

## Introduction to Macro Writing

Kate Morrical – Autodesk, Inc.

**CP214-1** Do you want to customize your AutoCAD® or AutoCAD LT® software, but don't have the time to learn a programming language? In that case, macros may be just what you need. I'll show you how to combine commands and options into repeatable macros that perform complex operations and can be launched with a single click on an interface element like a toolbar or a ribbon panel. If you want to maximize your productivity while minimizing effort, this class is for you.

### **About the Speaker:**

Kate joined Autodesk in 2008 as the Technical Marketing Manager for AutoCAD LT®. She started using AutoCAD® in 1999 with Release 14, and has been blogging about LT since April 2007. Prior to joining Autodesk, Kate was a structural engineer and CAD manager for the Washington, DC, office of Robert Silman Associates, PLLC, where she provided technical support for 20 users of AutoCAD, AutoCAD LT, and Revit® Structure. She has a B.S. in Civil Engineering from Case Western Reserve University.

[kate.morrical@autodesk.com](mailto:kate.morrical@autodesk.com)

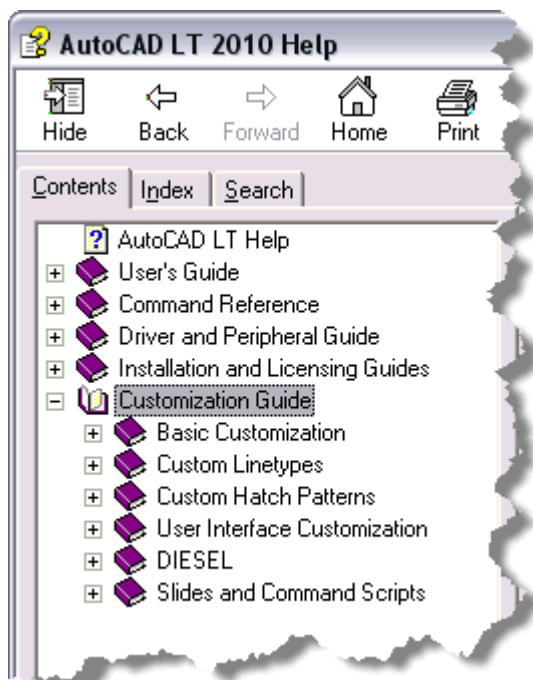
## Introduction

First of all, I'd just like to say that I'm not a programmer. I'm guessing most of you aren't either. There could be all sorts of reasons for this— too busy (who isn't?), programming languages don't work with the program (AutoCAD LT)—but you and I have just never learned how to write programs.

That doesn't mean we're content with the out-of-the-box configuration, though. No, we want to set up our programs just the way we want it, and to be able to quickly repeat commands and command sequences we use on a regular basis.

Fortunately, plenty of customization capabilities are built right in to AutoCAD and AutoCAD LT, and you don't need to be a coding expert to use them. In fact, all you need is a little knowledge of the command line and the CUI editor.

To give you an idea of what's available, here's a screenshot of the AutoCAD LT Help file. At the bottom of the Table of Contents, there's an entire section called "Customization Guide."



Can you believe how many topics are in there? The AutoCAD Help file looks just the same, except it has a few more links for external programming languages like LISP and ARX.

Having so many customization options means that they can't all possibly fit into a 90-minute class. So I'm going to highlight some of my favorites, and give you some more resources at the end to learn about the rest.

One of those resources is the Appendix, which includes all the macros referenced in the handout, as well as a bunch more I found on the Autodesk discussion groups.

## What's In

This class is focused on the macro-writing aspects of the built-in customization opportunities in AutoCAD and AutoCAD LT. Specifically:

- Writing Macros
- Using DIESEL in Macros
- Using Macros in CUI elements
- Using Macros in Tool Palettes
- The Action Recorder

Of these topics, the only one that doesn't apply to AutoCAD LT is the Action Recorder. Everything else translates perfectly between both programs—and for that matter, between all AutoCAD-based programs, including verticals like AutoCAD Architecture or AutoCAD MEP.

## What's Out

As you saw from the Help file contents, there are also a lot of customization topics that we don't have time to cover, including (but not limited to):

- Templates
- Command aliases
- Linetypes
- Hatch patterns
- Scripts
- Dynamic blocks

While all these can be important and effective tools for customizing your setup and CAD standards, they're not in the same category as macros and the CUI. (Besides, I had to draw the line somewhere.)

This class also excludes LISP, VBA, .NET, and any other external programming languages. There are plenty of other resources for learning how to program (including many excellent classes here at AU 2009 and on AU Online). This class will teach you how to customize without programming.

## Notes on Handout Formatting

This handout involves a lot of command-line input, so it's important to understand when a certain line is an AutoCAD prompt, text or numbers, or input like Enter or a left-mouse click.

Here are the formatting conventions that will be used in this handout:

- AutoCAD-generated command-line text
- **USER-GENERATED TEXT OR NUMBERS**
- *Enter, left-click, or other special characters*

## Writing Macros

The majority of this class is about understanding the syntax and rules for writing macros. Once you've learned how to create macros, you can use them in all sorts of places: pull-down menus, ribbon panels, toolbars, and tool palettes.

### What Is a Macro?

A macro is a text string that combines commands and options for repeated use. It's a way of taking command-line input and storing it for later use, so that one click of the mouse can accomplish a drawing task that would otherwise take a series of actions by a user. A macro can be as simple as calling a command and providing a single input, or a complex series of commands and options.

### Example: Draw a Circle

For now, let's start with something simple. Let's say you want to draw a circle with a radius of 1 unit.

Here's what that would look like at the command line:

```
Command: CIRCLE (enter)
Specify center point for circle or [3P/2P/Ttr (tan tan radius)]:
(left-click)
Specify radius of circle or [Diameter]: 1 (enter)
```

Now, let's convert that string of commands into a macro, one step at a time.

- `circle` Enters the command name.
  
- `;` A macro can't press ENTER for you, so you need to use a semicolon instead. This one starts the CIRCLE command. The effect of `circle;` is the same as if you'd typed `circle` at the command line and pressed ENTER.
  
- `\` Pauses for user input. In this case, the first prompt of the CIRCLE command is for a center point. So the macro will wait to allow you to specify your own center point each time you run the macro.
  
- `1` Provides input for the next prompt. Here, that's the radius. Every time you run this macro, the circle created will have a radius of 1 unit.
  
- `;` Another semicolon, this time to accept the value of 1 and end the command. This one isn't technically necessary. AutoCAD can see that this is the end of the macro, and automatically finishes the command for you. I still like to put them in, though, so that I can see exactly what's happening.

Put it all together, and it looks like this:

```
circle;\1;
```

But this isn't quite finished yet. What if there was another command running when you started the circle command? You'd get an error message, right? So let's cancel any running commands first.

```
^C^Ccircle;\1;
```

This actually runs “cancel” twice, just in case you happen to be doing something that would require more than one step to get out of. Most commands these days would only need one ^C to exit, but it's still considered good practice to have two—just in case.

