Blockbusters: Unleashing the Power of Dynamic Blocks – Revealed!

Matt Murphy - ACADventures

GD201-3P Discover the full potential of Dynamic Blocks in AutoCAD. Learn how each parameter and action behaves and how these tools can be applied and implemented into your existing design processes. See over 20 examples of how Dynamic Blocks can be assembled – from scratch – to enhance and automate your office's design processes. Also featured are AutoLISP code examples that demonstrate how to manipulate Dynamic Block settings.

About the Speaker:

Matt has been recognized by Autodesk as a leader in providing professional training since 1985. He has served as an advisor and consultant to the training industry as chair of the Autodesk Training Center Advisory Board (ATCAB) and Executive Committee (ATCEC). Matt is also an Autodesk Approved Instructor (AAI) and a Certified Technical Trainer (CTT). He has received numerous instructional awards, including the ATC Eastern Regional Instructor of the Year for 2004. A widely acclaimed top trainer and featured columnist for *CADalyst* and *AUGIWorld* magazines, he has presented at Autodesk University for 14 years and was the top Autodesk University speaker in a large session for 2006.

matt.murphy@acadventures.com

Autodesk University 2007

Introduction

Blocks have been around since the very early days of AutoCAD. They were the first reusable content we created. Our ancestors of AutoCAD built thousands of them and stored them in read-only folders, and we share them for everyone to use. The three basic reasons for using them haven't changed:

1. They are easy to manipulate as they are unified.
2. They provide consistency for standards parts and details.
3. They reduce drawing file size as each block is a single data reference.

In this session we'll look at how Dynamic Blocks are more flexible to manipulate than your existing traditional blocks. In this session we'll cover the following topics:

1. An overview of different parameters and actions
2. Details on how each action of the block behaves and why
3. Specific scenarios of creating unique blocks
4. Troubleshooting common problems including chaining actions
5. Automating Dynamic Blocks with Fields and LISP

I will also reveal many of the secrets that elude most users when creating and using Dynamic Blocks.

What Are Dynamic Blocks?

They are blocks that can have a variable appearance and placement using grip constraints. Dynamic Blocks are flexible and intelligent. They can automatically align themselves with existing geometry. They can have multiple insertion points. They can change visibility, representing multiple versions in one block instead of inserting multiple variations, and you can edit the geometry without exploding them. This enables you to modify the appearance of individual block references rather than searching for other block definitions to insert or redefining the existing ones.

Why Should I Use Dynamic Blocks?

Blocks, an essential part of nearly any drawing, are used to represent real-world objects. Different variations of real-world objects can require you to define just as many variations of blocks. Dynamic Blocks will reduce your block library size. They can be locked and protected from being exploded. They can be placed on a Tool Palette for greater organization and they will make you more productive by maintaining standards and reducing your clicks and picks.

Block References Are Easier to Use

Introduced in AutoCAD 2006, Dynamic Block functionality enables you to edit the appearance of block instances without having to explode them. You can even manipulate a block instance during and after inserting it into a drawing.

Traditional Blocks

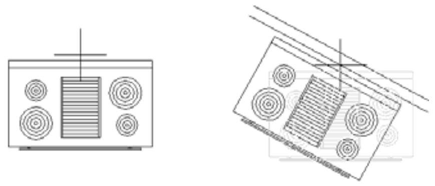
- Multiple steps to place and align
- Design changes require erasing and reinserting or exploding, editing and redefining

Dynamic Blocks

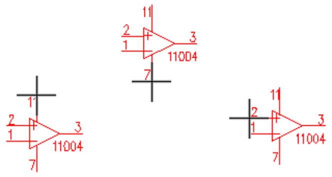
- Automatically align to nearby geometry
- Cycle between multiple insertion points

- Change visibility of geometry to streamline design changes
- You can edit geometry within a block without exploding it

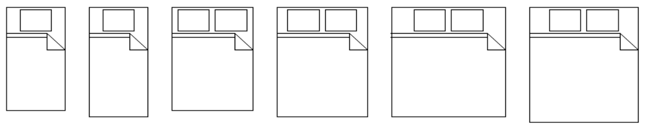
As you move your cursor near drawing geometry, blocks will automatically align themselves with other objects.



As you insert a Dynamic Block, you can cycle between key insertion points to find the one that makes the most sense for your current situation. The ability to cycle through insertion points can eliminate the need for you to move the block after it is inserted.

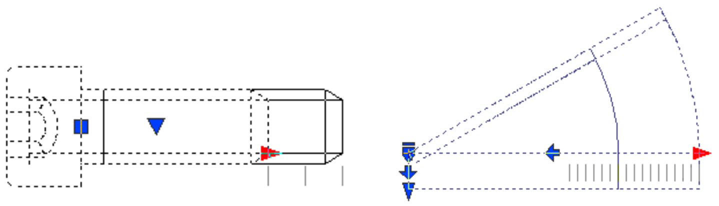


Block definitions can contain multiple representations of a particular symbol. Upon insertion, you can choose which representation to use. For example, a single block definition could store multiple representations of a bed, faucet, door or valve.



Understanding Dynamic Blocks

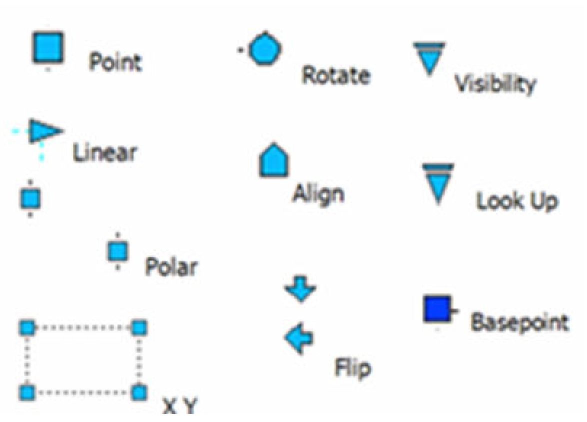
Parameters control the actions of Dynamic Blocks. Parameters are special grips within Dynamic Blocks that enable you to move, scale, stretch and rotate, array and flip individual block geometry. Parameters define the feature of the block that you can change. For example, you might have a bolt block, which you can stretch to a total length of between 1 and 4 units. As you stretch the bolt, the length is constrained to .5-unit increments and threads are automatically added or removed as you stretch the bolt. A second example might be a callout block that includes a circle, text and a leader line. You can rotate the leader around the circle while the text and circle remain static. A third example might be a door block. You can stretch the door width and flip the direction of the door swing.



