

AutoCAD®

autodesk®

Customization Guide

2005

February 2004

This publication, or parts thereof, may not be reproduced in any form, by any method, for any purpose.

AUTODESK, INC., MAKES NO WARRANTY, EITHER EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO ANY IMPLIED WARRANTIES OF MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE REGARDING THESE MATERIALS, AND MAKES SUCH MATERIALS AVAILABLE SOLELY ON AN "AS-IS" BASIS.

IN NO EVENT SHALL AUTODESK, INC., BE LIABLE TO ANYONE FOR SPECIAL, COLLATERAL, INCIDENTAL, OR CONSEQUENTIAL DAMAGES IN CONNECTION WITH OR ARISING OUT OF PURCHASE OR USE OF THESE MATERIALS. THE SOLE AND EXCLUSIVE LIABILITY TO AUTODESK, INC., REGARDLESS OF THE FORM OF ACTION, SHALL NOT EXCEED THE PURCHASE PRICE OF THE MATERIALS DESCRIBED HEREIN.

Autodesk, Inc., reserves the right to revise and improve its products as it sees fit. This publication describes the state of this product at the time of its publication, and may not reflect the product at all times in the future.

Autodesk Trademarks

The following are registered trademarks of Autodesk, Inc., in the USA and/or other countries: 3D Props, 3D Studio, 3D Studio MAX, 3D Studio VIZ, 3DSurfer, 3ds max, ActiveShapes, ActiveShapes (logo), Actrix, ADI, AEC Authority (logo), AEC-X, Animator Pro, Animator Studio, ATC, AUGI, AutoCAD, AutoCAD LT, AutoCAD Map, Autodesk, Autodesk Inventor, Autodesk (logo), Autodesk MapGuide, Autodesk Streamline, Autodesk University (logo), Autodesk View, Autodesk WalkThrough, Autodesk World, AutoLISP, AutoSketch, backdraft, Biped, bringing information down to earth, Buzzsaw, CAD Overlay, Character Studio, Cinepak, Cinepak (logo), cleaner, Codec Central, combustion, Design Your World, Design Your World (logo), EditDV, Education by Design, gmax, Heidi, HOOPS, Hyperwire, i-drop, Inside Track, IntroDV, Kinetix, MaterialSpec, Mechanical Desktop, NAAUG, ObjectARX, PeopleTracker, Physique, Planix, Powered with Autodesk Technology (logo), ProjectPoint, RadioRay, Reactor, Revit, Softdesk, Texture Universe, The AEC Authority, The Auto Architect, VISION*, Visual, Visual Construction, Visual Drainage, Visual Hydro, Visual Landscape, Visual Roads, Visual Survey, Visual Toolbox, Visual Tugboat, Visual LISP, Volo, *WHIP!*, and *WHIP!* (logo).

The following are trademarks of Autodesk, Inc., in the USA and/or other countries: AutoCAD Learning Assistance, AutoCAD LT Learning Assistance, AutoCAD Simulator, AutoCAD SQL Extension, AutoCAD SQL Interface, Autodesk Envision, Autodesk Map, AutoSnap, AutoTrack, Built with ObjectARX (logo), burn, Buzzsaw.com, CAICE, Cinestream, Civil 3D, cleaner central, ClearScale, Colour Warper, Content Explorer, Dancing Baby (image), DesignCenter, Design Doctor, Designer's Toolkit, DesignProf, DesignServer, Design Web Format, DWF, DWFit, DWFwriter, DWG Linking, DXF, Extending the Design Team, GDx Driver, gmax (logo), gmax ready (logo), Heads-up Design, jobnet, lustre, ObjectDBX, onscreen onair online, Plans & Specs, Plasma, PolarSnap, Real-time Roto, Render Queue, Visual Bridge, Visual Syllabus, and Where Design Connects.

Autodesk Canada Inc. Trademarks

The following are registered trademarks of Autodesk Canada Inc. in the USA and/or Canada, and/or other countries: discreet, fire, flame, flint, flint RT, frost, glass, inferno, MountStone, riot, river, smoke, sparks, stone, stream, vapour, wire.

The following are trademarks of Autodesk Canada Inc., in the USA, Canada, and/or other countries: backburner, Multi-Master Editing.

Third Party Trademarks

All other brand names, product names or trademarks belong to their respective holders.

Third Party Software Program Credits

ACIS Copyright © 1989-2001 Spatial Corp. Portions Copyright © 2002 Autodesk, Inc.

AnswerWorks ® Copyright © 1997-2003 WexTech Systems, Inc. Portions of this software © Vantage-Knexys. All rights reserved.

Copyright © 1997 Microsoft Corporation. All rights reserved.

International CorrectSpell™ Spelling Correction System © 1995 by Lernout & Hauspie Speech Products, N.V. All rights reserved.

InstallShield™ 3.0. Copyright © 1997 InstallShield Software Corporation. All rights reserved.

Macromedia Flash™ Player Copyright © 1995-2003 Macromedia, Inc. All rights reserved. Macromedia and Flash are trademarks of Macromedia, Inc.

PANTONE ® Colors displayed in the software application or in the user documentation may not match PANTONE-identified standards. Consult current PANTONE Color Publications for accurate color.

PANTONE ® and other Pantone, Inc. trademarks are the property of Pantone, Inc. © Pantone, Inc., 2002

Pantone, Inc. is the copyright owner of color data and/or software which are licensed to Autodesk, Inc., to distribute for use only in combination with certain Autodesk software products. PANTONE Color Data and/or Software shall not be copied onto another disk or into memory unless as part of the execution of this Autodesk software product.

Portions Copyright © 1991-1996 Arthur D. Applegate. All rights reserved.

Portions of this software are based on the work of the Independent JPEG Group.

RAL DESIGN © RAL, Sankt Augustin, 2002

RAL CLASSIC © RAL, Sankt Augustin, 2002

Representation of the RAL Colors is done with the approval of RAL Deutsches Institut für Gütesicherung und Kennzeichnung e.V. (RAL German Institute for Quality Assurance and Certification, re. Assoc.), D-53757 Sankt Augustin."

Copyright © Stade de France - Macary, Zublena et Regembal, Costantini - Architectes, ADACP - Paris - 2003

Typefaces from the Bitstream ® typeface library copyright 1992.

Typefaces from Payne Loving Trust © 1996. All rights reserved.

GOVERNMENT USE

Use, duplication, or disclosure by the U.S. Government is subject to restrictions as set forth in FAR 12.212 (Commercial Computer Software-Restricted Rights) and DFAR 227.7202 (Rights in Technical Data and Computer Software), as applicable.

Contents

Chapter 1	Basic Customization	1
	Overview of Customization	2
	Organize Program and Support Files.	4
	Overview of File Organization.	4
	Multiple Configurations	6
	Multiple Drawing Folders	7
	Create Custom Toolbars	8
	Create, Rename, and Delete Toolbars	8
	Modify the Appearance of a Toolbar.	10
	Create and Edit Toolbar Buttons	12
	Add Shortcut Keys to Menus and Toolbars.	16
	Customize a Publish to Web Template	19
	Define Custom Commands	21
	Define External Commands	21
	Create Command Aliases	25
Chapter 2	Custom Linetypes	27
	Overview of Linetype Definitions	28
	Simple Custom Linetypes	28
	Text in Custom Linetypes	31
	Shapes in Custom Linetypes	34

Chapter 3	Custom Hatch Patterns.	37
	Overview of Hatch Pattern Definitions	38
	Hatch Patterns with Dashed Lines	40
	Hatch Patterns with Multiple Lines	42
Chapter 4	The Menu File.	45
	Overview of Menu Files.	46
	Load and Unload Menu Files	50
	Create Menu Macros	54
	Overview of Menu Macros	54
	Pause for User Input in Menu Macros	56
	Provide International Language Support in Menu Macros	58
	Use Special Control Characters in Menu Macros	58
	Repeat Commands in Menu Macros	61
	Use Single Object Selection Mode in Menu Macros	61
	Use Menu Macros to Swap Menus	62
	Use Conditional Expressions in Menu Macros.	63
	Use AutoLISP in Menu Macros	64
	Customize Buttons on a Pointing Device	65
	Overview of Buttons and Aux Menus	65
	Swap Buttons and Aux Menus.	67
	Accept Coordinate Entry in Button Menus.	69
	Create Pull-Down and Shortcut Menus	70
	Overview of Pull-Down and Shortcut Menus	70
	Create Pull-Down Menus	72
	Create Shortcut Menus	74
	Control Display of Menu Item Labels	75
	Reference Pull-Down or Shortcut Menus	77
	Swap and Insert Pull-Down Menus	80
	Customize Toolbars	82
	Create Toolbars	82
	Create Toolbar Buttons	84
	Create Toolbar Flyouts	85
	Define Controls for a Toolbar	86
	Specify User-Defined Bitmaps	87
	Create Image Tile Menus	87
	Overview of Image Tile Menus	87
	Prepare Slides for Image Tile Menus	90
	Create Screen Menus	91
	Create Tablet Menus.	96
	Create Status Line Help Messages	97
	Create Shortcut Keys	98