And one more what-if: What if someone tried to use this macro in a non-English version of AutoCAD or AutoCAD LT? The French version, for example, has its own word for circle. So let's force the macro to use the English command definition by putting an underscore in front of the command name. If you know that your macro will only be used in English, you can leave this out, but it's easy to add and ensures that anyone around the world can also use this macro.

```
^C^C_circle;\1;
```

### Special Characters in Macros

The macro above uses three “special characters”: underscore (\_), semicolon (;), and backslash (\). These are characters that tell the macro to perform certain non-command actions. In addition to these three, there are several other special characters that can be used in macros. A full list can be found in the Help file, but here are some of the most important:

Character	Description
<i>i</i>	Issues ENTER
[blank space]	Enters a space; a blank space between command sequences in a macro is equivalent to pressing the SPACEBAR
\	Pauses for user input
_	Translates AutoCAD commands and options that follow
-	Runs the command line version of a command
*	Repeats the macro until the user cancels it
^	Equivalent to pressing CTRL on the keyboard. You can combine the caret with another character to construct macros that do such things as turn the grid on and off (^G) or cancel a command (^C)
\$	Introduces a DIESEL expression.
@	Uses the coordinates of the last point specified.

The behaviors described above only apply when the characters are used in macros. If you typed these characters at the command line, outside a macro, either they wouldn't work, they'd give you an error, or they'd have a different behavior than they would in a macro.

So if you wanted to use the above macro to create multiple circles at a time, just add an asterisk to the beginning:

```
*^C^C_circle;\1;
```

**Note:** Spaces in macros have the same effect as semicolons. This is important, because stray spaces in a macro can have unexpected effects. I always recommend using semicolons in macros instead of spaces—they're so much easier to count! None of the macros in this handout have spaces. Some of them do run on to more than one line, though, so if you're copying and pasting them, watch out for word wrap.

## Basic Macros

To get started writing a macro, you need to choose the command you want to run and figure out what happens at the command line when you run it. If you're a command line fan, this is probably already second nature to you. If you prefer toolbars or menus, though, all you need to do is run your sequence a few times and pay attention to what's going on at the command line while you do it.

Here's another basic example.

### Example: Rotate by Fixed Angle

Maybe you often need to rotate objects by a fixed angle.

If you did this by typing at the command line, you'd see this:

```
Command: ROTATE (enter)
Current positive angle in UCS: ANGDIR=counterclockwise ANGBASE=0
Select objects: (left-click) 1 found
Select objects: (enter)
Specify base point: (left-click)
Specify rotation angle or [Copy/Reference] <0>: 180 (enter)
```

So to translate this into a macro, just put all the pieces together, remembering to make use of the special characters.

<code>^C^C</code>	Cancel any running commands.
<code>_rotate;</code>	Start the ROTATE command.
<code>\</code>	Pause for user input, so you can select the object to rotate.
<code>;</code>	End the "select objects" mode.
<code>\</code>	Pause for user input, so you can select the base point.
<code>180;</code>	Specify a rotation angle of 180 degrees, and end the command.

Put it all together:

```
^C^C_rotate;\;\180;
```

Obviously this is still a pretty simple example. But the principles can be applied to all kinds of AutoCAD and AutoCAD LT commands, and tweaked to apply best to what you're doing.

### Example: Copy and Rotate

Let's expand on that rotate macro for a minute. You probably know that the ROTATE command has a copy option included now, which enables you to rotate a copy of an object while leaving the original intact. But what if you want to copy the object first, and then rotate it? Try this macro:

```
^C^C_copy;\;\_rotate;L;;@;\
```

`^C^C` Cancels any running commands.

`_copy;` Starts the COPY command.

`\` Pauses for user input, so you can select the object to copy.

`;` Ends the "select objects" mode.

`\\` Pauses for user input, so you can select the base point and the second point. (Even though COPY usually lets you make multiple copies, for some reason here it defaults to a single copy. It works, though.)

`_rotate;` Starts the ROTATE command.

`L;;` Selects the last object created as the object to rotate and ends the selection mode. (If we used "P" here, for "previous", the rotate command would select the original object, not the copy.)

`@;` Uses the coordinates of the last point specified. In this case, that's the end point of the copy.

`\` Pauses for user input, so you can specify the rotation angle.

### Example: Arc with Donut

Here's one more macro that makes use of the "@" function. It creates a leader-like object consisting of a 3-point arc and a solid donut on the end.

```
^C^C_arc;\\\donut;0;0.5;@;
```

The three pauses are there so you can specify the three parts of the arc. The donut has the same inside and outside diameters every time (0 and 0.5 units), and it's automatically placed at the end of the newly-created arc.

## Macros for Dialog-Based Commands

Creating macros based on most commands is very straightforward—all the options you have are right there in the command line. But you can't use a macro to select an option in a dialog box, because macros are completely text-based. So if you want to use a macro to create hatches, insert blocks, or run plots, you have to use the command line versions to do it.

Getting a dialog-based command on the command line is as simple as prefixing the command name with a hyphen (-). For example, `-hatch` or `-insert`.

When using text-based versions of dialog boxes, you have to pay attention to the order in which options and settings are presented. In a dialog box, you can pick any button or box at any time, but that's not necessarily the case with command-line commands.

### Example: Create a Hatch

Here's an example of the command-line syntax you might use to create a hatch:

```
Command: -HATCH(enter)
Current hatch pattern: ANSI31
Specify internal point or [Properties/Select objects/draw
boundary/remove Boundaries/Advanced/DRaw order/Origin/ANnotative]:
P(enter)
Enter a pattern name or [?/Solid/User defined] <ANSI31>: ANSI37(enter)
Specify a scale for the pattern <1.0000>: 4(enter)
Specify an angle for the pattern <0>: 45(enter)
Current hatch pattern: ANSI37
Specify internal point or [Properties/Select objects/draw
boundary/remove Boundaries/Advanced/DRaw order/Origin/ANnotative]:
S(enter)
Select objects: (left-click) 1 found
Select objects: (enter)
Current hatch pattern: ANSI37
Specify internal point or [Properties/Select objects/draw
boundary/remove Boundaries/Advanced/DRaw order/Origin/ANnotative]:
(enter)
```

This sequence sets the properties of the hatch to be the ANSI37 pattern with a scale of 4 and an angle of 45, and uses a pre-drawn object as the hatch boundary. Notice that once we entered the properties section, we had to specify pattern, then scale, then angle. It's one example of how sequence matters in a macro where it wouldn't in a dialog box.



You'll notice that we could have specified an internal point, or drawn the boundary from scratch. Other boundary options, like retention and island detection, would be available through the advanced option. We could also have set the draw order, origin, and annotative properties of the hatch with this macro.

Replacing all the "enters" with semicolons and the left-clicks with backslashes results in the following macro:

```
^C^C-hatch;P;ANSI37;4;45;S;\;;
```

Hatches can also be inserted from Tool Palettes (see section later in this handout).

### Example: Insert a Block

Like HATCH, using INSERT at the command line means that some of your options are presented in a specific order.

```
Command: -INSERT (enter)
Enter block name or [?]: SAMPLE (enter)
Units: Inches Conversion: 1.0000
Specify insertion point or [Basepoint/Scale/Rotate]: S (enter)
Specify scale factor for XYZ axes <1>: 12 (enter)
Specify insertion point or [Basepoint/Scale/Rotate]: R (enter)
Specify rotation angle <0.0000>: 90 (enter)
Specify insertion point or [Basepoint/Scale/Rotate]: (left-click)
```

In this case, the block is specified to have a scale factor of 12 and a rotation of 90 degrees. Replacing "enters" and "left-clicks" with semicolons and backslashes gives us this:

```
^C^C-insert;sample;S;12;R;90;\
```

You can also replace any of the numbers here with backslashes for user input.