Working with Dynamic Blocks
Dynamic Blocks will appear with a lightning bolt next to their preview before you insert them.

When you insert a Dynamic Block, you can toggle through or cycle through the multiple insertion points with the CTRL key before you place it.

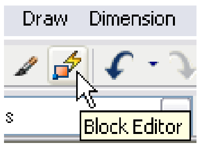
Once you insert a Dynamic Block you can edit it using the Parameter control grip. Now you can grip edit blocks the way we have grip edited other types of AutoCAD geometry. The shape of the grip determines the parameter type.

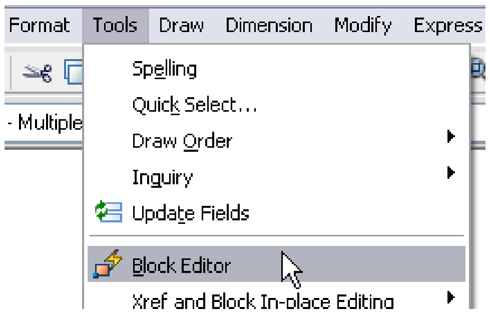


How Do I Create Dynamic Blocks?
The Block Editor enables you to create new block definitions or update your existing blocks. You can access the Block Editor from several locations and then use the block authoring tools to add parameters and actions to your block definition.

Use the Block Editor
The block editing environment is specifically designed for defining blocks. Make sure that the BLOCKEDITLOCK system variable is set to 0. You can access the Block Editor using any of the following methods:

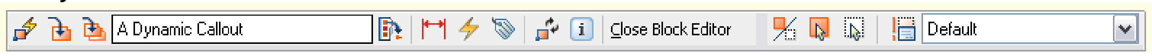
- BEDIT command
- BE command alias
- Standard toolbar
- Tools menu
- Right-click menu with a block selected
- Block Definition dialog box



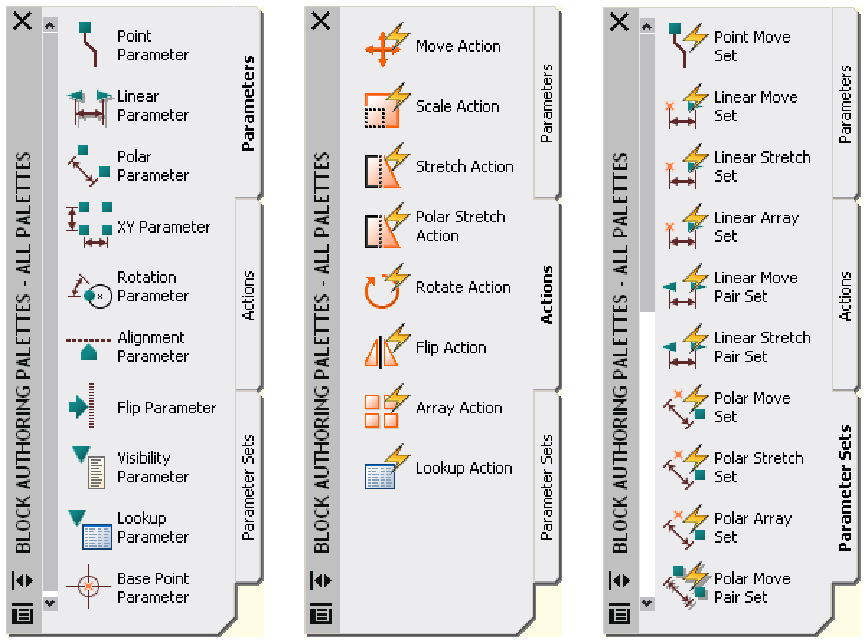


Add Parameters and Actions to Block Definitions

In the Block Editor, you can use typical AutoCAD drawing and editing functionality to create and modify the geometry for your block definition. In addition, the Block Editor includes a toolbar and a block authoring palette, which enable you to apply parameters and actions to your block geometry.



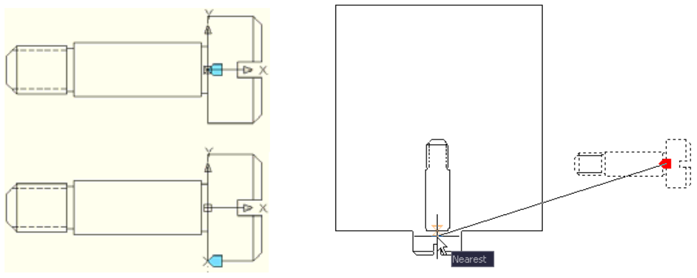
The block authoring palette includes three tabs. The first tab contains all of the available parameters. The second tab contains all of the available actions. And the third tab contains sets of the most commonly used combinations of parameters and actions.



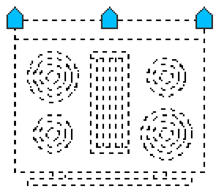
Parameters and actions work together to provide the editing capability of Dynamic Blocks. Parameters are dimensions that drive the block geometry. They are objects with their own relevant properties, which you can edit using the Properties window. For example, you can create a linear parameter to drive the width of a door and then apply properties that constrain the door width to 2-inch increments between the values of 18 and 36 inches. Actions are what change the geometry as you edit a block instance. For example, if you want to change the width of an inserted door block, you must apply a stretch action to the linear parameter that defines the door width.

Each parameter only works with specific types of actions, and a few parameters require no actions. We'll begin with the most basic parameters, the ones that don't require any actions: Alignment, Visibility and Base Point. These parameters are easy to create and they can dramatically increase the efficiency of your existing blocks with minimal effort.