Chapter 5	DIESEL.	103
	Customize the Status Line	104
	Overview of the MODEMACRO System Variable	104
	Set MODEMACRO Values.	104
	Set MODEMACRO with AutoLISP	106
	DIESEL Expressions in Menu Macros	108
	Catalog of DIESEL Functions.	111
	+ (addition)	111
	– (subtraction)	111
	* (multiplication)	111
	/ (division)	111
	= (equal to)	111
	< (less than)	111
	> (greater than)	112
	!= (not equal to)	112
	<= (less than or equal to)	112
	>= (greater than or equal to)	112
	and	112
	angtos	112
	edtime	113
	eq	114
	eval.	115
	fix	115
	getenv	115
	getvar	115
	if	115
	index	115
	nth	116
	or	116
	rtos	116
	strlen	116
	substr	116
	upper	117
	xor	117
	DIESEL Error Messages.	117
Chapter 6	Slides and Command Scripts.	119
	Create Slides	120
	Overview of Slides	120
	View Slides	121
	Create and View Slide Libraries	121
	Create Command Scripts	123
	Overview of Command Scripts	123
	Run Scripts at Startup	125

Run Slide Shows from Scripts	126
Chapter 7 Introduction to Programming Interfaces	129
ActiveX Automation.	130
Overview of ActiveX	130
Define a Command to Start Your Application	131
Start an Application from a Menu	131
AutoCAD VBA	133
Overview of AutoCAD VBA	133
Use AutoCAD VBA Applications	134
Automatically Load and Execute VBA Projects	135
AutoLISP and Visual LISP	136
Overview of AutoLISP and Visual LISP	136
Use AutoLISP Applications	138
Automatically Load and Run AutoLISP Routines	139
Overview of AutoLISP Automatic Loading	139
The <i>acad.lsp</i> File	140
The <i>acaddoc.lsp</i> File	141
The MNL File for an AutoLISP Menu	142
Prevent AutoLISP Errors When Loading Startup Files.	143
S::STARTUP Function: Postinitialization Execution	143
Object ARX	145
Overview of ObjectARX.	145
Use ObjectARX Applications	145
Automatically Load ObjectARX Applications	146
Appendix A ASCII Codes	149
Appendix B Shapes and Shape Fonts	153
Overview of Shape Files.	154
Create Shape Definition Files	155
Shape Descriptions	155
Vector Length and Direction Code	157
Special Codes	158
Codes 0, 1, and 2: End of Shape and Draw Mode Control	159
Codes 3 and 4: Size Control	159
Codes 5 and 6: Location Save/Restore	160
Code 7: Subshape	160
Codes 8 and 9: X-Y Displacements	160
Code 00A: Octant Arc	161
Code 00B: Fractional Arc	162
Codes 00C and 00D: Bulge-Specified Arcs	163
Code 00E: Flag Vertical Text Command	164

Text Font Descriptions.	165
Big Font Descriptions	166
Define a Big Font	166
Define an Extended Big Font File	167
Use Big Font Text in a Drawing	173
Use a Big Font to Extend a Font	174
Unicode Font Descriptions	175
Superscripts and Subscripts in SHX Files	176
Index	179

Basic Customization

AutoCAD is a general-purpose drafting system designed with an open architecture so you can customize and extend its many features. As a result, you can expand and shape AutoCAD according to your needs.

Your dealer can offer you independently developed applications that can further tailor AutoCAD to your needs.

In this chapter

- Overview of Customization
- Organize Program and Support Files
- Create Custom Toolbars
- Add Shortcut Keys to Menus and Toolbars
- Customize a Publish to Web Template
- Define Custom Commands

Overview of Customization

AutoCAD® can be customized in simple ways, for example, changing the directory structure or moving a button from one toolbar to another. If you want to change the interface further, you can edit the MNU file and use DIESEL code to create menus with your own commands.

You can also use a number of powerful application programming interfaces (APIs) to add to and modify AutoCAD to suit your needs.

The list that follows is arranged roughly in order of complexity:

- **Organize files.** You can organize program, support, and drawing files. For example, you can make a separate folder for each project that includes only the support files that project needs.
- **Modify the user interface.** The CUSTOMIZE command displays dialog boxes in which you can create, modify, or remove toolbars and toolbar buttons and assign shortcut keys to commands.
- **Create custom templates.** Use templates to define common parameters when you publish a drawing using the Publish to Web wizard.
- **Run external programs and utilities from within AutoCAD.** You can, for example, copy a disk or delete a file from within AutoCAD by adding the appropriate external command to the program parameters (PGP) file, *acad.pgp*.
- **Define command aliases.** You can define simple abbreviations, or aliases, for frequently used commands. For example, you might want to start the BHATCH command by entering **b**.
- **Create custom linetypes, hatch patterns, shapes, and text fonts.** You can create linetypes, hatch patterns, shapes, and text fonts that conform to your company standards and working methods.
- **Edit the menu file.** The menu (MNU) file is an ASCII file that controls many aspects of the user interface, including the behavior of your pointing device buttons and the functionality and appearance of pull-down, tablet, and image tile menus, toolbars, and accelerator keys. You can edit the MNU file or create a new one to add commands or combine commands and assign them to a menu, toolbar, or other location.
- **Customize the status line.** You can use the DIESEL string expression language and the MODEMACRO system variable to provide additional information at the status line, such as the date and time, system variable settings, or retrievable information using AutoLISP®.

- **Automate repetitive tasks by writing scripts.** A script is an ASCII text file containing commands that are processed like a batch file when you run the script. For example, if a set of drawings needs to be plotted a certain way, you can write a script that opens each drawing, hides and displays various layers, and issues PLOT commands. You can use scripts with slides to create automated presentations like those used at trade shows. A slide is a “snapshot” of the drawing area that cannot be edited. Slides can also be used in image tile menus and dialog boxes.
- **Redefine or disable selected AutoCAD commands, either at the Command prompt or as part of an AutoLISP or ObjectARX® program.** You can redefine certain AutoCAD commands to issue supplementary messages and instructions or, for example, to create a drawing management system in which the QUIT command is redefined to write billing information to a log file before ending the editing session.

In addition to the methods described in the *Customization Guide*, there are application programming interfaces (APIs) available for customizing AutoCAD. “Introduction to Programming Interfaces” on page 129 briefly describes these APIs and provides cross-references to more information.

See Also

“Organize Program and Support Files”
“Create Custom Toolbars”
“Add Shortcut Keys to Menus and Toolbars”
“Customize a Publish to Web Template”
“Create Command Aliases”
“Custom Linetypes”
“Custom Hatch Patterns”
“The Menu File”
“DIESEL”
“Customize the Status Line”
“Introduction to Programming Interfaces”
“Slides and Command Scripts”

Organize Program and Support Files

You can change the default directory structure for the program and support files to suit your needs.

Overview of File Organization

AutoCAD uses support files for purposes such as storing menu definitions, loading AutoLISP and ObjectARX applications, and describing text fonts.

The default directory structure for the AutoCAD program and support files is designed to efficiently organize those files into logical groups. If this organization does not suit your needs, you can change it. However, some applications look for certain files in specific locations, and you should verify that your modifications do not conflict with the requirements of those applications. Without the full path, including drive and directory, AutoCAD can locate only files that are found in the library search path.

The location of the *support* folder changed in AutoCAD 2004. The location of local customizable files is stored in the LOCALROOTPREFIX system variable. The location of roamable customizable files is stored in the ROAMABLEROOTPREFIX system variable. If a network supports roaming, customizable files in the user's roaming profile are available on the machine the user is logged onto.

The following LISP script creates the CUSTFILES command, which launches Windows Explorer in the correct folder.

```
(defun c:custfiles ()
  (command "shell"
    (strcat "explorer \"\" (getvar "roamablerootprefix") "\\")
  )
  (princ)
)
```

Library Search Path

The library search path specifies where the program searches for files when you do not specify a full path name, as follows:

- Current directory. (This is typically determined by the “Start In” setting in your shortcut icon.)
- Directory that contains the current drawing file.

- Directories listed in the search path specified on the Files tab in OPTIONS. (See “Specify Search Paths, File Names, and File Locations” on page 100 in the *User’s Guide*.)
- Directory that contains the AutoCAD program files.

Depending on the current environment, two or more directories may be the same.

If a file is not in this search path, you must specify both its path name and file name before AutoCAD can find it. For example, if you want to insert the *part5.dwg* drawing into your current drawing and it is not in the library search path, you must specify its full path name, as shown here:

Command: **insert**

Enter block name or [?]: **/files2/olddwgs/part5**

If the drawing exists at that location, AutoCAD prompts you to finish the INSERT command in the usual manner.

Directory Structure

AutoCAD uses tree-structured directories and subdirectories. It is recommended that you keep supplemental files (such as AutoLISP applications and menu files) separate from the AutoCAD program and support files. This makes it easier to track possible conflicts and to upgrade each application without affecting the others.

The default location for AutoCAD is in the *Program Files* folder. You can create a new directory on the same level, for example, */AcadApps*, and store your custom AutoLISP and VBA macros, menu files, and other third-party applications in subdirectories on the next level. If you want to maintain multiple drawing directories (for separate job files), you can create a directory, such as */AcadJobs*, with subdirectories for each job.

Command Search Procedure

When you enter a command, AutoCAD goes through a series of steps to evaluate the validity of the command name. A command can be a built-in command or system variable, an external command or alias defined in the *acad.pgp* file, or a user-defined AutoLISP command. Commands can also be defined by ObjectARX applications or a device driver command. You can enter a command on the command line or choose a command from the appropriate menu. Commands can be entered also from a script file or by an AutoLISP or ObjectARX application.

The following list describes the search order AutoCAD uses to validate a command name.