Sometimes you want blocks to be exploded on insertion, so their contents can be edited immediately. (I used to do this for typical detail files, which needed a new sheet name and number—with the same title—every time.)

The easiest way to accomplish this is to precede the block name with an asterisk.

```
Command: -INSERT (enter)
Enter block name or [?]: *SAMPLE
Specify insertion point for block: (left-click)
Specify scale factor for XYZ axes: 12 (enter)
Specify rotation angle <0>: 90(enter)
```

Notice that the order of the prompts here is a little different using this method. You don't need to enter R or S for rotation and scale, because you're prompted for them by default. Here's what the macro looks like with all the "enters" and "left-clicks" replaced:

```
^C^C-insert;*sample;\12;90;
```

A side-effect to this method is that the block definition isn't added to the drawing if it's not already there. This can either be a good thing or a bad thing, depending on your workflow. Another side-effect is that you can't see a preview of the block before you insert it. So, if seeing where to place the block is more important than keeping the definition out of the drawing, you can use the following macro. It uses EXPLODE with the "last" selection option to explode the most recently created object—in this case, the newly-inserted block.

```
^C^C-insert;sample;S;12;R;90;\explode;L;
```

You might also want the best of both worlds, where you can see the block as you place it but still keep the block definition out of the drawing. In that case, tack the PURGE command onto the end of the macro.

Command: **-PURGE (enter)**

Enter type of unused objects to purge

[Blocks/Dimstyles/Layers/LTypes/Materials/Multileaderstyles/Plotstyles/SHapes/textStyles/Mlinestyles/Tablestyles/Visualstyles/Regapps/Zero-length geometry/Empty text objects/All]: **B (enter)**

Enter name(s) to purge <\*>: **SAMPLE (enter)**

Verify each name to be purged? [Yes/No] <Y>: **N (enter)**

Deleting block "sample".

1 block deleted.

The resulting macro would look like this:

```
^C^C-insert;sample;S;12;R;90;\explode;L;-purge;B;sample;N;
```

This only works if no other copies of the block exist in the drawing when you run the macro, because you can't purge objects in use.

**Note:** When using a macro to insert a block, the block you're inserting either has to already exist in your drawing, or a file with that name must be in your support file search paths. You can get around this by including the entire path to the block file in the macro, but I've found it less cumbersome to keep the macros short and control the block location with the support file paths.

**Example 3: Change Layer Settings**

Using macros is a great way to create layers in a drawing or reset layer properties. Here are the steps for creating a red, continuous layer called S-ANNO-TEXT.

```
Command: -LAYER (enter)
Current layer: "0"
Enter an option
[?/Make/Set/New/Rename/ON/OFF/Color/Ltype/LWeight/MATerial/Plot/Freeze
/Thaw/LOck/Unlock/stAte/Description/rEconcile]: M (enter)
Enter name for new layer (becomes the current layer) <0>: S-ANNO-TEXT
(enter)
Enter an option
[?/Make/Set/New/Rename/ON/OFF/Color/Ltype/LWeight/MATerial/Plot/Freeze
/Thaw/LOck/Unlock/stAte/Description/rEconcile]: C (enter)
New color [Truecolor/Colorbook] : 1 (enter)
Enter name list of layer(s) for color 1 (red) <S-ANNO-TEXT>: (enter)
Enter an option
[?/Make/Set/New/Rename/ON/OFF/Color/Ltype/LWeight/MATerial/Plot/Freeze
/Thaw/LOck/Unlock/stAte/Description/rEconcile]: L (enter)
Enter loaded linetype name or [?] <Continuous>: Continuous (enter)
Enter name list of layer(s) for linetype "Continuous" <S-ANNO-TEXT>:
(enter)
Enter an option
[?/Make/Set/New/Rename/ON/OFF/Color/Ltype/LWeight/MATerial/Plot/Freeze
/Thaw/LOck/Unlock/stAte/Description/rEconcile]: (enter)
```

Notice how many places you have to press Enter more than once. Replacing all the “enters” with semicolons (be sure to count them all!) results in this:

```
^C^C-layer;M;S-ANNO-TEXT;C;1;;L;Continuous;;;
```

The next example uses a linetype that may or may not exist in the drawing already. (In the previous example, it’s a safe bet that “continuous” is always loaded.)

To get around this, the macro first inserts a drawing that contains nothing but that linetype definition. No objects, no styles—just that linetype. It inserts it exploded (note the asterisk), so that the block definition isn’t saved in the current drawing. Then the macro can continue with the layer settings.

```
^C^C-insert;*LT_beam_removed;0,0;1;;-layer;M;S-BEAM-STEL-
DEMO;C;1;;L;BEAM_REMOVED;;;
```

Both of these examples use the command line version of the Layer command with the Make option. I like Make here, instead of Set or New, because Make combines Set and New.

### Example 4: Plot a Drawing

This plotting macro assumes that you already have a page setup defined in the drawing.

We'll continue to follow the #1 rule of macro writing and go through the command line prompts first:

```
Command: -PLOT (enter)
Detailed plot configuration? [Yes/No] <No>: N(enter)
Enter a layout name or [?] <Layout1>: Layout1 (enter)
Enter a page setup name <>: Setup1 (enter)
Enter an output device name or [?] <HP Officejet 4300 series>: (enter)
Write the plot to a file [Yes/No] <N>: (enter)
Save changes to page setup [Yes/No]? <N>: (enter)
Proceed with plot [Yes/No] <Y>: (enter)
```

This would plot Layout1 to the HP Officejet using the page setup Setup1. Replacing all the “enters” with semicolons results in the following:

```
^C^C-plot;N;Layout1;Setup1;;;;
```

If you wanted to use the same setup with a different plotter, you would put the plotter name right after the semicolon for the Setup1 call.

Once you have the basic structure of the macro down, you can copy it and refine it to use different page setup names or different plotters.

### Repeating Macros

Because macros are basically series of commands, you can't repeat them by just pressing Enter or the space bar, or with a right-click (if right-click is set to act as Enter).

However, if your right-click is set to bring up shortcut menus, or if you have the time-sensitive right-clicks turned on (my favorite), then you can use them to repeat macros. It'll be the top option in the shortcut menu.

If you launched the macro from the ribbon, though, you can't repeat it without selecting the ribbon icon again. This is true for the 2009 and 2010 families of AutoCAD-based products.

### Bringing Macros Through Upgrades

Command options (and sometimes command names) don't always stay the same from release to release. For example, the variable PEDITACCEPT was introduced a few versions ago, which meant that one of the prompts in the PEDIT command could be eliminated. (“Selected object is not a polyline. Do you want to turn it into one?”) If you had an old macro that ran PEDIT, it might have an extra character or two that could cause some unexpected behavior.

Some commands also actually change behavior from release to release. For those, you might need to use the control character ^R (for “command versioning”) to launch the version of the command that will work in a macro. You’ll see below that the FILLET command is one that requires command versioning.