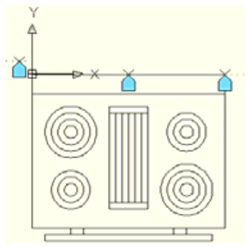
Alignment parameters require no actions although they can be included in the selection set of an action. You can add an alignment parameter to enable a block to align automatically to nearby geometry. If you create the alignment parameter at the origin point of the block, the alignment functionality will be available upon insertion. If you do not add an alignment parameter at the origin point, the alignment capability is available through cycling before you place it or after if you select the alignment grip on an inserted block.



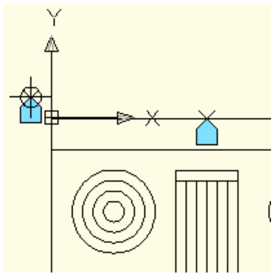
For some blocks, you might find it useful to include multiple insertion points. For example, when inserting a stove block, you might want to insert it using the left corner, the right corner, or the center, depending on the other geometry in the drawing. You can define your blocks with multiple alignment parameters and then use the CTRL key to cycle between the alignment grips upon insertion.



The parameter grips are actual objects in the Block Editor. If you don't want a grip to be included in the cycling options, you can select the grip in the Block Editor and then use the Properties window to turn off the Cycling option. If you want to change the location of the alignment grips, you can move the grip in the Block Editor. For example, if you want the stove to insert a slight distance away from the wall, you would create or move the alignment grip so that it is away from the stove geometry. In this example, changing the location of the left alignment grip means that it is no longer located at the origin point of the block. Of course, you could move all of the geometry so that it is in the correct relation to the origin. However, the simpler option is to insert a base point parameter.

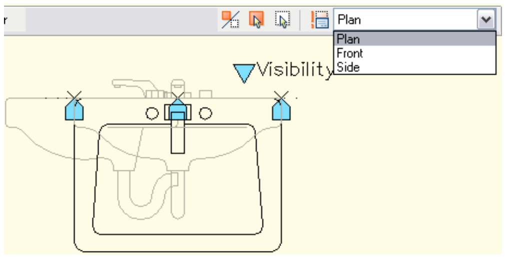


The base point parameter defines a base point for the Dynamic Block reference relative to the geometry in the block, overriding the default origin point of the block. Like the alignment parameter, the base point parameter does not require any actions, but can belong to an action's selection set. The base point parameter displays in the Block Editor as a circle with crosshairs.



You can use the visibility parameter to define a block that turns geometry on or off depending on the state. A visibility parameter must include at least two states and you can use the visibility tools in the upper right corner of the Block Editor to switch between states and to show or hide the geometry for each state. For example, a valve block might include five visibility states. The block definition contains the geometry for all of the states but the visibility of the geometry varies between each of the states.

You can combine multiple types of parameters to create a more powerful block definition. For example, you might create a sink block with a visibility parameter to turn on different nested blocks for the plan, front and side views. In addition, you could include alignment parameters at the appropriate locations in each of the three visibility states.



Tip Revealed: Although you can create individual parameters for each action, that would require you to grip edit three different times (one for each action). Instead, create one parameter and let it drive all three actions. In this example, you know you need a parameter that supports the Move, Stretch and Scale actions.

How Do I Know Which Actions Can Be Associated to Parameters?

The following matrix table is a quick reference you can use to determine the appropriate action for each parameter. If you work the matrix backwards by determining the action you want first, then you'll be able to attach the appropriate parameter.

Parameters

	Move	Stretch	Scale	Array	Polar Stretch	Rotate	Flip	Lookup
Alignment								
Visibility								
Base Point								
Point	▨	▨						
Linear	▨	▨	▨	▨				
Polar	▨	▨	▨	▨	▨			
XY	▨	▨	▨	▨				
Rotation						▨		
Flip							▨	
Lookup								▨

Actions

After you determine which combination of parameters and actions will enable you to edit each individual object to meet your needs, you can begin adding the parameters. The process for adding parameters varies depending on the type of parameter. After you add the parameter, you may see an exclamation point indicating that the parameter requires an action. You can either select an appropriate action from the block authoring tools, or you can double-click on the parameter and select from the list of actions that are available for that parameter. For each action that you add, you will select the specific object(s) that you want that action to edit.

You can continue to add parameters and actions to make your blocks more powerful. In the door example, you might want to add a base point parameter and a flip parameter and action. Using the Properties window, you can apply various properties to the actions, parameters and parameter grips. For example, you might want to change the names of the parameters and actions to something more meaningful.

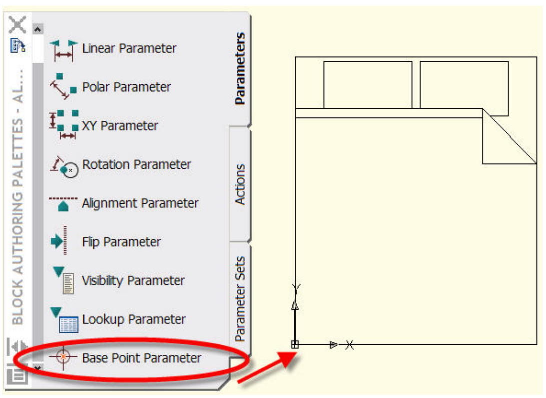
Tip Revealed: It is best to add all of the parameters before adding the actions because you might want to include some parameters in the selection sets of other actions. However, as you are learning to create Dynamic Blocks, you might find it helpful to get one piece working at a time. You can always add to an action's selection set by double-clicking on the action or clicking on the number of selection set objects in the Properties window.

Solutions to Common Problems and Pitfalls

Nothing ever works perfectly. Here are the five most common issues people have posted in the Autodesk Discussion Groups and AUGI forums. If other people had these problems, you will probably encounter them as well.