- 1 If the input is a null response (SPACEBAR or ENTER), AutoCAD uses the name of the last command issued. HELP is the default.
- 2 AutoCAD checks the command name against the list of built-in commands. If the command is in the list and is not preceded by a period (.), AutoCAD then checks the command against a list of undefined commands. If the command is undefined, the search continues. Otherwise, the command is run, unless another reason prevents it from doing so. Running it transparently or in Perspective mode might be impossible.
- 3 AutoCAD checks the command name against the names of commands defined by a device driver, and then by those defined by the display driver.
- 4 AutoCAD checks the command name against the external commands defined in the program parameters file (*acad.pgp*). If the command name corresponds to a defined external command, that command runs, and the search is complete.
- 5 AutoCAD checks the command name against the list of commands defined by AutoLISP or ObjectARX applications. At this point, an auto-loaded command is loaded.
- 6 AutoCAD checks the command name against the list of system variables. If the command name is in the list, AutoCAD executes the SETVAR command, using the input as the variable name.
- 7 If the command name corresponds to a command alias defined in the program parameters file, AutoCAD uses the expanded command name and continues the search, starting a new search against the list of built-in commands.
- 8 If all the preceding steps fail, the search terminates with a warning message about illegal command names.

See Also

“Overview of AutoLISP Automatic Loading” on page 139

“Specify Search Paths, File Names, and File Locations” in the *User’s Guide*

Multiple Configurations

When you configure AutoCAD for a pointing device and plotter drivers, the information you supply is recorded in a configuration file. The default location of the *acad2005.cfg* configuration file is listed on the Files tab in the Options dialog box under Menu, Help, and Miscellaneous File Names, but you can specify an alternate path, file name, or both.

Typically, only a single configuration is necessary, but sometimes you may need multiple configurations. For example, if you use a mouse for most of your work but occasionally require a large digitizing tablet, you can set up your system to handle multiple configurations rather than reconfiguring each time you change devices.

The values of numerous AutoCAD system variables and the configuration options in the Options dialog box are stored in the configuration file. If you want different settings for these variables and operating parameters, you can save their respective values to different configuration files. For a list of the system variables and where they are stored, see “System Variables” in the *Command Reference*.

To take advantage of multiple configurations, you must set up AutoCAD to use different configuration files. Use the `/c` switch to specify alternate configuration files at startup.

See Also

“Customize Startup” in the *User’s Guide*

Multiple Drawing Folders

Maintaining multiple drawing directories is not only convenient, but also often necessary. Keeping your drawing and other associated files in separate directories makes it easier to perform basic file maintenance. The scenario described in this topic is based on the sample directory structure described in “Overview of File Organization” on page 4, but you can expand or alter it to meet your needs.

You can set up the `/AcadJobs` directory to contain your drawing subdirectories. The drawing subdirectories can contain other subdirectories that hold related support files for that particular drawing type or job. The `/AcadJobs/Job1/Support` directory can contain blocks and AutoLISP files specific to the drawing files in `/AcadJobs/Job1`. Specifying **support** (with no path prefix) in the Support path adds the *support* directory within the current directory to the Support path. Notice that if you use the Options dialog box to specify a directory, AutoCAD creates a hard-coded path to that directory. To use the relative naming convention previously described, you must specify the Support path with the `/s` switch on the command line. See “Customize Startup” in the *User’s Guide*.

If you make sure that the required drawing directory is the current directory when you start AutoCAD, all files and subdirectories in that directory are easily accessible. You can create a program icon or Start menu item that specifies the correct working directory for each job.

You can use a batch program as an alternative to using icons or menus. With batch programs you can create new job directories automatically. The following batch program verifies that a specified directory exists, sets that directory to be current, and then runs AutoCAD.

```
@echo off
C:
if exist \AcadJobs\Jobs\%1 goto RUNACAD
echo.
echo *** Creating \AcadJobs\Jobs\%1
echo *** Press Ctrl+C to cancel.
echo.
pause
mkdir \AcadJobs\Jobs\%1
:RUNACAD
cd \AcadJobs\Jobs\%1
start C:\AutoCAD\acad.exe
```

Using Notepad, or any ASCII text editor, save this batch program to a file named *acad.bat*. Be sure to change the drive and directory names to match those on your system. Place this file in a directory that is on your system search path (for example, *C:\winnt*). You can run this batch program using the Run command on the Start menu or by double-clicking the file in Explorer. If you saved the file as *acad.bat*, use the following syntax:

acad jobname

where *jobname* is the name of the job directory to make current.

Create Custom Toolbars

You can add buttons to toolbars, remove buttons you use infrequently, and rearrange buttons and toolbars. You can also create your own toolbars and create or change the button image associated with a command.

Create, Rename, and Delete Toolbars

When you create a new toolbar, you first assign it a name. The new toolbar is displayed as “empty” or without buttons. You then drag buttons onto the new toolbar from existing toolbars or from commands listed on the Commands tab in the Customize dialog box.

On the Toolbars tab of the Customize dialog box, the Menu Group list displays the menu group with which the selected toolbar is associated. For example, ACAD represents the AutoCAD menu group. The menu group determines the file in which your toolbar configuration is saved.

Note If you have not customized toolbars before, try experimenting on new toolbars before you change an existing one.

To create a toolbar

- 1 From the Tools menu, choose Customize. Then choose Toolbars.

The Toolbars tab is automatically selected.

- 2 On the Toolbars tab, choose New.

- 3 In the New Toolbar dialog box, enter a name for the toolbar.

- 4 Under Save Toolbar in Menu Group, specify the menu group associated with the toolbar.

A menu group is a file that contains a group of menu items. The default menu group is ACAD, and it contains the menu that is loaded into AutoCAD by default.

- 5 Choose OK. Then choose Close.

The new, empty toolbar is displayed. You can now add buttons to the toolbar using the method described in “To move or copy a button from another toolbar” on page 15.

Fast Method

- 1 From the Tools menu, choose Customize. Then choose Toolbars.

- 2 On the Commands tab, select a command from the list and drag it to an empty place on your screen.

A toolbar with the selected command is automatically created.

Command line CUSTOMIZE

To rename a toolbar

- 1 From the Tools menu, choose Customize. Then choose Toolbars.

The Toolbars tab is automatically selected.

- 2 Under Toolbars, select the toolbar name you want to change.

- 3 Choose Rename.

- 4 In the Rename Toolbar dialog box, enter a new name.

- 5 Choose OK. Then choose Close.

Command line CUSTOMIZE

To delete a toolbar

- 1 From the Tools menu, choose Customize. Then choose Toolbars.
The Toolbars tab is automatically selected.
- 2 Under Toolbars, select the toolbar you want to delete.
- 3 Choose Delete.
A confirmation box is displayed.
- 4 Choose OK to delete the toolbar.
- 5 Choose Close.

Command line CUSTOMIZE

Modify the Appearance of a Toolbar

You can reposition buttons on a toolbar, add space between them, and adjust the toolbar's shape. A toolbar changes size automatically when buttons are added or deleted.

You can also change the size of buttons and specify information to be displayed when the cursor passes over a button.

When the Customize dialog box is open, you can right-click any button and use a shortcut menu to

- Copy or move buttons from one toolbar to another
- Edit buttons
- Delete buttons
- Reset the properties of a button to its default state
- Display the properties of buttons

To reposition buttons on a toolbar

- 1 From the Tools menu, choose Customize. Then choose Toolbars.
The Customize dialog box is displayed.
- 2 On any docked or floating toolbar, drag a button to a new position.
You must drag the button more than halfway across the button that already exists in the new location.
- 3 Choose Close to close the dialog box.

Command line CUSTOMIZE

To add space between buttons on a toolbar

- 1 From the Tools menu, choose Customize. Then choose Toolbars.
The Customize dialog box is displayed.
- 2 On the toolbar you want to modify, drag the button to the left or right edge of the button beside it, but not past the middle.
- 3 Choose Close to close the dialog box.

Command line CUSTOMIZE

To change the shape of a floating toolbar

- To create a vertical toolbar, drag the bottom border of the toolbar. To create a horizontal toolbar, drag a side border.
- To create a square toolbar, drag a corner of the toolbar to size it.

Note A docked toolbar cannot be reshaped.

To change the appearance or accessibility of a toolbar

- 1 From the Tools menu, choose Customize. Then choose Toolbars.
- 2 Configure the toolbar by turning on or turning off the following options:
 - **Large Buttons.** Changes the size of buttons from 16 by 15 pixels to 32 by 30 pixels.
 - **Show ToolTips on Toolbars.** Displays a pop-up label describing each button as the cursor passes over it.
 - **Show Shortcut Keys on Toolbars.** Shows the shortcut key whenever a tooltip is displayed.
- 3 Choose Close to save your selections.

For more information about entering shortcut keys for buttons, see “Add Shortcut Keys to Menus and Toolbars” on page 16.

Command line CUSTOMIZE

Create and Edit Toolbar Buttons

After you create a toolbar, you can add buttons that are supplied by Autodesk, or you can create your own by modifying existing buttons.

Create a Flyout

A flyout is a set of buttons nested under a single button on a toolbar. Flyout buttons have a black triangle in the lower-right corner. To create a flyout, you add a flyout image to a button and associate a toolbar with it.

Change a Button

You can associate buttons with different commands or create your own button.

Edit a Button Image

AutoCAD provides standard button images for buttons that start AutoCAD commands. You can create custom button images to run custom macros. You can either modify an existing button image or create your own. Button images are saved as BMP files. The BMP files must be saved in a directory that is on the support path.