Even if you don’t anticipate any changes, it’s a good idea to review all your custom commands when you upgrade to a new release of AutoCAD or AutoCAD LT.

## Using DIESEL in Macros

Okay, I’ll admit it. DIESEL is a programming language. If you use it, you’re *technically* programming. But I don’t classify it with the other languages like LISP and VBA—everything’s in one line, there aren’t too many parentheses to worry about, and it still does basically what you would do in a regular command. That’s my story, and I’m sticking to it.

DIESEL stands for Direct Interpretively Evaluated String Expression Language, and it’s the only programming language available in AutoCAD LT. It’s completely built in to the program—you don’t have to create external files to use it, the way you do with LISP and VBA.

You can use DIESEL expressions to do basic math calculations, and to retrieve or set system variables. Using DIESEL means that you can do much more with a macro than simple command prompts might allow.

### DIESEL Macro components

DIESEL expressions that are used in CUI elements must always begin with the same command string:

\$M=

Dollar signs also always precede DIESEL expressions, and they’re always surrounded by parentheses:

*\$(diesel expression here)*

The next thing to know about DIESEL expressions is that their command sequence is backwards from regular calculators. In a standard calculator, you would enter 2+3 to get 5, or 12/3 to get 4. But in DIESEL, the operator comes first.

$\$(+, 2, 3)$

$\$(/, 12, 4)$

It helps to think about this verbally. “I want to add 2 and 3.” “I want to divide 12 by 4.” If you visualize the expression this way, you’ll always get the order right.

## Using Variables in DIESEL

You probably wouldn't use either of the above examples in an actual macro. You'd just do the math yourself first, and plug in the result where you needed it. But what if one of the numbers you needed was a variable—maybe the current DIMSCALE of your drawing?

This is where DIESEL really comes in handy. The GETVAR function can be used in some really powerful ways, although we'll start with a pretty simple example.

```
$(getvar , DIMSCALE)
```

This expression returns a number, specifically the current DIMSCALE of the drawing. Once you have it, you can use it in another expression.

```
$M=$( * , $(getvar , DIMSCALE) , 0.25)
```

Reading from left to right, this expression multiplies (with the asterisk) the DIMSCALE (the number from the GETVAR expression) by 0.25 (the last number in the expression).

So now we have a new number, one that's 1/4 of the current DIMSCALE. Finally, we can use it to set the radius for the FILLET command.

```
^C^C^Rfillet ; R ; $M=$( * , $(getvar , DIMSCALE) , 0.25) ;
```

Again reading from left to right, this macro cancels any running command (with ^R to launch the correct version of the fillet command), starts FILLET, enters "R" to set the radius, and lastly plugs in our DIESEL expression to set the value.

You could use DIMSCALE in an expression to set the insertion scale of a block, or linetype scales for plotting, or any number of things.

Here are two more macros for starting the TEXT and MTEXT commands with a specific text height (they assume that the style height is 0):

```
^C^C_dtext ; \ $M=$( * , $(getvar , DIMSCALE) , 0.125) ;  
^C^C_mtext ; \ H ; $M=$( * , $(getvar , DIMSCALE) , 0.125) ; \
```

And of course, DIMSCALE isn't the only variable you can use—anything that you can set from the command line, you can call with GETVAR.

## Conditional Expressions

DIESEL also supports an IF function, which is handy for toggling between two variable settings, or for performing an operation only if another value meets certain criteria. IF functions look like this:

```
$(if , expression , true , false)
```

It's a lot like the similar function in Excel. First, you give it an expression to evaluate. If that comes back as "true", IF will run the "true" part of the expression. Otherwise, it'll run "false."

Here's a macro for switching between cursor sizes of 5% and 100%.

```
^C^C$M=$(if,$(=,$(getvar,cursorsize),100),cursorsize;5;;cursorsize;100;;)
```

In this case, the expression is:

```
$(=,$(getvar,cursorsize),100)
```

In other words, is the cursor size already set to 100? If it is, then it runs the "true" part:

```
cursorsize;5;;
```

And if the cursor size isn't 100, it runs the "false" part:

```
cursorsize;100;;
```

So the value of CURSORSIZE is switched back and forth each time the macro is run.

## USER Variables

DIESEL doesn't have the same flexibility with variable names as LISP, but there are several preset-values you can use.

There are ten variables that can be used to store numbers: five USERI (for integers) and five USERR (for real numbers, or numbers with decimals). The next macro uses them in a pretty neat way—to automatically increment the value of a block attribute each time it's inserted.

```
*^C^C_-
insert;label;\;;$M=$(getvar,USERI1);USERI1;$M=$(+,$(getvar,USERI1),1);
```

The first half of this macro (up to the first \$) just inserts the block the way any other macro would. But after that, it calls up the USERI1 variable, and puts that in as the attribute value. When it's done, it uses the addition function to add 1 to USER1, incrementing it for the next insertion.

This macro will work best if you only have one attribute to fill in. You also need to reset USERI1 to your desired starting value before each use. You could do this resetting with a macro, or simply type it in at the command line.