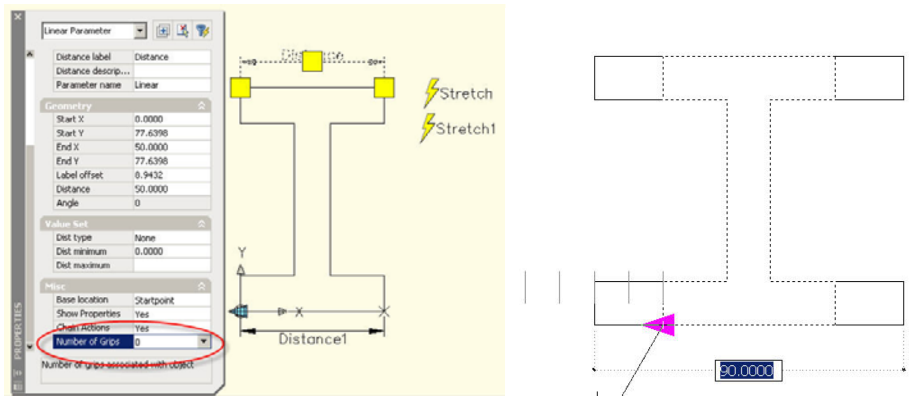
5- Shifty Base Points

Once you've created your dynamic block and then inserted it into another drawing, you may find the base point has shifted and is no longer on the original origin point. This is a common problem when modifying legacy (existing non-dynamic blocks) into dynamic blocks. When using the Block Editor, it makes sense to create the geometry at 0,0 as the Block Editor has its own coordinate origin point. This occasionally causes the block to shift off its insertion point.



Tip Revealed: To correct this problem, simply open the block inside the Block Editor and attach a base point parameter from the authoring palette. Save the block. Any existing blocks of that name or future instances of that block will have the correct insertion point.

4- Hide Those Unwanted Grips
Too many grips? Not a problem. There are times when you'll want to hide some of the grips to prevent additional modifications to the geometry. This is also a common concern when you have parameter control grips that have no editable feature. You can turn off parameter control grips from the Properties palette when you are inside the Block Editor or Bedit command. Choose the parameter, open the Properties palette and choose Number of Grips. This allows you to control the number of grips to display when you are editing an inserted block.

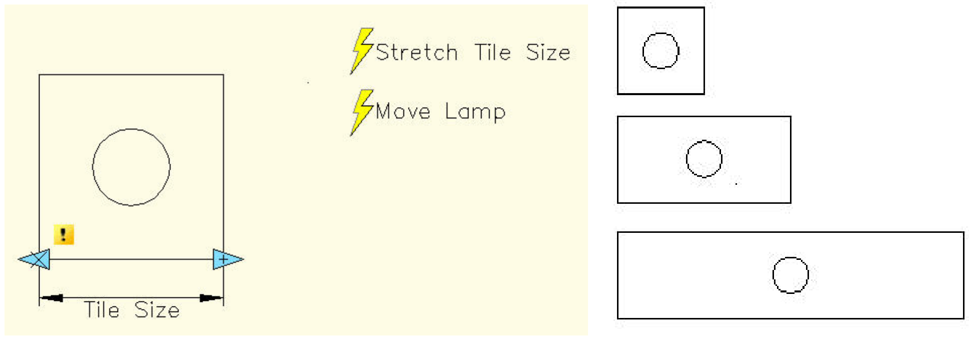


Distance label	Distance
Distance descrip...	
Parameter name	Linear
Geometry	
Start X	0.0000
Start Y	77.6398
End X	50.0000
End Y	77.6398
Label offset	0.9432
Distance	50.0000
Angle	0
Value Set	
Dist type	None
Dist minimum	0.0000
Dist maximum	
Misc	
Base location	Startpoint
Show Properties	Yes
Chain Actions	Yes
Number of Grips	0

Number of grips associated with object

This technique is commonly used when you have two or more actions associated with the same parameter or multiple parameters chained together. This gives you better control by only displaying the parameters you need to edit when you insert the block.

3- Multiple Actions Using One Parameter Control
There are times when you'll want to have one parameter control but have two or more actions occur. If you refer back to the matrix table (on page 8), the relationship between the two actions is simple and direct. You can accomplish this easily. You just attach two actions to the same grip on the same parameter. When you grip edit one action, the other happens at the same time.



In the reflective ceiling tile example, we have one parameter and two actions. By editing the location of one of the linear parameter controls, it does a “Stretch” action on the size of the tile while doing a “Move” action on the circle (lamp) to maintain its location in the center of the tile.

2- Chaining Actions
What we have covered up to this point is two or more actions associated to the same parameter. Occasionally, your geometry may require more than one parameter and you’ll want the second parameter to be activated by the first parameter. This linking of parameters is called a “chaining action.” Here is where the confusion occurs. The chaining feature works by one parameter driving the action associated with two or more other parameters. Notice that I said parameters, not actions! The linking or “chaining” operation is actually linking parameters together…not actions.

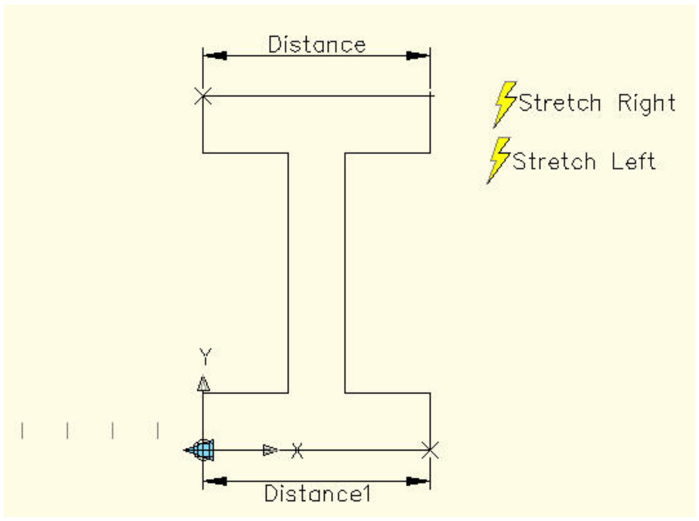
Tip Revealed: There are a few restrictions to getting this technique to work. First, the secondary parameter, the one whose action will be activated by the primary action, must be a point, linear, polar, XY or rotation parameter. These are the only parameters that allow chaining.