See Also

“Create Menu Macros” on page 54

“Overview of File Organization” on page 4

To change the command associated with a button

- 1 From the Tools menu, choose Customize. Then choose Toolbars.
- 2 Choose the Properties tab. Then choose the button whose command you want to change.
- 3 Under Macro Associated with This Button, edit the command information to specify a new command for the button.

For most commands, start with ^C^C to cancel a command that may be running and display the Command prompt. An underscore (_) character enables commands to work on international versions of AutoCAD. An apostrophe (') enables the command to work transparently. If you associate a series of commands with a button, separate them with semicolons or spaces. You create button macros the same way you create menu macros.

For more information about entering commands in macros, see “Create Menu Macros” on page 54.

- 4 Choose Apply. Then choose Close.

Command line CUSTOMIZE

To create a flyout

- 1 From the Tools menu, choose Customize. Then choose Toolbars.
- 2 In the Customize dialog box, choose the Commands tab.
- 3 On the Commands tab under Categories, choose User-Defined.
- 4 Drag the flyout button image from the Commands box to the destination toolbar.

If you do not have a destination toolbar, AutoCAD creates one when you drop the button anywhere except on another toolbar.

- 5 Under Toolbar Associated with This Flyout, select the toolbar to associate with the flyout.

The buttons for the selected toolbar are displayed to the right of the list box. The most recently used button is displayed as the button for the flyout.

- 6 Choose Apply. Then choose Close or choose another tab.

Command line CUSTOMIZE

To edit or create a button image

- 1 On the Tools menu, click Customize ► Toolbars.

The Toolbars tab is automatically selected.

- 2 Click the toolbar button you want to edit on any toolbar.

The Button Properties tab is automatically selected. The Button Editor displays the button image in the upper-left corner of the dialog box.

Note You cannot edit flyout buttons.

- 3 Click Edit to display a close-up view of the button.
- 4 In the Button Editor, choose one of the following options:
 - Select Grid to divide the view into a grid. Each grid box represents one pixel.
 - Click Open to open an existing button image.
- 5 In the Button Editor, use the Pencil, Line, Circle, and Erase buttons to create or edit the button image. To use color, select a color from the color palette, or click More to open the Select Color dialog box, True Color tab.

- The Pencil button edits one pixel at a time in the selected color. You can drag the pointing device to edit several pixels at once.
 - The Line button creates lines in the selected color. Press the pick button on the pointing device to set the first endpoint of the line. Drag to draw the line. Release the pick button to complete the line.
 - The Circle button creates circles in the selected color. Press the pick button on the pointing device to set the center of the circle. Drag to set the radius. Release the pick button to complete the circle.
 - The Erase button sets the pixels to white. Click Erase to set all pixels to white.
- 6 To save the customized button, click Save. Use Save As to save it under a different name.
 - 7 Click Close to close the Button Editor. To edit a different button, click the button in the list under Button Image.
 - 8 Click Close to exit the Customize dialog box.

Command line CUSTOMIZE

To create a new button for a toolbar

- 1 On the Tools menu, click Customize ► Toolbars.
- 2 In the Customize dialog box, Commands tab, select User-Defined from the Category list.
- 3 Under Commands, drag User Defined Button to an existing toolbar. To create a new toolbar, drag the button to an empty space on your screen.
- 4 Select the new (blank) button to display its properties.
The Button Properties tab is automatically displayed.
- 5 On the Button Properties tab in the Name box, enter a new name for the button.
- 6 In the Description field, enter the text you want to appear on the status line.
- 7 To associate a command with the button, enter the command in the box under Macro.

For most commands, start with ^C^C to cancel a command that may be running and display the Command prompt. An underscore (_) character enables commands to work on international versions of AutoCAD. An apostrophe (') enables the command to work transparently. If you associate a series of commands with a button, separate them with semicolons or spaces. You create button macros the same way you create menu macros. For more information about entering commands in macros, see “Create Menu Macros” on page 54.

- 8 Under Button Image, select a button image for the button.
- 9 Click Apply and then close the dialog boxes.

Command line CUSTOMIZE

To add a button to a toolbar

- 1 Display the toolbar you want to customize by right-clicking any toolbar and choosing a toolbar from the list.
- 2 On the Tools menu, click Customize ► Toolbars.
- 3 In the Customize dialog box, Commands tab, clear the check box next to Show Command Name so that all button images are displayed under Commands.
- 4 Drag the button to the toolbar you want to add it to.
The toolbar size changes to fit the buttons you add.
- 5 Click Close to close the dialog box.

Command line CUSTOMIZE

To delete a button from a toolbar

- 1 On the Tools menu, click Customize ► Toolbars.
The Toolbars tab is automatically selected.
- 2 Drag the button you want to delete off its toolbar.
- 3 Click OK when the confirmation dialog box is displayed.
- 4 Click Close to close the dialog box.

If you delete a button supplied with AutoCAD from a toolbar, you can restore it from the Commands tab of the Customize Toolbars dialog box.
If you delete a button you have created or customized, you cannot restore it.

Command line CUSTOMIZE

To move or copy a button from another toolbar

- 1 On the Tools menu, click Customize ► Toolbars.
- 2 To move a button from one toolbar to another, drag the button to the new toolbar.
- 3 To copy a button from from one toolbar to another, hold down CTRL and drag the button image from one toolbar to the new toolbar.
- 4 Click Close to close the dialog box.

Command line CUSTOMIZE

Add Shortcut Keys to Menus and Toolbars

You can assign shortcut keys (sometimes called accelerator keys) to commands you use frequently. For example, pressing CTRL+O opens a file and pressing CTRL+S saves a file, just as choosing Open and Save from the File menu do. Shortcut keys can be associated with both menu items and toolbar buttons. You can create new shortcut keys or modify existing shortcut keys.

Shortcut keys are interpreted using AutoCAD classic standards (pre-Windows) or Microsoft® Windows® standards. AutoCAD uses internal shortcut key assignments when

- Windows Standard Accelerator Keys is not selected in the Options dialog box, User Preferences tab
- The menu file *acad.mnu* is unloaded in the Menu Customization dialog box

The following table lists the default actions for both types of settings.

Shortcut key assignments		
Shortcut keys(s)	AutoCAD classic shortcuts	Windows shortcuts
CTRL+A	Toggles group selection on/off	Selects objects in drawing
CTRL+B	Toggles Snap	Toggles Snap
CTRL+C	Cancels current command	Copies objects to Clipboard
CTRL+D	Toggles coordinate display	Toggles coordinate display
CTRL+E	Cycles through isometric planes	Cycles through isometric planes
CTRL+F	Toggles running object snaps	Toggles running object snaps
CTRL+G	Toggles Grid	Toggles Grid
CTRL+H	(No action)	Toggles PICKSTYLE on/off
CTRL+J	Executes last command	Executes last command
CTRL+L	(No action)	Toggles Ortho mode
CTRL+M	Repeats last command	(No action)

Shortcut key assignments (*continued*)

Shortcut keys(s)	AutoCAD classic shortcuts	Windows shortcuts
CTRL+N	(No action)	Creates a new drawing
CTRL+O	Toggles Ortho mode	Opens existing drawing
CTRL+P	(No action)	Prints current drawing
CTRL+R	(No action)	Cycles layout viewports
CTRL+S	(No action)	Saves current drawing
CTRL+T	Toggles Tablet mode	Toggles Tablet mode
CTRL+V	Cycles layout viewports	Pastes data from Clipboard
CTRL+X	Cancels current input	Cuts objects to Clipboard
CTRL+Y	(No action)	Repeats last action
CTRL+Z	(No action)	Reverses last action
CTRL+[	Cancels current command	Cancels current command
CTRL+\	Cancels current command	Cancels current command
F1	Displays Help	Displays Help
F2	Toggles text window on/off	Toggles text window on/off
F3	Toggles OSNAP	Toggles OSNAP
F4	Toggles TABMODE	Toggles TABMODE
F5	Toggles ISOPLANE	Toggles ISOPLANE
F6	Toggles COORDS	Toggles COORDS
F7	Toggles GRIDMODE	Toggles GRIDMODE
F8	Toggles ORTHOMODE	Toggles ORTHOMODE
F9	Toggles SNAPMODE	Toggles SNAPMODE

See Also

“The Menu File” on page 45

To create a shortcut key for a toolbar or menu

- 1 From the Tools menu, choose Customize. Then choose Keyboard.
The Keyboard tab is automatically selected.
- 2 Under Categories, select the toolbar or menu that contains the command for which you want to modify shortcut key assignments. If you can't find the right category, click AutoCAD Commands to display all AutoCAD commands.
- 3 Under Commands, select the command for which you want to modify the shortcut key assignment.
The current shortcut key, if any, is displayed under Current Keys.
- 4 Click inside the Press New Shortcut Key box. Hold down a modifier key and press a letter key. Modifier keys can include CTRL, ALT, CTRL+SHIFT, and ALT+SHIFT.
- 5 Currently Assigned To displays the current assignment (if any) for the shortcut key. If you do not want to replace the current assignment, go back to step 4 and use a different shortcut key.
- 6 Choose Assign. Then choose Close.

Command line CUSTOMIZE

To remove a shortcut key from a toolbar or menu

- 1 From the Tools menu, choose Customize. Then choose Keyboard.
The Keyboard tab is automatically selected.
- 2 Under Categories, select the toolbar or menu that contains the command for which you want to delete the shortcut key assignment. If you can't find the right category, click AutoCAD Commands to display all the AutoCAD commands.
- 3 Under Commands, select the command for which you want to remove the shortcut key assignment.