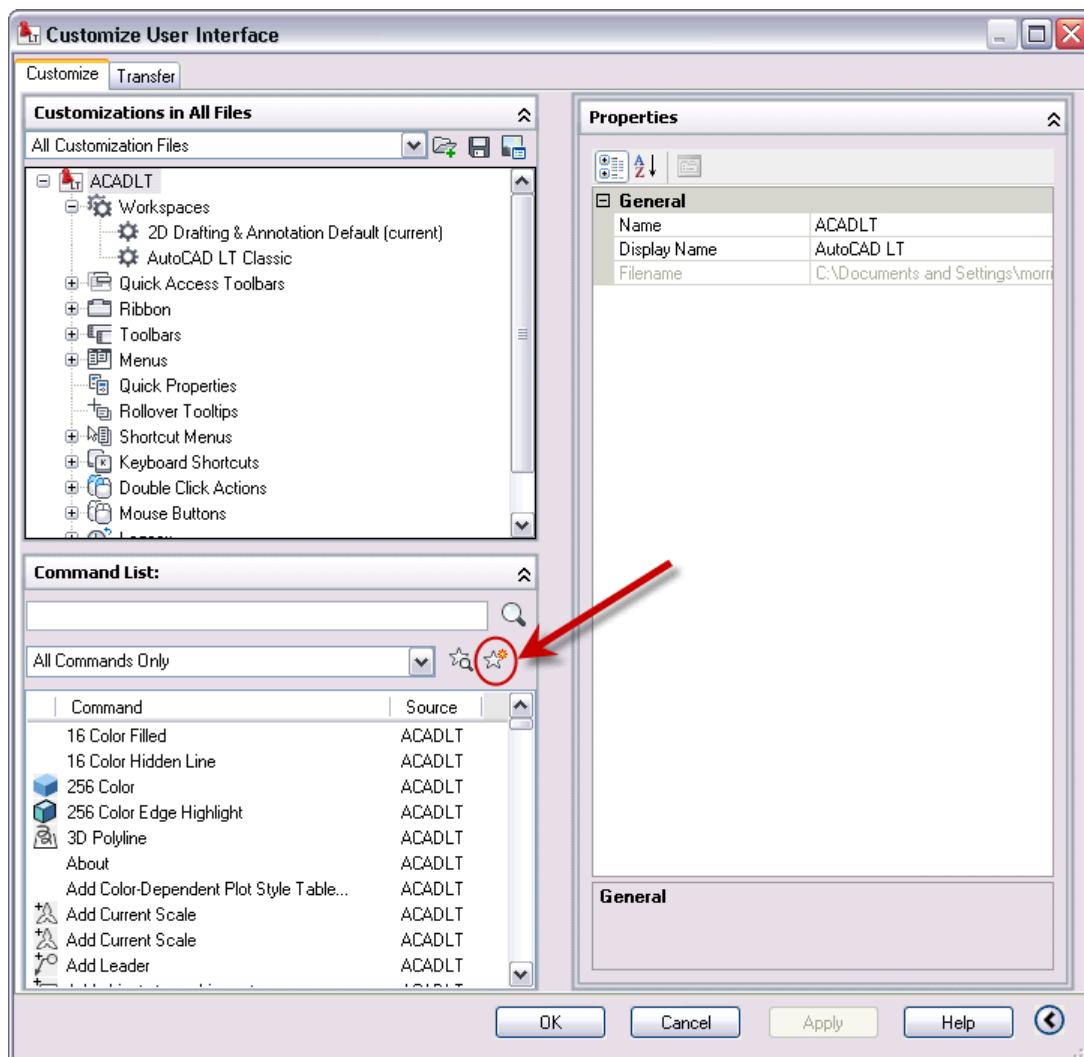
## Using Macros in CUI Elements

Now that you have a macro, you need a place to put it. Macros can be added to a number of interface elements using the Customize User Interface (CUI) Editor.

The CUI is a big topic, but the next few pages will give you a crash course in what you need to know to begin experimenting with your own macros.

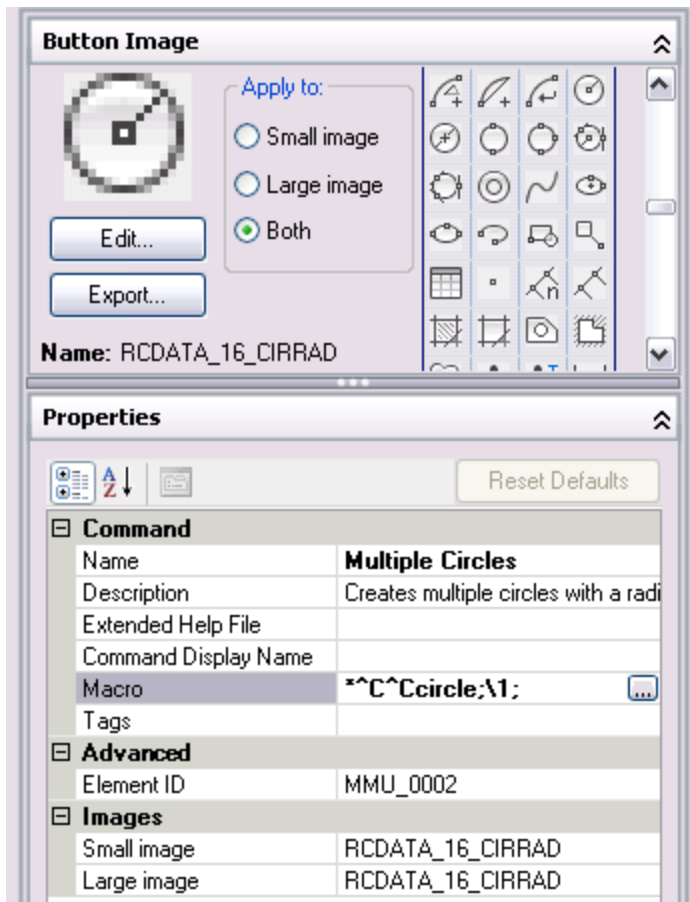
### Creating a New Macro-Based Command

Macros are custom commands, so you'll need to use the "New Command" button in the Command List pane of the CUI Editor.





In the new command pane that opens, you can fill in all the information about your command.



In the Name Field, give your command a descriptive name, something that will help you remember what it does.

The Description Field is what will show up in the tooltip.

The Extended Help File can point to a separate file containing information that would show up in an extended tooltip (the ones that show up after you've hovered over a button for a while).

The Command Display name shows the command-line version of the command. It isn't necessarily helpful for macros, since you can't call them from the command line.

The Macro field is where you put your custom command syntax. The small button at the end of the line opens a bigger window that can be helpful when entering long macros.

Tags are keywords that can be used as search terms in the Menu Browser.

The Element ID is a unique tag assigned to each command. AutoCAD will automatically generate one, and if you use it you're guaranteed that it won't overlap anything else.

Finally, in the Images section, you can either use one of the included icons or point the macro to your own BMP files.

The critical fields here for you to fill in are Name, Macro, and Images. The rest are optional, but may be helpful to other people using your macro.

## Using Your New Command

The last step in using this new command is to put it on an interface element so you can access it. Within the CUI Editor, your choices for elements are ribbon panels, toolbars, and menus. Creating any one of these makes use of the two main editing functions in the CUI: right-clicking and drag-and-dropping.

## Toolbars

To put your new command on a toolbar, you have a couple of choices. You can either drag-and-drop it straight onto an existing toolbar, or you can create a new toolbar and drag-and-drop the command there.

Either way, the process is very straightforward. First, locate the Toolbar node in one of your loaded CUI files, and expand it to see your available toolbars. If you want a new toolbar, just right-click on the Toolbar node and select “New Toolbar”.

Once you have the toolbar you want, simply drag-and-drop your command from the Command List pane onto the toolbar.

## Menus

The procedure for putting a command on a menu is very similar to the one for toolbars, except that menus can have multiple levels.

Again, if the menu you want to add the command to already exists, simply drag-and-drop the command onto it.

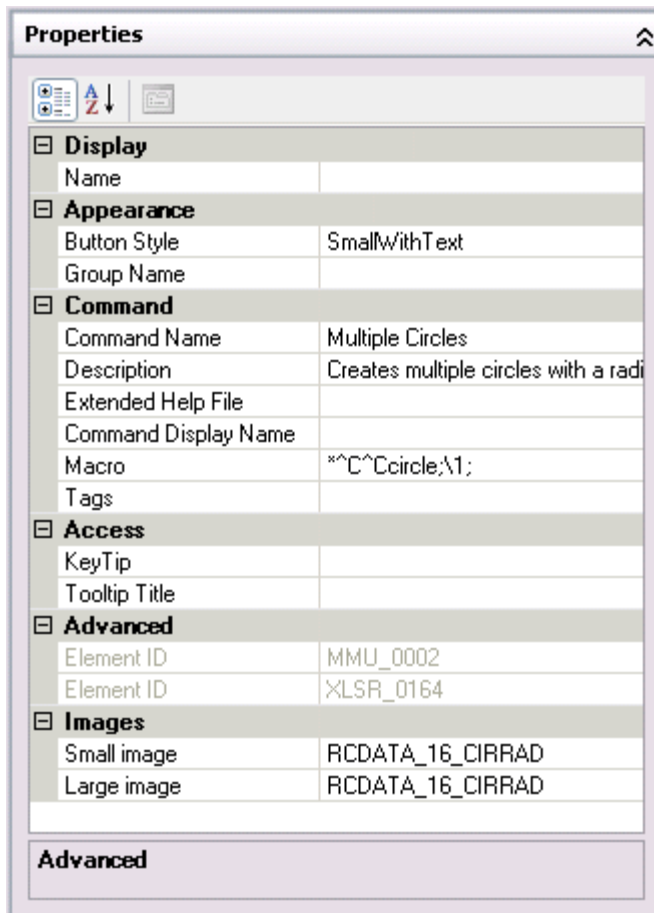
If you want a new menu, right-click on either the Menu node or on an existing menu. From there, select New Menu or New Sub-menu. “New Menu” adds a new menu at the same level as the one on which you clicked, while “New Sub-menu” creates a new menu one level in from the original menu. Then you can drag-and-drop your command into the new menu.