Remember that in order for one action to activate another action, both actions must be associated to a parameter. In this case there are at least two actions and two parameters. To create a chained parameter you would do the following:

1. Create the block and open it in the Block Editor.
2. Decide on the parameters you’ll need and their actions.
3. Decide which action you’ll grip-edit in the drawing. This is the primary action that will activate the other, secondary, action.
4. Create both parameters first, before creating any actions.
5. Create the primary action and attach it to its parameter (the primary parameter).
6. When you specify the objects for the primary action, include the parameter of the secondary action. This is very important. But DON’T select or include the objects that will be in the selection set of the secondary action.
7. Create the secondary action, attach it to its parameter, and select its objects.
8. Select the secondary parameter, open the Properties palette, and set its chaining property to Yes.

Let’s take a closer look at the I-beam example below where we first shut off the unwanted parameter control grips.

In order for the I-beam to stretch in both the left and right directions, I used two parameters and two actions, both linear parameters with stretch actions. If you think of how you might do this with a regular AutoCAD command, you would stretch one side of the I-beam and then stretch the other side.



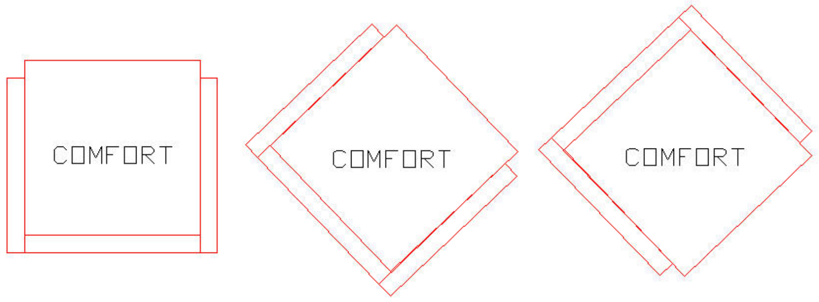
In this modified version of the I-beam, chaining allows me to stretch both sides at the same time using just one parameter control grip to activate the action of both sides stretching.

So, to summarize, the basic principles of chaining are as follows:

- The primary parameter has an action whose selection set includes the secondary parameter in addition to any other objects it will act on. (In this case the action is a stretch action, the stretch frame also needs to include the secondary parameter but NOT the secondary objects or action.)
- The secondary parameter's chaining property is set to Yes.

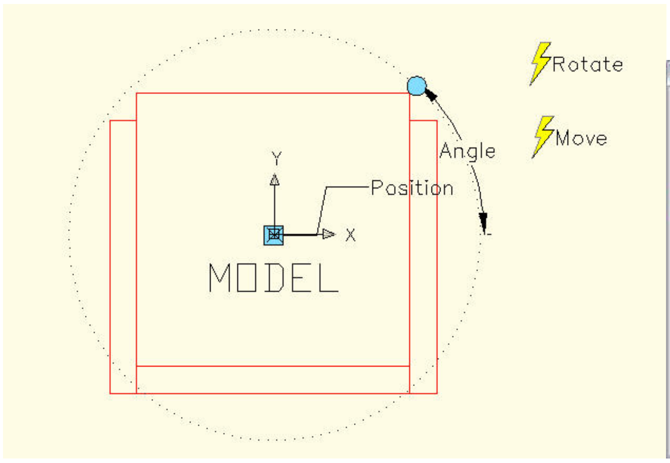
1- Stop Those Spinning Attributes

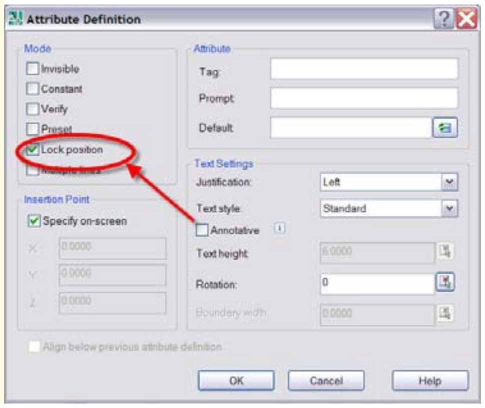
There will be times when you insert a block with attributes and the block needs to be rotated. As soon as you rotate the block, the attributes rotate with it. Wouldn't it be nice if you could stop the attributes from spinning, so they all stay at the correct orientation angle, usually at zero degrees?



The above example of a chair block with an attribute value for the type of chair does NOT rotate when the chair is rotated. This gives you the ability to have attribute text stay at the zero-degree angle or horizontal, making it easier to read. There are two schools of thought on this, depending on how you typically manipulate the block.

Method number one: lock the rotation position of the attribute, so when you rotate the block using the rotate parameter grip, the attribute text will stay locked at the desired angle. The trick to making this work is using the point parameter with a move action on the attribute. Then associate the rotate parameter and action to the object geometry and the point parameter but NOT the attribute. Again, this is the same concept as described above for chaining parameters. This way, when you insert the block at the default angle (zero), and then rotate the block using the rotate parameter grip, the attribute text will NOT rotate. This will keep the attribute text locked in its position but allows you to move it from its location and reposition it as needed.




Attribute Definition

Mode
- [ ] Invisible
- [ ] Constant
- [ ] Verify
- [ ] Preset
- [✓] Lock position
- [ ] Multiple lines

Attribute
Tag:
Prompt:
Default:

Text Settings
Justification: Left
Text style: Standard
- [ ] Annotative
Text height: 6.0000
Rotation: 0
Boundary width: 0.0000

Insertion Point
- [✓] Specify on-screen
X: 0.0000
Y: 0.0000
Z: 0.0000

- [ ] Align below previous attribute definition

Tip Revealed: The common mistake here is NOT to lock the position of the attribute. Locking the position actually locks the orientation, which stops the attribute text from spinning or rotating when you rotate the block. Note that in the attdef dialog box above there is an option for locking the position of the attribute. If you forget to lock the position, you can change that property using the property palette inside the Block Editor. That's wicked cool!

The second method involves adding a rotate parameter and action to the attribute. You would add this if you preferred to insert the block and use the rotate option when inserting the block instead of rotating the block using the rotate parameter. Both methods allow you to change the location of the attribute using parameter control grips but this second method, attaching a rotate parameter and action to the attribute itself, gives you the ability to change the rotation angle of the attribute text after the insertion.