Note If you cannot find the command associated with a shortcut key, type the shortcut in the Press New Shortcut Key box and press ENTER to see what command it is currently assigned to.

- 4 Choose Remove. Then choose Close.

Command line CUSTOMIZE

Customize a Publish to Web Template

You can create customized templates for use in the Publish to Web wizard by modifying one of the Publish to Web template (PWT) files provided. Use any HTML editor or text editor.

To create a custom template you can add or modify any of the following elements:

- Images
- Text
- Hyperlinks
- Color
- Title
- Video, animation, and so on

There are four default Publish to Web templates that you can customize:

- **Array of Thumbnails.** Creates a web page containing an array of thumbnail images.
- **Array Plus Summary.** Creates a web page containing an array of thumbnail images and summary information about each image.
- **List of Drawings.** Creates a web page containing a list of drawings and an image frame.
- **List Plus Summary.** Creates a web page containing a list of drawings, an image frame, and summary information about a selected image.

Note You must be familiar with HTML syntax in order to customize the Publish to Web templates.

You can make changes or additions to the look and feel of a template, but you cannot change the arrangement of images within it. For example, in the *Array of Thumbnails* template, the images are presented across the page in rows. You cannot alter the presentation of the images, but you can wrap text and graphics around this table of images.

Warning! To ensure that you do not overwrite the default Publish to Web template files, back up these files before you make any changes to them.

To create quick access to the Publish to Web templates

- 1 On the Tools menu, click Options.
- 2 In the Options dialog box, Files tab, click the plus sign next to Drawing Template Settings and then click the plus sign next to Drawing Template File Location.
- 3 Move the cursor to the path name that is displayed and click inside it, and press F2, and press CTRL + C to copy it.
- 4 Click OK to close the Options dialog box.
- 5 On the File menu, click Open.
- 6 In the Select File dialog box, right-click an empty area in the vertical panel on the left side, and click Add on the shortcut menu.
- 7 Enter a name in the Item name box (for example, **Templates**).
- 8 Press CTRL + V to paste the path into the Item Path box, and click OK.
You can now access the Templates folders by clicking the button in the left panel of the Select File dialog box.

To customize a Publish to Web template

- 1 On the File menu, click Open, and access the Publish to Web template folder.
See “To create quick access to the Publish to Web templates” on page 20.
- 2 Double-click the *PTWTemplates* folder to open it. The following folders are displayed. Each contains a Publish to Web template and preview images (BMP) that you see when you run the Publish to Web wizard.
 - **Template1.** Contains the *Array of Thumbnails* template and a preview image
 - **Template2.** Contains the *Array Plus Summary* template, a preview image, and HTML frames
 - **Template3.** Contains the *List of Drawings* template, a preview image, and HTML frames
 - **Template4.** Contains the *List Plus Summary* template, a preview image, and HTML frames
- 3 Right-click the folder you want to use, and click Copy.
- 4 Press ALT+2, right-click the *PTWTemplates* folder, and click Paste.
- 5 Reopen the *PTWTemplates* folder, and right-click the new folder and rename it.
- 6 Right-click the new folder and click Open to display its contents.

- 7 Rename the Publish to Web template (PWT) file with an *.htm* or *.html* file extension.
- 8 Open the template file in an HTML editor or a text editor.
The template file contains comments that help you determine which areas of the code you can modify to create your new web page.
- 9 Review the comments and make changes to the parts of the template you want to customize.
- 10 Save the template with a *.pwt* file extension. Make sure you save the file to the template folder you created in step 3.

Note Each template folder can contain only one PWT file. If you create a new PWT file, make sure you delete any other PWT files that exist in the same folder.

When you run the Publish to Web wizard, the new template is displayed in the list of templates.

Define Custom Commands

You can define external commands to run from within AutoCAD and create command aliases for AutoCAD commands in the program parameters file, *acad.pgp*, an ASCII file that stores command definitions.

Define External Commands

While AutoCAD is running, you can invoke other programs or utilities, such as the following:

- Windows system commands and utilities, such as **start**, **type**, **dir**, or **copy**
- Applications such as text editors or word processors
- Database managers, spreadsheets, and communications programs
- User-supplied programs, such as batch files or VBA macros

When you enter a command that isn't a normal command, AutoCAD looks for the command in *acad.pgp*, a file that stores command definitions. The first section of *acad.pgp* defines external commands. You can add command definitions by editing *acad.pgp* in Notepad or any text editor that saves files in ASCII format. To open the PGP file, on the Tools menu, click Customize ► Edit Custom Files ► Program Parameters (*acad.pgp*).

Note Before editing *acad.pgp*, first create a backup so that you can restore it later, if necessary.

When you define an external command, you specify a command name to be used at the Command prompt and an executable command string that is passed to the operating system. Each line in the external commands section has five comma-delimited fields as follows:

```
command,[executable],flags[,[*]prompt[,return_code]]
```

command The command to enter at the Command prompt. If this name is an internal AutoCAD command name, it is ignored. The name is not case-sensitive.

executable The constant string sent to the operating system when you enter the command name. It can be any command that you can execute at the operating-system prompt. The string can include switches or parameters. The case sensitivity of this string depends on the application you are running.

flags A required bitcoded parameter. Add these integer values together in any combination to achieve the result you want.

- 0 Start the application and wait for it to finish.
- 1 Don't wait for the application to finish.
- 2 Run the application minimized.
- 4 Run the application "hidden."
- 8 Put the argument string in quotes.

Bit values 2 and 4 are mutually exclusive; if both are specified only the 2 bit is used. Using values 2 or 4 without value 1 should be avoided, because AutoCAD becomes unavailable until the application has completed.

Bit value 8 allows commands like **del** to work properly with file names that have embedded spaces. Note that this eliminates the possibility of passing a space-delimited list of file names to these commands. If you prefer multiple file support, do not use the bit value 8.

prompt

An optional field. It specifies the prompt to display on the AutoCAD command line. The response to this prompt is appended to the string supplied in the executable field. If the first character of the prompt field is an asterisk (*), the response can contain spaces and the user must press ENTER to terminate it. Otherwise, the response is terminated by either SPACEBAR or ENTER. If no prompt is specified, no input is requested; however, you must add a comma if a return code is to be supplied or if you want the prompt to have a trailing space.

return_code

An optional bitcoded parameter. You can add these integer values together in any combination to achieve the result you want. For example, if values 1 and 2 are required, you use 3 as the return code. The values are defined as follows (codes 0 and 4 are meaningless in a windowed environment and are therefore not included):

1 *Loads a DXB file.* AutoCAD loads the DXB file named *\$cmd.dxb* into the drawing after the command is terminated. After the DXB file is loaded, the *\$cmd.dxb* file is deleted. This action produces the same result as the DXBIN command.

2 *Constructs a block definition from a DXB file.* AutoCAD creates a block definition from the DXB file named *\$cmd.dxb*. The response to the prompt field is used as the block name. This name must be a valid block name that does not currently exist in the drawing; therefore, this mode cannot redefine a previously defined block. After AutoCAD loads the DXB file, the *\$cmd.dxb* file is deleted. The default name for the INSERT command is set to the newly defined block.

The file can also contain comment lines preceded by a semicolon (;).

Windows System Commands

The **start** and **cmd** Windows system commands are very useful when defining external commands. If you specify an executable string that does not use the **start** or **cmd** commands, AutoCAD is unavailable until that window is closed.

The **start** command starts a separate window and runs a specified program or command. If **start** is used without any parameters, it opens a new command prompt window. The **start** command has many command line switches that affect the display of the new window. To launch a Windows application, use **start** without any switches. The **start** command is also very useful for starting a document that is associated with an application. For example, you can use **start** to directly open a document created with a word processor or an HTML file.

The **cmd** command opens a Command prompt window that acts as a shell of AutoCAD. This window must be closed before control returns to the AutoCAD Command prompt. Two command line switches, **/c** and **/k**, are useful for external commands. The **/c** switch carries out the specified command and then stops (the window closes). The **/k** switch carries out the specified command and then continues (the window remains open). When using the **/k** switch, you must close the command window (with the **exit** command).

In general, use **start** to start a new window or application that is to be a separate process from AutoCAD. Use **cmd** to run a batch file or command script that does not create a separate window, or to create a window that must be closed before control is passed back to AutoCAD. For more information about these commands and switches, see your Windows system command documentation.

Custom-Defined Commands

The following example defines three new commands: RUN, LISTSET, and DXB2BLK.

```
RUN, cmd /c,0,*Batch file to run: ,  
LISTSET,cmd /k SET,0  
DXB2BLK,cmd /c DXBCOPY,0,DXB file: ,2
```

The RUN command runs a batch file or command script. The **cmd** command followed by the **/c** switch opens a command window, runs the batch file, and then closes.

The LISTSET command displays the current DOS environment variable settings. Because this example uses **cmd /k** rather than **start**, the command window must be closed before returning to AutoCAD. If you want this window to remain active, use **start /realtime**. For more information about these commands and switches, see your Windows system command documentation.

The DXB2BLK command creates a block definition from the specified DXB file. The DXB file converts all objects into lines. One beneficial by-product of this procedure is that it provides a simple method for exploding text objects into lines.

