact away, and not realising it will make containers really confusing to use.



There's another important fact coming up in a second too, so keep reading carefully.

By default, this property is set to **False**. This means you can place an object onto a container it doesn't belong with, and you'll only notice it's not really contained by one of two ways:

- The reward sound, if set, won't play.
- When you move the container, the object will be left behind.

For best effect you'll want to set this to **True**. Then the object will always jump back to its starting location unless it is placed into a valid container.

Unfortunately, setting this property to **True** brings up another curious design choice that just adds fuel to the fire of confusion: once an object is correctly contained, **the next time it is moved, it can be moved anywhere**. That doesn't sound like a big deal, and it's meant to allow you to move an object out of a container onto the page, even when **Return if not Contained** is set to **True**.

However, it also means you can move an object out of a correct container and into an incorrect container, and the object will stay there as if **Return if not Contained** was set to **False**. Worse still, if you move the object again, out of the incorrect container into another incorrect container, or back onto the page, it will jump back into the incorrect container! It's simply obeying the setting of "returning if not contained", but it will look like it thinks it's in the correct container.

So, when using this, remember that once an object is properly contained, make sure its **next** move is onto the page, never into another container. You have been warned!

### Identification: Keywords

For the Keywords setting of a container to work correctly, the objects to be contained must have matching keywords. These keywords are set in the **Identification** property group.

Identification	
Name	Fly
Keywords	verb noun
Question Tag	None
Assessment Tag	None

Keywords are just single words, and can be anything you like. Each object can have several keywords, just as a container can accept several keywords. **As long as at least one keyword matches between the container and the object, the object will be accepted.**

This means, if you want a container that only accepts, say, objects that are red circles, it's no use having the container accept both keywords "red" and "circle" and assigning red circle objects the **Identification: Keywords** "red" and "circle". While the container will accept these objects, it will also accept red squares with the keywords "red" and "square", as well as yellow circles with the keywords "yellow" and "circle".

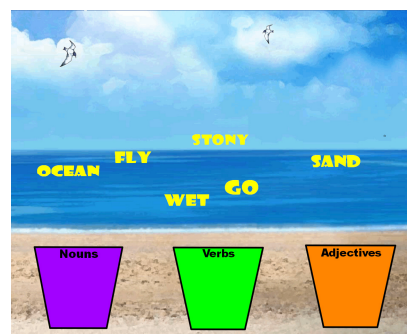
Instead, you have to create keywords such as, "RedAndCircle", with no spaces. It may look silly, but that's the way computers work.

## Container Examples

### Keywords – Types of Words

*This example is included in the flipchart that accompanies this session.*

This example can be easily adapted to a myriad topics focussing on categories (e.g. types of numbers, climates, animals). Here's a word example.



1. Create three “bucket” shapes (the containers). *(Remember, the containing area is the rectangular boundary of the shape.)*
2. For each container, set their **Container: Can Contain** property to **Keywords**.
3. Also for each container, in the **Container: Contain Words** property, enter “noun” for the first, “verb” for the second and “adjective” for the last.
4. Create several text objects: a mixture of nouns, verbs and adjectives, including some that fit into more than one category.
5. For each word, in its **Identification: Keywords** property, enter “noun”, “verb” and/or “adjective”, whichever and all that apply. *Either use the Keywords editor to enter each, or just type them straight into the property box leaving a space between each word – make sure you press Enter afterwards. Be sure you spell them the same way as in step 3! (I’m fairly sure case is irrelevant.)*
6. Also for each word, set **Container: Return if not Contained** to **True**. *(Make sure you’ve read the warnings about this property above!)*

You should find each word will now only stay in the container(s) that matches its keywords.

## Specific Objects – Definitions

*This example is included in the flipchart that accompanies this session.*




This use of containers is unusual but very effective, creating a more interesting (and self-checking) matching exercise than just having someone draw lines on the board.

The pictures all depict a type of phobia (the flipchart example uses the Label property to create pop-up clarifications of each phobia). The object is to match each picture with the correct word, by placing the green triangles on the left into the matching boxes on the right.

Incorrect connections will snap the triangles back to their starting point.



Here's how it's set up:

1. The green triangles are actually modified Diamond shapes, using Edit Shape Points  to create the triangle – see the Box and Whisker example above for a similar trick.
2. Each triangle is joined to a connector (hence the need for edited Diamond shapes, to create the anchor point halfway along one side).
3. Each triangle has its **Container: Return if not Contained** property set to **True**. *(Make sure you've read the warnings about this property above!)*
4. Each box on the right is a container, with **Container: Can Contain** set to **Specific Object**.
5. The **Contain Object** in each case is one of the triangles. *(Giving each triangle a unique name will make this step much easier!)*
6. Position each triangle and connector next to an item to be matched on the left, then position each box container next to an item on the right, ensuring the triangle/container pairs are next to matching items. *(Using Freely Move  or switching Design Mode to orange  will help with the initial placement.)*

That's all there is to it. The connector will stretch as the triangle is dragged towards one of the boxes. If the triangle is accepted by the container, it will remain in place. Otherwise it will snap back.