Dynamic Block Automation

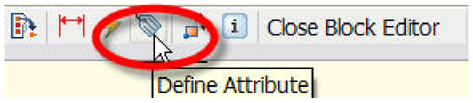
Using Block Properties as Fields within Attribute Definitions

Attributes have always been available to provide valuable text information and intelligence to blocks. Yet attributes still require the user to provide input, which is prone to human error. With the incorporation of Fields we can now eliminate the potential human error when we want the block to provide its property information as attribute text.

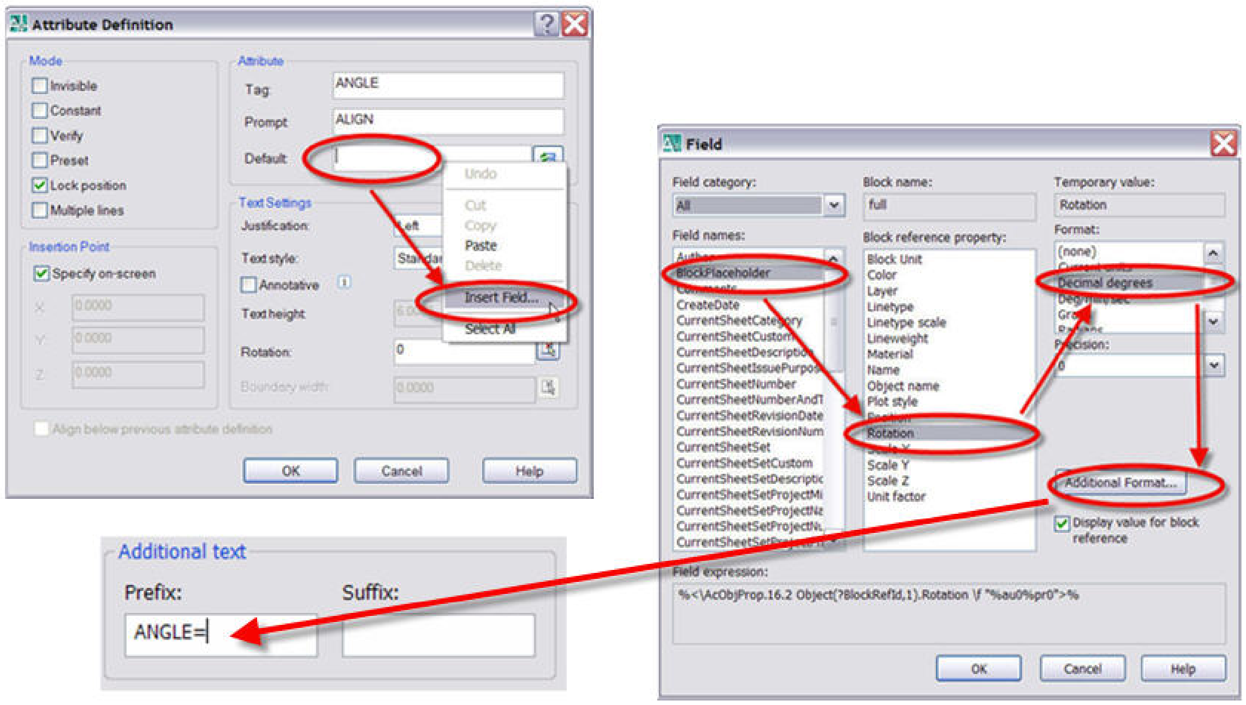
In this example we'll have the rotation angle of the block be displayed as an attribute field when the block is inserted.

The process for creating an intelligent attribute field to a block is very similar to creating an attribute.

1. Create the geometry of the block or open an existing block in the Block Editor BEDIT.
2. While in the Block Editor use the ATTDEF command or choose it from the Block Editor toolbar.



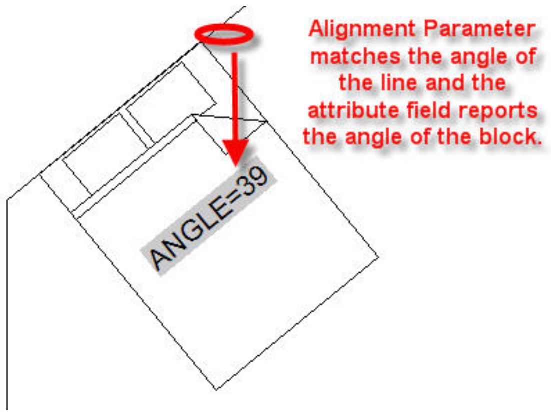
3. Create the attribute text in the normal manner.
4. Instead of creating the "Default" attribute value by typing a value, right-click in the text window and choose "Insert Field…"
5. Entering the values from left to right in the dialog box. Under "Field names:" choose "BlockPlaceholder". For "Block reference property:" choose "Rotation" and for "Format:" choose "Decimal degrees". Steps 4 and 5 are shown in the figure below.



Tag: ANGLE
Prompt: ALIGN
Justification: Left
Rotation: 0
Field category: All
Field names: BlockPlaceholder
Block name: full
Block reference property: Rotation
Temporary value: Rotation
Format: Decimal degrees
Field expression: %<\AcObjProp.16.2 Object(?BlockRefId,1).Rotation \f "%au0%pr0">%
Prefix: ANGLE=

6. Finally choose the "Additional Format…" button and add "ANGLE=" as a prefix.
7. Choose "OK" and place the attribute definition on the block geometry in the Block editor in the typical manner.
8. Add an Alignment Parameter to the block, save and exit the Block editor.

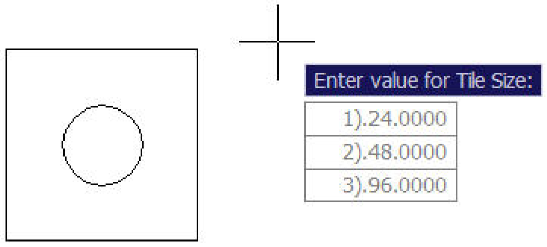
Any future insertions of this block will automatically provide the rotation angle of the Block insertion as an attribute field.