DXB2BLK passes the specified DXB file name to the *dxbcopy* batch file, which copies that file name to the file name *\$cmd.dxb*. AutoCAD then creates a block from the specified DXB file. The name provided to the DXB file prompt is used as the new block name. To create the *dxbcopy.cmd* file, enter the following at the Windows Command Prompt:

```
echo copy %1.dxb $cmd.dxb > dxbcopy.cmd
```

This creates the *dxbcopy.cmd* file in the current directory. Move this file to a directory that is in your DOS path, or explicitly specify the file's location in the *acad.pgp* file. For example, if the *dxbcopy.cmd* file is in *D:\cad*, enter the following in the external commands section of your *acad.pgp* file.

```
DXB2BLK, cmd /c D:\CAD\DXBCOPY,0,DXB file: ,2
```

To create a DXB file, choose AutoCAD DXB File Format as the current printer, and then plot to a file. For more information about configuring printers, see “Set Up Plotters and Printers” in the *Driver & Peripheral Guide*.

To open the program parameters file (*acad.pgp*)

- On the Tools menu, click Customize ► Edit Custom Files ► Program Parameters (*acpl.pgp*).

Create Command Aliases

A command alias is an abbreviation that you enter on the command line instead of entering the entire command name.

For example, you can enter **c** instead of **circle** to start the CIRCLE command. An alias is not the same as a keyboard shortcut, which is a combination of keystrokes, such as CTRL+S for SAVE.

An alias can be defined for any AutoCAD command, device driver command, or external command. The second section of the *acad.pgp* file defines command aliases. You can change existing aliases or add new ones by editing *acad.pgp* in Notepad or any text editor that saves files in ASCII format. To open the PGP file, on the Tools menu, click Customize ► Edit Custom Files ► Program Parameters (*acad.pgp*). The file can also contain comment lines preceded by a semicolon (;).

Note Before editing *acad.pgp*, first create a backup so that you can restore it later, if necessary.

To define a command alias, add a line to the command alias section of the *acad.pgp* file using the following syntax:

*abbreviation,*command*

where *abbreviation* is the command alias that you enter at the Command prompt and *command* is the command being abbreviated. You must enter an asterisk (*) before the command name to identify the line as a command alias definition.

If you can enter a command transparently, you can also enter its alias transparently. When you enter the command alias, the full command name is displayed at the Command prompt and the command is executed.

You can create command aliases that include the special hyphen (–) prefix, such as those listed here, that access the command line version of certain commands.

```
BH, *-BHATCH  
BD, *-BOUNDARY
```

Note You cannot use command aliases in command scripts. Using command aliases in menu files is not recommended.

If you edit *acad.pgp* while AutoCAD is running, enter **reinit** in order to use the revised file. Restarting AutoCAD automatically reloads the file.

See Also

“Add Shortcut Keys to Menus and Toolbars” on page 16

Custom Linetypes

2

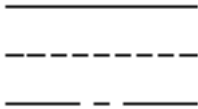
AutoCAD provides a library of standard linetypes in the *acad.lin* and *acadiso.lin* files. You can use the linetypes as they are, modify them, or create your own custom linetypes.

In this chapter

- Overview of Linetype Definitions
- Simple Custom Linetypes
- Text in Custom Linetypes
- Shapes in Custom Linetypes

Overview of Linetype Definitions

The linetype name and definition determine the particular dash-dot sequence, the relative lengths of dashes and blank spaces, and the characteristics of any included text or shapes. You can use any of the standard linetypes that AutoCAD® provides, or you can create your own linetypes.



examples of linetypes

Linetypes are defined in one or more linetype definition files that have an *.lin* file extension. An LIN file can contain definitions of many simple and complex linetypes. You can add new linetypes to an existing LIN file, or you can create your own LIN file. To create or modify linetype definitions, edit the LIN file using a text editor or word processor or use LINETYPE at the Command prompt.

When you create a linetype, you must load the linetype before you can use it.

The LIN files included in AutoCAD are *acad.lin* and *acadiso.lin*. You can display or print these text files to better understand how to construct linetypes.

Simple Custom Linetypes

Each linetype is defined on two lines in a linetype definition file. The first line contains the linetype name and an optional description. The second line is the code that defines the actual linetype pattern.

The second line must begin with the letter A (alignment), followed by a list of pattern descriptors that define pen-up lengths (spaces), pen-down lengths (dashes), and dots. You can include comments in an LIN file by beginning the line with a semicolon (;).

Linetype Definition Format

The format of the linetype definition is

```
*linetype_name,description  
A,descriptor1,descriptor2, ...
```

For example, a linetype called DASHDOT is defined as

```
*DASHDOT,Dash dot  
A,.5,-.25,0,-.25
```

This indicates a repeating pattern starting with a dash 0.5 drawing units long, a space 0.25 drawing units long, a dot, and another space 0.25 drawing units long. This pattern continues for the length of the line, ending with a dash 0.5 drawing units long. The linetype would be displayed as shown below.

— · — · — · — · — · — · —

LIN files must be saved in ASCII format and use an *.lin* file extension. Additional information about each field in a linetype definition follows.

Linetype Name

The linetype name field begins with an asterisk (*) and should provide a unique, descriptive name for the linetype.

Description

The description of the linetype should help you visualize the linetype when you edit the LIN file. The description is also displayed in the Linetype Manager and in the Load or Reload Linetypes dialog box.

The description is optional and can include

- A simple representation of the linetype pattern using ASCII text
- An expanded description of the linetype
- A comment such as "Use this linetype for hidden lines"

If you omit the description, do not insert a comma *after* the linetype name. A description cannot exceed 47 characters.

Alignment Field (A)

The alignment field specifies the action for pattern alignment at the ends of individual lines, circles, and arcs. Currently, AutoCAD supports only A-type alignment, which guarantees that the endpoints of lines and arcs start and stop with a dash.

For example, suppose you create a linetype called CENTRAL that displays the repeating dash-dot sequence commonly used as a centerline. AutoCAD adjusts the dash-dot sequence on an individual line so that dashes and line endpoints coincide. The pattern fits the line so that at least half of the first dash begins and ends the line. If necessary, the first and last dashes are lengthened. If a line is too short to hold even one dash-dot sequence, AutoCAD draws a continuous line between the endpoints. For arcs also, the pattern is adjusted so that dashes are drawn at the endpoints. Circles do not have endpoints, but AutoCAD adjusts the dash-dot sequence to provide a reasonable display.

You must specify A-type alignment by entering **a** in the alignment field.

Pattern Descriptors

Each pattern descriptor field specifies the length of segments making up the linetype, separated by commas (no spaces are allowed):

- A positive decimal number denotes a pen-down (dash) segment of that length.
- A negative decimal number denotes a pen-up (space) segment of that length.
- A dash length of 0 draws a dot.

You can enter up to 12 dash-length specifications per linetype, provided they fit on one 80-character line in the LIN file. You need to include only one complete repetition of the linetype pattern defined by pattern descriptors. When the linetype is drawn, AutoCAD uses the first pattern descriptor for the starting and ending dashes. Between the starting and ending dashes, the pattern dash specifications are drawn sequentially, beginning with the second dash specification and restarting the pattern with the first dash specification when required.

A-type alignment requires that the first dash length be 0 or greater (a pen-down segment). The second dash length should be less than 0 if you need a pen-up segment and more than 0 if you are creating a continuous linetype. You must have at least two dash specifications for A-type alignment.

To create a simple linetype

- 1 At the Command prompt, enter **-linetype**.
- 2 Enter **c** (Create).
- 3 Enter a name for the linetype and press ENTER.

The linetype name can include up to 255 characters. Linetype names can contain letters, digits, and the special characters dollar sign (\$), hyphen (-), and underscore (_). Linetype names cannot include blank spaces.

- 4 In the Create or Append Linetype File dialog box, select an LIN linetype library file from the File Name box and choose Save.

If you select an existing file, the new linetype name is added to the linetype names in the file.

- 5 Enter text that describes the new linetype (optional).
- 6 At the Enter Pattern prompt, specify the pattern of the line. Follow these guidelines:

- All linetypes must begin with a dash.
- Enter zeros for dots.
- Enter negative real numbers for spaces. The value defines the length of the space in drawing units.
- Enter positive real numbers for dashes. The value defines the length of the dash in drawing units.
- Separate each dot, dash, or space value from the next with a comma.
- Use a space between a dot and a dash.

7 Press ENTER to end the command.

Note When you create a linetype, it is not loaded into your drawing automatically. Use the Load option of LINTYPE.

Text in Custom Linetypes

Characters from text fonts can be included in linetypes. Linetypes with embedded characters can denote utilities, boundaries, contours, and so on. As with simple linetypes, lines are dynamically drawn as you specify the vertices. Characters embedded in lines are always displayed completely; they are never trimmed.

Embedded text characters are associated with a text style in the drawing. Any text styles associated with a linetype must exist in the drawing before you load the linetype.

The format for linetypes that include embedded characters is similar to that for simple linetypes in that it is a list of pattern descriptors separated by commas.