## Ribbon Panels

Ribbon panels are a little more complicated than menus and toolbars, because they have more icon display options.

Just like menus and toolbars, you can drag-and-drop commands directly onto an existing Ribbon panel, or create new panels by right-clicking on nodes in the CUI editor. Keep in mind that if you create a new Ribbon panel, you also have to create a new Ribbon tab for it (or drag it onto an existing tab), or it won't show up.

This is a screenshot of the Properties pane of a Ribbon button for the same command we created a page earlier. You can see that although most of the fields are the same, there are three new sections—Display, Appearance, and Access—with more options.



Let's start at the top.

“Display Name” is the text that is displayed on the ribbon, if you want text. If it's left blank, the Command Name is used as the display name.

Under “Appearance”, you have four choices for the button style:

- LargeWithText (Vertical)
- LargeWithText (Horizontal)
- SmallWithText
- SmallWithoutText

These control the size of the icon and the position of the text, if any.


“Grouping” only applies to flyout commands.

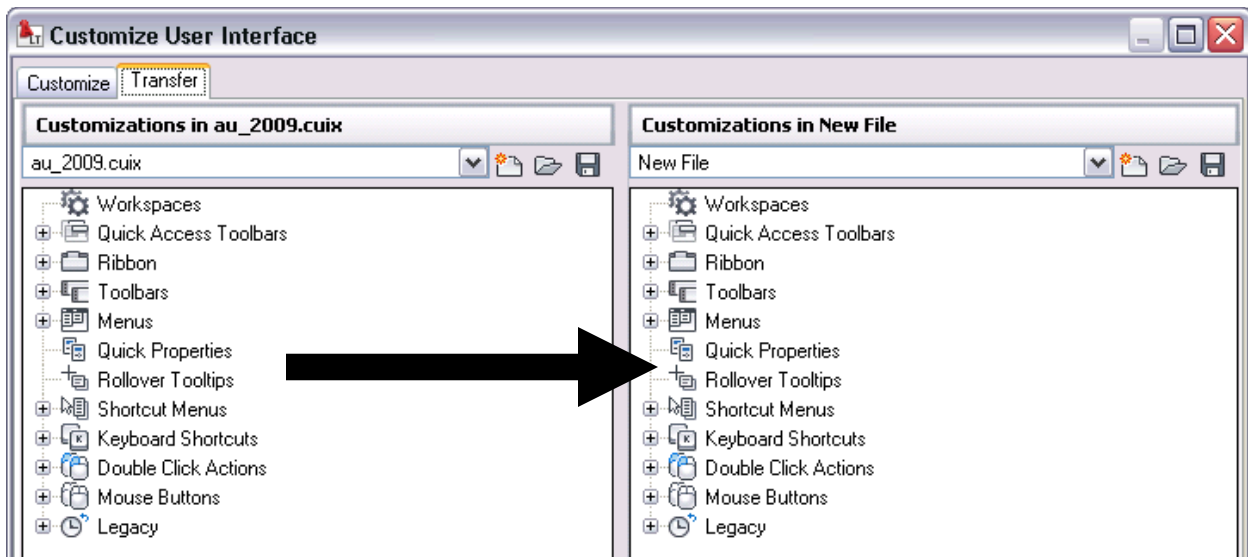
Finally, “KeyTip” is the hotkey combination that can be used to launch the macro. They're used in combination with the Alt key to activate ribbon tabs and commands.

Yes, that means it's now possible to run macros from the keyboard.

## Sharing CUI Macros

The easiest way to share the macros you create is by transferring them to a new CUI file, using the Transfer tab in the CUI editor. Simply drag and drop your Ribbon panels/tabs, menus, and toolbars from the left side to the right, and save the new CUI file. Then you can either give it to people to save on their own computer, or put it in a networked location.

To load this new partial CUI file, use the “open” icon in the Customize tab of the editor.  After it's loaded, you can modify your workspace to turn on the new tools.

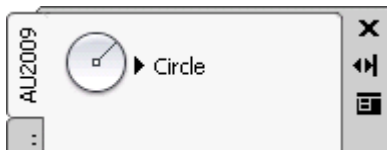


## Using Macros on Tool Palettes

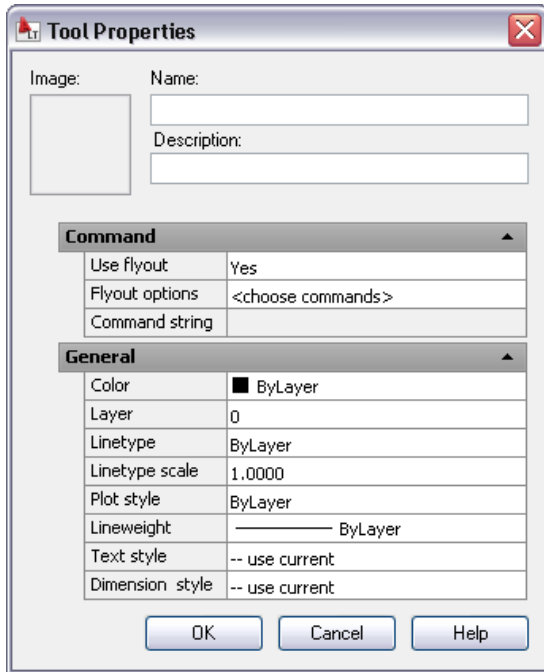
You probably already know that you can add blocks and hatches to Tool Palettes. You may even know that you can add commands to Tool Palettes. But did you know you can add macros to those command tools?

### Command Tools

Let's say you want to add that 1-unit-radius circle to a tool palette. So you create a circle and drag-and-drop it onto a new or existing palette.



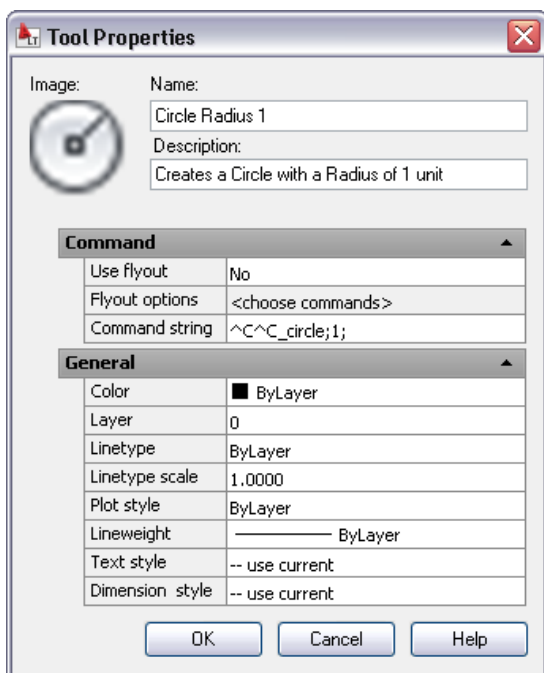
To make sure that all of your settings are correct, you can right-click on the circle and select Properties. Here's what you see:



Notice that the Name and Description, as well as the Image, are blank. That's because it's using the default command name and description. You can also see that "Use Flyout" is set to "Yes", and that the "Command String" is gray—you can't type there.