Alignment Parameter matches the angle of the line and the attribute field reports the angle of the block.
ANGLE=39

Using AutoLISP to Automate Dynamic Blocks
One of the most frequent questions I am asked is, "Can you create an on-screen prompt to have actions and values be set before you place the block?" The prompt would be similar to an attribute prompt but would be asking you for the parameter values. This would eliminate the extra step of editing the block, once inserted. When created, this prompt would even eliminate using the Properties palette to modify the block values before insertion.

Prompt for Block Actions and Values upon Insertion
Through research into the AUGI forums and Autodesk discussion groups I have found a number of creative and intuitive LISP routines that can be used to automate the parameter control grip actions and streamline the block editing process.

The following routine creates an on-screen prompt for setting the parameter values and works with all dynamic blocks, regardless of the value type and set.



For additional information on this routine and others, I encourage you to check the AutoCAD forum on AUGI.com. The LISP routine shown below is one I found in the Autodesk Discussion Group. For a complete download of the file, go to Autodesk.com, choose "Autodesk Discussion Groups", choose "AutoCAD" and then "Dynamic Blocks". Do a search for "InsertPrompt.lsp". The file reads as follows:

InsertPrompt.lsp

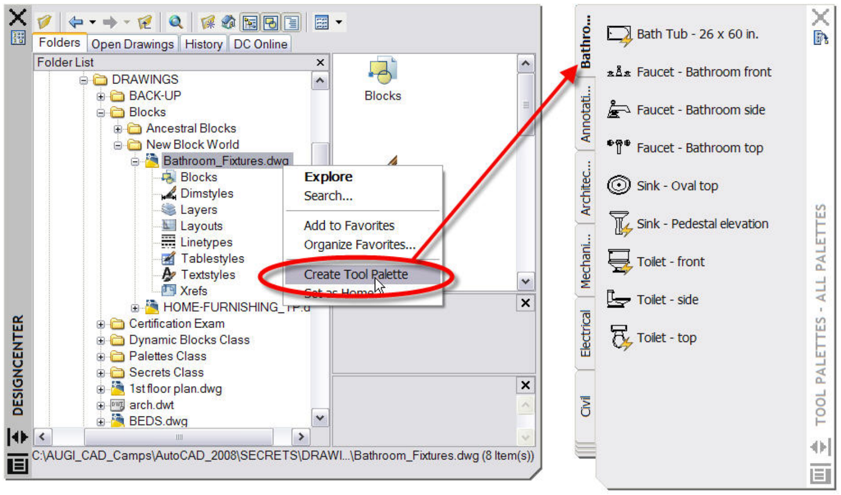
```
(vl-load-com)
(vl-load-reactors)
(defun ss->objlist (ss / cnt objlist)
  (setq cnt (sslength ss))
  (repeat cnt (setq objlist (append objlist (list (vlax-ename->vla-object (ssname ss (- cnt 1))))))
    (setq cnt (- cnt 1)))  )
  (setq ss nil)
  Objlist)
(defun gp:binsertatts (a b / ss obj objattr nwstr)
  (if (or (eq (car b) "EXECUTETOOL") (eq (car b) "INSERT"))
    (setvar "attreq" 0)))
 (defun gp:binsertatte (a b / ss obj objattr nwstr objdyn newvalue prmpts cnt dyn dynp)
  (if (or (eq (car b) "EXECUTETOOL") (eq (car b) "INSERT"))
    (progn
      (setq dyn (getvar "dynmode"))
      (setq dynp (getvar "dynprompt"))
      (setvar "dynprompt" 1)
      (setvar "dynmode" 1)
      (setq ss (ssget "L"))
      (setq obj (ss->objlist ss))
      (foreach o obj
        (if (= (cdr (assoc 0 (entget (vlax-vla-object->ename o)))) "INSERT")
          (progn
            (if (= (vla-get-HasAttributes o) :vlax-true)
              (progn
                (setq objattr (vlax-safearray->list (vlax-variant-value (vla-GetAttributes o))))
                (foreach oa objattr
                  (setq oatr oa)
                  (if (= (vla-get-Constant oa) :vlax-false)
                    (progn
                      (setq nwstr (getstring (strcat "\nSpecify " (vla-get-TagString oa) ": <" (vla-get-
TextString oa) ">: ")))
                      (if (/= nwstr "") (vla-put-TextString oa nwstr))
                      (setq nwstr nil))))))
            (if (= (vla-get-IsDynamicBlock o) :vlax-true)
              (progn
                (setq objdyn (vlax-safearray->list (vlax-variant-value (vla-GetDynamicBlockProperties
o))))
                (foreach od objdyn
(if (and (= (vla-get-Show od) :vlax-true) (= (vla-get-ReadOnly od) :vlax-false)
(/= (vla-get-PropertyName od) "Origin"))
                    (progn
                    (if (= (vlax-safearray-get-u-bound (vlax-variant-value (vla-get-AllowedValues od)) 1)
-1)
                        (progn
                          (if (= (vla-get-Description od) "")
                            (setq prmpts (strcat "\nEnter value for " (vla-get-PropertyName od) ":"))
                            (setq prmpts (strcat "\nEnter value for " (vla-get-Description od) ":")))
                            (cond
                            ((= (vla-get-UnitsType od) acAngular) (setq newvalue (getorient prmpts)))
                            ((= (vla-get-UnitsType od) acDistance) (setq newvalue (getdist prmpts)))
                            ((= (vla-get-UnitsType od) acArea) (setq newvalue (getreal prmpts))))
                            (if (/= newvalue nil) (vla-put-Value od (vlax-make-variant newvalue))))
                          (progn
                          (setq prmpts "[")
                          (setq cnt 1)
            (foreach pt (vlax-safearray->list (vlax-variant-value (vla-get-AllowedValues od)))
             (if (= (vla-get-UnitsType od) 0)
(setq prmpts (strcat prmpts (itoa cnt) ")." (vl-string-translate "/" "|" (vl-string-translate " " "-" (vlax-
variant-value pt))) " "))
(setq prmpts (strcat prmpts (itoa cnt) ")." (vl-string-translate "/" "|" (vl-string-translate " " "-" (rtos
(vlax-variant-value pt) 2))) " ")))
             (setq cnt (+ cnt 1)))
            (setq prmpts (strcat (vl-string-right-trim " " prmpts) "]"))
            (initget 0 (vl-string-trim "[]" prmpts))
            (if (= (vla-get-Description od) "")
(setq newvalue (getkword (strcat "\nEnter value for " (vla-get-PropertyName od) ":" (vl-string-
translate " " "/" prmpts))))
(setq newvalue (getkword (strcat "\nEnter value for " (vla-get-Description od) ":" (vl-string-
translate " " "/" prmpts)))))
            (if (/= newvalue nil)
             (progn
              (setq newvalue (nth (- (atoi (substr newvalue 1 (vl-string-position 41
newvalue))) 1) (vlax-safearray->list (vlax-variant-value (vla-get-AllowedValues od)))))
              (vla-put-Value od newvalue)
             ))))))))))))))
  (setvar "attreq" 1)
  (setvar "dynmode" dyn)
  (setvar "dynprompt" dynp)
  (princ))
(setq rinsrte (vlr-command-reactor nil '((:vlr-commandEnded . gp:binsertatte))))
(setq rinsrts (vlr-command-reactor nil '((:vlr-commandWillStart . gp:binsertatts))))
```