Character Descriptor Format

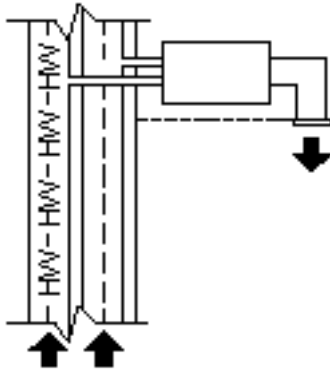
The format for adding text characters in a linetype description is as follows:

```
[ "text", textstyle, scale, rotation, xoffset, yoffset ]
```

This format is added as a descriptor to a simple linetype. For example, a linetype called HOT_WATER_SUPPLY is defined as

```
*HOT_WATER_SUPPLY,---- HW ---- HW ---- HW ---- HW ---- HW ----
A,.5,-.2,[ "HW", STANDARD, S=.1, R=0.0, X=-0.1, Y=-.05 ],-.2
```

This indicates a repeating pattern starting with a dash 0.5 drawing units long, a space 0.2 drawing units long, the characters *HW* with some scale and placement parameters, and another space 0.2 drawing units long. The text characters come from the text font assigned to the STANDARD text style at a scale of 0.1, a relative rotation of 0 degrees, an X offset of -0.1, and a Y offset of -0.05. This pattern continues for the length of the line, ending with a dash 0.5 drawing units long. The linetype would be displayed as shown below.



Notice that the total upstroke length is $0.2 + 0.2 = 0.4$ and that the text origin is offset -0.01 units in the X direction from the end of the first upstroke. An equivalent linetype would be

```
*HOT_WATER_SUPPLY,---- HW ---- HW ---- HW ---- HW ---- HW ---- HW ----
A,.5,-.1,["HW",STANDARD,S=.1,R=0.0,X=-0.0,Y=-.05],-.3
```

The total upstroke is still $0.1 + 0.3 = 0.4$, but the text origin is not offset in the X direction.

Additional information about each field in the character descriptor follows. The values to be used are signed decimal numbers such as 1, -17, and 0.01.

text	The characters to be used in the linetype.
text style name	The name of the text style to be used. If no text style is specified, AutoCAD uses the currently defined style.
scale	$s=value$. The scale factor to be used for the text style relative to the scale of the linetype. The height of the text style is multiplied by the scale factor. If the height is 0, the value for $s=value$ alone is used as the height.

rotation

R=value OR A=value. R= specifies relative or tangential rotation with respect to the line. A= specifies absolute rotation of the text with respect to the origin; that is, all text has the same rotation regardless of its position relative to the line. The value can be appended with a *d* for degrees (degrees is the default value), *r* for radians, or *g* for grads. If rotation is omitted, 0 relative rotation is used.

Rotation is centered between the baseline and the nominal cap height.

xoffset

X=value. The shift of the text on the X axis of the linetype, which is along the line. If *xoffset* is omitted or is 0, the text is elaborated with no offset. Use this field to control the distance between the text and the previous pen-up or pen-down stroke. This value is not scaled by the scale factor defined by *s=value*, but it is scaled to the linetype.

yoffset

Y=value. The shift of the text in the Y axis of the linetype, which is at a 90-degree angle to the line. If *yoffset* is omitted or is 0, the text is elaborated with no offset. Use this field to control the vertical alignment of the text with respect to the line. This value is not scaled by the scale factor defined by *s=value*, but it is scaled to the linetype.

To include text characters in linetypes

- 1 Create a simple linetype, as described in “To create a simple linetype” on page 30.
- 2 Add the text character descriptor within the linetype pattern, using the following format:
`["text",textstylename,scale,rotation,xoffset,yoffset]`
- 3 Press ENTER to exit LINETYPE.

Shapes in Custom Linetypes

A complex linetype can contain embedded shapes that are saved in shape files. Complex linetypes can denote utilities, boundaries, contours, and so on.

As with simple linetypes, complex lines are dynamically drawn as the user specifies vertices. Shapes and text objects embedded in lines are always displayed completely; they are never trimmed.

The syntax for complex linetypes is similar to that of simple linetypes in that it is a comma-delimited list of pattern descriptors. Complex linetypes can include shape and text objects as pattern descriptors, as well as dash-dot descriptors.

The syntax for shape object descriptors in a linetype description is as follows:

```
[shapename,shxfilename] or [shapename,shxfilename,transform]
```

where *transform* is optional and can be any series of the following (each preceded by a comma):

```
R=## Relative rotation  
A=## Absolute rotation  
S=## Scale  
X=## X offset  
Y=## Y offset
```

In this syntax, ## is a signed decimal number (1, -17, 0.01, and so on), the rotation is in degrees, and the remaining options are in linetype-scaled drawing units. The preceding *transform* letters, if they are used, must be followed by an equal sign and a number.

The following linetype definition defines a linetype named CON1LINE that is composed of a repeating pattern of a line segment, a space, and the embedded shape CON1 from the *ep.shx* file. (Note that the *ep.shx* file must be in the support path for the following example to work properly.)

```
*CON1LINE, --- [CON1] --- [CON1] --- [CON1]  
A,1.0,-0.25,[CON1,ep.shx],-1.0
```

Except for the code enclosed in square brackets, everything is consistent with the definition of a simple linetype.

As previously described, a total of six fields can be used to define a shape as part of a linetype. The first two are mandatory and position-dependent; the next four are optional and can be ordered arbitrarily. The following two examples demonstrate various entries in the shape definition field.

```
[CAP,ep.shx,S=2,R=10,X=0.5]
```

The code above draws the CAP shape defined in the *ep.shx* shape file with a scale of two times the unit scale of the linetype, a tangential rotation of 10 degrees in a counterclockwise direction, and an X offset of 0.5 drawing units before shape elaboration takes place.

```
[DIP8,pd.shx,X=0.5,Y=1,R=0,S=1]
```

The code above draws the DIP8 shape defined in the *pd.shx* shape file with an X offset of 0.5 drawing units before shape drawing takes place, and a Y offset of one drawing unit above the linetype, with 0 rotation and a scale equal to the unit scale of the linetype.

The following syntax defines a shape as part of a complex linetype.

```
[shapename,shapefilename,scale,rotate,xoffset,yoffset]
```

The definitions of the fields in the syntax follow.

shapename	The name of the shape to be drawn. This field must be included. If it is omitted, linetype definition fails. If <i>shapename</i> does not exist in the specified shape file, continue drawing the linetype but without the embedded shape.
shapefilename	The name of a compiled shape definition file (SHX). If it is omitted, linetype definition fails. If <i>shapefilename</i> is unqualified (that is, no path is specified), search the library path for the file. If <i>shapefilename</i> is fully qualified and not found at that location, remove the prefix and search the library path for the file. If it is not found, continue drawing the linetype but without the embedded shape.
scale	<i>S=value</i> . The scale of the shape is used as a scale factor by which the shape's internally defined scale is multiplied. If the shape's internally defined scale is 0, the <i>S=value</i> alone is used as the scale.
rotate	<i>R=value</i> Or <i>A=value</i> . R= signifies relative or tangential rotation with respect to the lines elaboration. A= signifies absolute rotation of the shape with respect to the origin; all shapes have the same rotation regardless of their relative position to the line. The value can be appended with a <i>d</i> for degrees (if omitted, degree is the default), <i>r</i> for radians, or <i>g</i> for grads. If rotation is omitted, 0 relative rotation is used.

<code>xoffset</code>	<code>x=value</code> . The shift of the shape in the <i>X</i> axis of the linetype computed from the end of the linetype definition vertex. If <code>xoffset</code> is omitted or is 0, the shape is elaborated with no offset. Include this field if you want a continuous line with shapes. This value is not scaled by the scale factor defined by <code>s</code> .
<code>yoffset</code>	<code>y=value</code> . The shift of the shape in the <i>Y</i> axis of the linetype computed from the end of the linetype definition vertex. If <code>yoffset</code> is omitted or 0, the shape is elaborated with no offset. This value is not scaled by the scale factor defined by <code>s</code> .

See Also

“Shapes and Shape Fonts” on page 153

Custom Hatch Patterns

3

AutoCAD provides a library of standard hatch patterns in the *acad.pat* and *acadiso.pat* files. You can use the hatch patterns as they are, modify them, or create your own custom hatch patterns.

In this chapter

- Overview of Hatch Pattern Definitions
- Hatch Patterns with Dashed Lines
- Hatch Patterns with Multiple Lines

Overview of Hatch Pattern Definitions

In addition to using the predefined hatch patterns that are supplied, you can design and create your own custom hatch patterns. Developing a hatch pattern definition requires knowledge, practice, and patience. Because customizing hatches requires familiarity with hatch patterns, it is not recommended for new users.

The hatch patterns supplied by AutoCAD® are stored in the *acad.pat* and *acadiso.pat* text files. You can add hatch pattern definitions to this file or create your own files.

Regardless of where the definition is stored, a custom hatch pattern has the same format. It has a header line with a name, which begins with an asterisk and is no more than 31 characters long, and an optional description:

```
*pattern-name, description
```

It also has one or more line descriptors of the following form:

```
angle, x-origin,y-origin, delta-x,delta-y,dash-1,dash-2, ...
```

The default hatch pattern ANSI31 shown in the Boundary Hatch and Fill dialog box looks like this:



and is defined as follows:

```
*ANSI31, ANSI Iron, Brick, Stone masonry  
45, 0,0, 0,.125
```

The pattern name on the first line, **ANSI31*, is followed by a description: *ANSI Iron, Brick, Stone masonry*. This simple pattern definition specifies a line drawn at an angle of 45 degrees, that the first line of the family of hatch lines is to pass through the drawing origin (0,0), and that the spacing between hatch lines of the family is to be 0.125 drawing units.