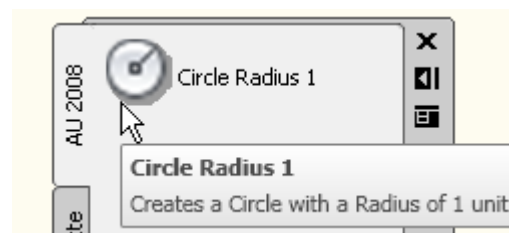
But in this case, you don't need all the flyout options—you just want to use the macro to create a circle. So change the flyout to "No", and enter your macro in the "Command String" box. You could set the General properties here as well, if you wanted to create the circle on a particular layer, or with a particular color, linetype, etc.

When you move away from the default options, this is also your chance to enter your own name for the command, and a description that will show up as a tooltip.



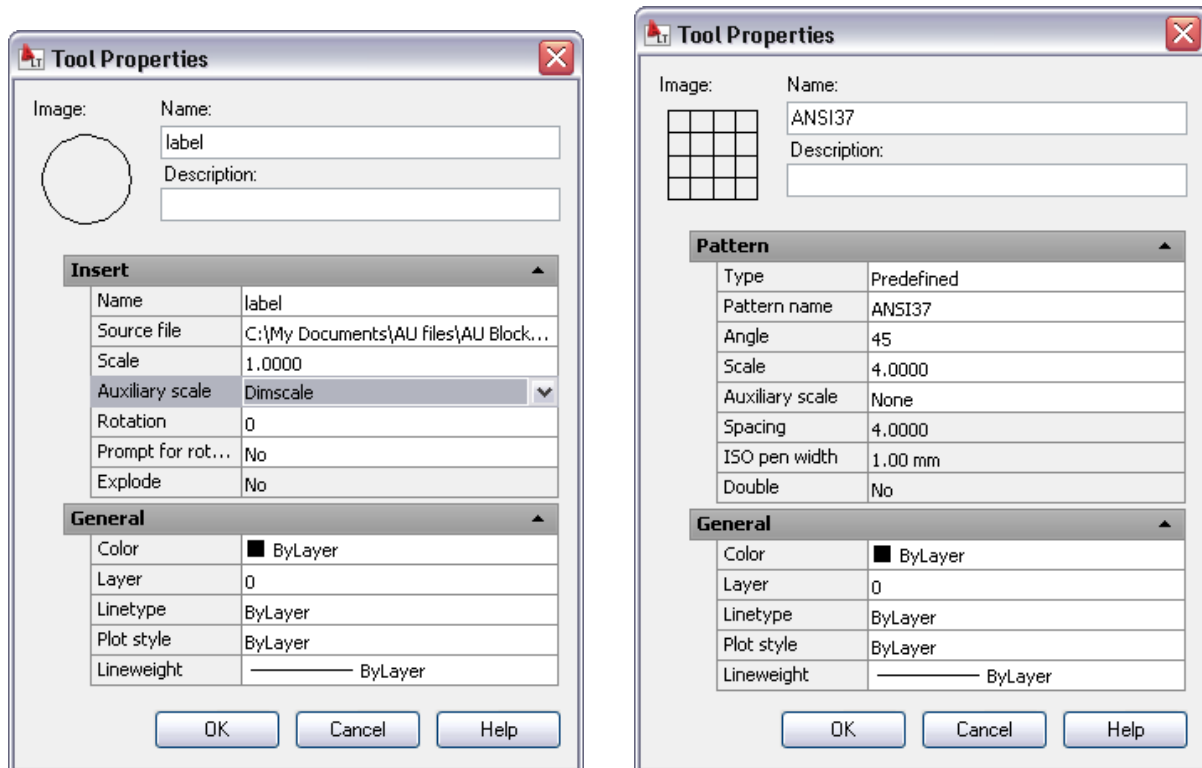
The image in this screenshot was assigned automatically—if you'd rather use a different one, right-click and select "Specify Image."

Below is the final tool:



## Block & Hatch Tools

When you drag and drop blocks or hatches onto Tool Palettes, they automatically inherit the properties of the source object. If that's what you want, then you're done. If not, you can right-click the new tool and select Properties, just like with command tools. The dialog boxes will look a little different, but the effects are the same.



Any property that could be set in a command-line macro can be set through the tool's properties. It's an easy way to make sure the same settings are used every time.

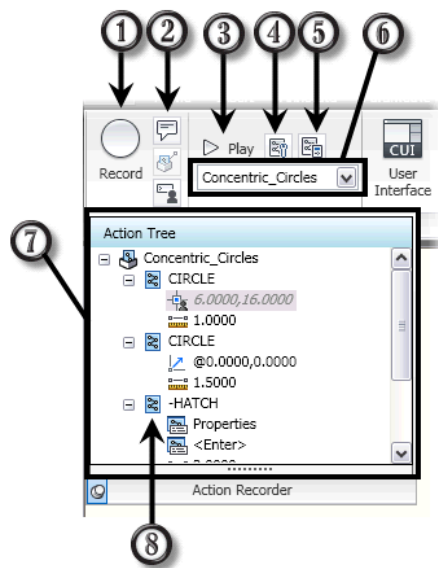
You can even mimic the effect of `$(getvar,DIMSCALE)` in a tool with the Auxiliary Scale property. (See the block properties image above.) If you set that to Dimscale, each time you insert the block or use the hatch it will read the current DIMSCALE, multiply that by the Scale property, and scale itself accordingly. You'll never have to do the math yourself again.

## Creating Macros with the Action Recorder

The Action Recorder is the one part of this handout that doesn't apply to LT. (Sorry.) But if you're running full AutoCAD, it can be a handy way to create and store your macros without having to type them out by hand.

### Introduction

The Action Recorder is a way to record mouse clicks and command entry automatically. Then you can play them back as needed to repeat the performed steps with minimal effort. Action macros are created with an ACTM file extension, and those files can be easily shared between users.



The following options are available on the Action Recorder panel:

1. **Record.** Starts recording the series of actions that will create an action macro.
2. **Insert a Message, Insert a Base Point, Pause for User Input.** A message is text you want to show the user during the action macro playback. Base Points provide a reference location for following relative coordinates in the action macro. Pausing for User Input replaces the stored input with a place where the user can enter information.
3. **Play.** Plays back the selected action macro.
4. **Preferences.** Enables the Action Recorder Preferences dialog.
5. **Manage Action Macros.** Use the Action Macro Manager to copy, rename, modify, or delete action macro files.
6. **Action Macro list.** Displays a list of all available action macros.
7. **Action Tree.** The series of actions that make up the macro.
8. **Action Item.** One command or input in the Action Tree.