Even if you’re not an expert in AutoLISP, you can still see how the routine is using “get” to retrieve the information of all editable features of the block for prompt and display. This file could be modified to narrow down just the parameters that you want to be prompted for when inserting.

No More WBLOCK!
Although in AutoCAD 2008 you can WBLOCK a dynamic block, the recommended method for creating and storing these blocks is to have them nested in one drawing file. In the old days, we would create a network drive folder and have an individual file for every block insertion, wblock. This is no longer necessary.

Dynamic blocks can be stored in categorized drawing files. These files can then be easily made into Tool Palettes from DesignCenter. This will streamline your block library and eliminate the clutter from hundreds of individual block files.

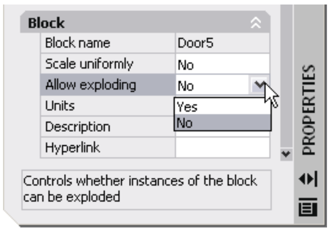
To build a Tool Palette from a drawing file, simple navigate to the file in DesignCenter and right-click on the file. AutoCAD will automatically make a tool palette of all the blocks in the file, using the file name as the palette name.



Don't Forget to Lock Your Blocks!
No longer do you need to worry about your renegade designers exploding your blocks and breaking your standards. Now you can lock your blocks and protect them from being exploded. You can even lock the Block Editor!

Secret Revealed: Lock your block! A new option in the Properties window enables you to prevent someone from exploding the block references. To access this option, open your block definition in the Block Editor. Use the Properties window without any objects selected.

Tip Revealed: Lock the Block Editor! Prevent your renegade designers from editing your Dynamic Blocks with the system variable BLOCKEDITLOCK set to 1. Then undefine the command.



Block	
Block name	Door5
Scale uniformly	No
Allow exploding	No
Units	
Description	
Hyperlink	

Controls whether instances of the block can be exploded

Final Tip Revealed: This class wouldn't be complete without a list of the common system variables that affect Dynamic Blocks.

Dynamic Blocks System Variables

BAUTHORPALETTE - Opens the Block Authoring Palettes window in the Block Editor.

BAUTHORPALETTECLOSE - Closes the Block Authoring Palettes window in the Block Editor.

BCLOSE - Closes the Block Editor.

BCYCLEORDER - Changes the cycling order of grips for a Dynamic Block reference.

BEDIT - Opens the Edit Block Definition dialog box and then the Block Editor.

-BEDIT - Opens the Edit Block Definition dialog box and then the Block Editor (command line).

BGRIPSET - Creates, deletes or resets grips associated with a parameter.

BSAVE - Saves the current block definition.

BSAVEAS - Saves a copy of the current block definition with a new name.

RESETBLOCK - Resets one or more Dynamic Block references to the default values of the block definition.

BACTIONCOLOR - Sets the text color of actions in the Block Editor.

BGRIPOBJCOLOR - Sets the color of grips in the Block Editor.

BGRIPOBJSIZE - Sets the display size of custom grips in the Block Editor relative to the screen display.

BLOCKEDITLOCK - Prevents opening of the Block Editor and editing of Dynamic Blocks definitions. Set it to 1. By doing this, when you double click on a block, it will open the REFEDIT feature rather than the Block Editor. By default, in AutoCAD 2006 the BLOCKEDITLOCK is set to 0. This prevents users from editing Dynamic Blocks.

BLOCKEDITOR - Reflects whether or not the Block Editor is open.

BPARAMETERCOLOR - Sets the color of parameters in the Block Editor.

BPARAMETERFONT - Sets the font used for parameters and actions in the Block Editor.

BPARAMETERSIZE - Sets the size of parameter text and features in the Block Editor relative to the screen display.

BTMARKDISPLAY - Controls whether or not value set markers are displayed.

GRIPDYNCOLOR - Controls the color of custom grips for Dynamic Blocks.

INSUNITS - Specifies a drawing-units value for automatic scaling of block, images, or xrefs inserted into or attached to a drawing.

Summary

Adding parameters and actions to your existing block library allows you to increase power and functionality with very little effort. And for those of you with plenty of spare time, you can rebuild your block libraries or build new Dynamic Block libraries with actions associated with parameters. You can also add chaining to link multiple parameters and actions together so they can be controlled with just one parameter control grip.

It’s all about the clicks and picks. If you learn to apply tool palettes for blocks, you’ll never use the insert command again. If you add dynamic parameters to your existing blocks, you’ll never need to use the explode command again. In both cases you will be more productive because you will be able to do your design work with fewer clicks and picks. Reducing repetitive steps and reducing clicks and picks is how you become more productive working with AutoCAD, period! Dynamic blocks will also ensure that you maintain your standards while adding an entire new level of productivity to your design process.