## Creating Action Macros

You use the Record button in the Action Recorder to begin creating an action macro. While it is recording you can launch commands, enter values and select objects using familiar AutoCAD functionality. The Action Recorder can record typical actions from the command line as well as from toolbars, ribbon panels, pulldown menus, the Properties palette, the Layer Properties Manager and Tool Palettes. However, most dialog boxes are not recorded in the Action Recorder unless you use the command line equivalent of the dialog box setting. (See? Learning how to write macros by hand is still useful.) When you're finished recording, click Stop on the Action Recorder panel.

Remember that this is not a timed exercise! Even if you sit there for five minutes while you decide what your next step should be, it won't affect how fast the macro will play back. It'll play back as fast as it can—as fast as an ordinary macro would. So don't let the bright red Record button make you nervous.

## Action Macro File Locations

Each action macro you record is saved as an individual file and is recorded by the local machine at the Actions Recording File Location with an ACTM extension. The file location is specified in the Options dialog box, Files tab, under Action Recorder Settings.

## Shared Action Macros

To allow others to use a macro you create, you must copy the macro file onto each computer where you want to use it. Copy it to the location indicated by the Actions Recording File Location option.

You can also put action macros on your network, and point the support paths in AutoCAD to the appropriate location.

## Guidelines for Using Action Macros

Here are a few recommended practices to enable you to use action macros efficiently. You can use these guidelines to create, edit and play action macros.

- While recording a macro, only launch commands from ribbon panels, tool palettes, menus or toolbars. Do not launch commands that bring up a dialog box.
- If you want to record the settings changed in a dialog box, use the command line equivalent of the command and specify the settings at the command line. This will ensure that the macro records the steps.
- When you use the command line equivalents for dialog box settings in an action macro, don't accept command line defaults. Always type in a value.
- If you don't want an action macro to be changed, you should use Windows Explorer to place a read-only attribute on the file or on the directory it is shared in.



## Additional Resources

If you need more information, check out these websites:

- The AutoCAD home page: [www.autodesk.com/autocad](http://www.autodesk.com/autocad)
- The AutoCAD LT home page: [www.autodesk.com/autocadlt](http://www.autodesk.com/autocadlt)
- The Autodesk Discussion Groups: [www.autodesk.com/discussion](http://www.autodesk.com/discussion)
- LT Unlimited – the official LT blog: <http://ltunlimited.typepad.com>
- AUGI: [www.augi.com](http://www.augi.com)

## Appendix A: Summary of Handout Macros

Draw a circle with a radius of 1

```
^^C_circle;1;
```

Draw multiple circles with a radius of 1

```
*^^C_circle;1;
```

Rotate by a fixed angle

```
^^C_rotate;\;\180;
```

Copy then Rotate

```
^^C_copy;\;\_rotate;L;@;\
```

Arc with Donut

```
^^C_arc;\;\donut;0;0.5;@;
```

Create a hatch with pattern ANSI37, scale of 4, angle of 45, applied to a selected object

```
^^C-hatch;P;ANSI37;4;45;S;\;
```

Insert a block with a scale of 12 and a rotation of 90

```
^^C-insert;sample;S;12;R;90;\
```

Insert an exploded block with a scale of 12 and a rotation of 90

```
^^C-insert;*sample;\12;90;
```

Insert a block with a scale of 12 and a rotation of 90, then explode it

```
^^C-insert;sample;S;12;R;90;\explode;L;
```

Insert a block with a scale of 12 and a rotation of 90, then explode and purge it

```
^^C-insert;sample;S;12;R;90;\explode;L;-purge;B;sample;N;
```

Create a red layer with a continuous linetype

```
^^C-layer;M;S-ANNO-TEXT;C;1;L;Continuous;;;
```

Load a linetype, then create a red layer that uses that linetype

```
^^C-insert;*LT_beam_removed;0,0;1;;-layer;M;S-BEAM-STEL-
DEMO;C;1;;L;BEAM_REMOVED;;;
```

Plot a drawing

```
^^C-plot;N;Layout1;Setup1;;;;
```

Create a fillet with a radius equal to 1/4 times the DIMSCALE

```
^^C^Rfillet;R;$M=$( *,$(getvar,DIMSCALE),0.25);
```

Create text with a height equal to 1/8 times the DIMSCALE

```
^^C_dtext;\$M=$( *,$(getvar,DIMSCALE),0.125);
```

Create mtext with a height equal to 1/8 times the DIMSCALE

```
^^C_mtext;\H;$M=$( *,$(getvar,DIMSCALE),0.125);\
```

Toggle CURSOR\_SIZE between 5% and 100%

```
^^C$M=$(if,$(=,$(getvar,cursor_size),100),cursor_size;5;;,cursor_size;100;;)
```

Insert a block with a single attribute, and increase the attribute value by 1 every time

```
*^^C^_
insert;label;\;;$M=$(getvar,USERI1);USERI1;$M=$(+,$(getvar,USERI1),1);
```

## Appendix B: User Macros (Untested)

I haven't been able to test most of these, but the people who posted them on the discussion groups swear by them. If you have any questions about them, that's the place to look.

(<http://discussion.autodesk.com>)

General clean-up before closing

```
^^C^P-layout;set;Model;-layer;s;0;;zoom;e;.-purge;a;*;no;-
layout;set;;zoom;e;'spell;_qsave
```

Used for joining lines into polylines prior to sending to a CNC machine

```
^^C_SELECT;SI;\^^C_EXPLODE;@;^^C_PEDIT;@;Y;J;ALL;;^^C^C
```

Redefine a block

```
^^C-insert;sample=sample;\;;
```

Lock and hide viewports

```
^^C-layer;thaw;viewport;;MVIEW;L;ON;ALL;;-LAYER;S;0;OFF;VPOR;;
```

Unlock and show viewports

```
^^C-layer;thaw;viewport;;MVIEW;L;OFF;ALL;;-LAYER;ON;VPOR;;
```

Toggle between rectangular and isometric snaps (from Kevin Chandler, Lane Associates)

This macro demonstrates the automatic checking/unchecking of a pulldown menu:

For the menu's Name field:

```
$(if,$(and,$(getvar,SNAPSTYL),1),!.)Isometric Cursor
```

For the macro field:

```
$M=$(if,$(and,$(getvar,SNAPSTYL),1),^C^C_SNAPSTYL;0,^C^C_SNAPSTYL;1)
```

Toggle between Layouts and Modelspace (from Erik Deyo, <http://itsideofthings.blogspot.com>)

```
^C^Csetvar;tilemode;$M=$(!,$(getvar,tilemode),1)
```

Sets the new current layer to DIM, creates a linear dimension, and restores the original layer to current:

```
^C^C_setenv;CURRENTLAYER;$M=$(getvar,CLAYER);CLAYER;Dim;_dimlinear;\\\
CLAYER;$M="$(getenv,CURRENTLAYER)";
```

Labels a polyline with its area:

```
^C^Carea;e;\_-
mtext;$M=$(getvar,viewctr);@;AREA=$m=$(rtos,$(/,$(getvar,area),1000000
),2,3)m2;i_move;_L;i@;\
```