Hatch pattern definitions follow these rules:

- Each line in a pattern definition can contain up to 80 characters. You can include letters, numbers, and the special characters underline (), hyphen (-), and dollar sign (\$). However, you must begin a pattern definition with a letter or number, not a special character.
- AutoCAD ignores both blank lines and text to the right of a semicolon.

- Each pattern line is considered to be the first member of a line family, created by applying the delta offsets in both directions to generate an infinite family of parallel lines.
- The *delta-x* value indicates the displacement between members of the family in the direction of the line. It is used only for dashed lines.
- The *delta-y* value indicates the spacing between members of the family; that is, it is measured perpendicular to the lines.
- A line is considered to be of infinite length. A dash pattern is superimposed on the line.

The process of hatching consists of expanding each line in the pattern definition to its infinite family of parallel lines. All selected objects are checked for intersections with any of these lines; any intersections cause the hatch lines to be turned on and off as governed by the hatching style. Each family of hatch lines is generated parallel to an initial line with an absolute origin to guarantee proper alignment.

If you create a very dense hatch, AutoCAD may reject the hatch and display a message indicating that the hatch scale is too small or its dash length too short. You can change the maximum number of hatch lines by setting the MaxHatch system registry variable using SETENV.

To create a simple hatch pattern

- 1 Open the *acad.pat* or *acadiso.pat* file in a text editor that saves in ASCII format (for example, Microsoft® Windows Notepad).
- 2 Create a header line that includes an asterisk and a pattern name. The name of the hatch pattern is limited to 31 characters.
- 3 (Optional) To include a description in the header line, follow the pattern name with a comma and description text.
- 4 Create a descriptor line that includes
 - An angle at which the line is drawn
 - An *X,Y* origin point
 - A *delta-x* of 0
 - A *delta-y* of any value

Hatch Patterns with Dashed Lines

To define dashed-line patterns, you append dash-length items to the end of the line definition item. Each dash-length item specifies the length of a segment making up the line. If the length is positive, a pen-down segment is drawn. If the length is negative, the segment is pen-up, and it is not drawn. The pattern starts at the origin point with the first segment and cycles through the segments in circular fashion. A dash length of 0 draws a dot. You can specify up to six dash lengths per pattern line.

The hatch pattern ANSI33, shown in the Boundary Hatch and Fill dialog box, looks like this:



and is defined as follows:

```
*ANSI33, ANSI Bronze, Brass, Copper
45, .176776695,0, 0,.25, .125,-.0625
```

For example, to modify a pattern for 45-degree lines to draw dashed lines with a dash length of 0.5 units and a space between dashes of 0.5 units, the line definition would be

```
*DASH45, Dashed lines at 45 degrees
45, 0,0, 0,.5, .5,-.5
```

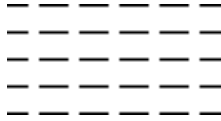
This is the same as the 45-degree pattern shown in “Overview of Hatch Pattern Definitions” on page 38 but with a dash specification added to the end. The pen-down length is 0.5 units, and the pen-up length is 0.5, meeting the stated objectives. If you wanted to draw a 0.5-unit dash, a 0.25-unit space, a dot, and a 0.25-unit space before the next dash, the definition would be

```
*DDOT45,Dash-dot-dash pattern: 45 degrees
45, 0,0, 0,.5, .5,-.25, 0,-.25
```

The following example shows the effect of *delta-x* specifications on dashed-line families. First, consider the following definition:

```
*GOSTAK
0, 0,0, 0,.5, .5,-.5
```

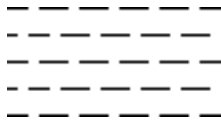
This draws a family of lines separated by 0.5, with each line broken equally into dashes and spaces. Because *delta-x* is zero, the dashes in each family member line up. An area hatched with this pattern would look like this:



Now change the pattern to

```
*SKEWED
0, 0,0, .5,.5, .5,-.5
```

It is the same, except that you have set *delta-x* to 0.5. This offsets each successive family member by 0.5 in the direction of the line (in this case, parallel to the X axis). Because the lines are infinite, the dash pattern slides down the specified amount. The hatched area would look like this:



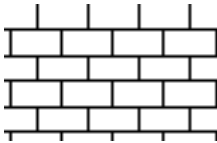
To create a hatch pattern with dashed lines

- 1 Open the *acad.pat* or *acadiso.pat* file in a text editor that saves in ASCII format (for example, Notepad).
- 2 Create a header line that includes an asterisk and a pattern name. The name of the hatch pattern is limited to 31 characters.
- 3 (Optional) To include a description in the header line, follow the pattern name with a comma and description text.
- 4 Create a descriptor line that includes
 - An angle at which the line is drawn
 - An *X,Y* origin point
 - A *delta-x* of any value if you want to offset alternating lines in the line family
 - A *delta-y* of any value
 - A value for a dash length
 - A value for a dot length
 - An optional second value for a different dash length
 - An optional second value for a different dot length

Hatch Patterns with Multiple Lines

Not all hatch patterns use origin points of 0,0. Complex hatch patterns can have an origin that passes through offsets from the origin and can have multiple members in the line family. In composing more complex patterns, you need to carefully specify the starting point, offsets, and dash pattern of each line family to form the hatch pattern correctly.

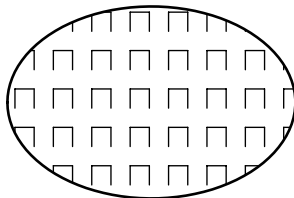
The default hatch pattern ANSI31 shown in the Boundary Hatch and Fill dialog box looks like this:



and is defined as follows with multiple lines describing the pattern:

```
*AR-B816, 8x16 Block elevation stretcher bond  
0, 0,0, 0,8  
90, 0,0, 8,8, 8,-8
```

The following figure illustrates a squared-off, inverted-U pattern (one line up, one over, and one down). The pattern repeats every one unit, and each unit is 0.5 high and wide.



This pattern would be defined as follows:

```
*IUS, Inverted U's  
90, 0,0, 0,1, .5,-.5  
0, 0,.5, 0,1, .5,-.5  
270, .5,.5, 0,1, .5,-.5
```

The first line (the up bar) is a simple dashed line with 0,0 origin. The second line (the top bar) should begin at the end of the up bar, so its origin is 0,.5. The third line (the down bar) must start at the end of the top bar, which is at .5,.5 for the first instance of the pattern, so its origin is at this point. The third line of the pattern could be the following:

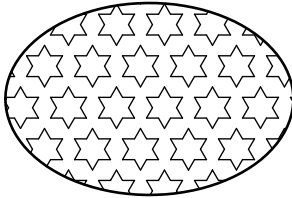
```
90, .5,0, 0,1, .5,-.5
```

or

```
270, .5,1, 0,1, -.5,.5
```

The dashed pattern starts at the origin points and continues in the vector direction given by the angle specification. Therefore, two dashed-line families that are opposed 180 degrees are not alike. Two solid-line families are alike.

The following pattern creates six-pointed stars.



This example can help you refine your skills at pattern definition. (Hint: 0.866 is the sine of 60 degrees.)

The following is the AutoCAD definition of this pattern:

```
*STARS,Star of David
0, 0,0, 0,.866, .5,-.5
60, 0,0, 0,.866, .5,-.5
120, .25,.433, 0,.866, .5,-.5
```

To create a hatch pattern with multiple lines

- 1 Open the *acad.pat* or *acadiso.pat* file in a text editor that saves in ASCII format (for example, Notepad).
- 2 Create a header line that includes an asterisk and a pattern name. The name of the hatch pattern is limited to 31 characters.
- 3 (Optional) To include a description in the header line, follow the pattern name with a comma and description text.

- 4 Create a descriptor line that includes
 - An angle at which the line is drawn
 - An X,Y origin point
 - A *delta-x* of any value if you want to offset alternating lines in the line family
 - A *delta-y* of any value
 - A value for a dash length
 - A value for a dot length
 - An optional second value for a different dash length
 - An optional second value for a different dot length
- 5 Create a second line including all the parameters in the previous step.
- 6 (Optional) Create additional lines to complete the multiple-line hatch pattern.

The Menu File

4

The menu file is a TXT file that defines most of the user interface. You can modify the menu file or create new menu files to add commands or macros to menus (including shortcut menus, image tile menus, and tablet menus) and to toolbars, assign commands to buttons on your pointing device, and create and modify shortcut keys.

In this chapter

- Overview of Menu Files
- Load and Unload Menu Files
- Create Menu Macros
- Use AutoLISP in Menu Macros
- Customize Buttons on a Pointing Device
- Create Pull-Down and Shortcut Menus
- Customize Toolbars
- Create Image Tile Menus
- Create Screen Menus
- Create Tablet Menus
- Create Status Line Help Messages
- Create Shortcut Keys

Overview of Menu Files

The menu file is an ASCII TXT file consisting of sections that define the function of each part of the user interface except the command line, for example, pull-down menus, toolbars, and the buttons on a pointing device.

The default menu file is *acad.mnu*. The *acad.mns* file is the same as *acad.mnu* but does not include comments or special formatting. AutoCAD® modifies this file each time the contents of the menu file change. You can open *acad.mns* in Notepad to see what a complete menu file looks like. To open the menu file, on the Tools menu, click Customize ► Edit Custom Files ► Current Menu.

You can create or modify menu files to

- Add or change menus (including shortcut menus, image tile menus, and tablet menus) and toolbars
- Assign commands to buttons on your pointing device
- Create and modify shortcut keys
- Add tooltips
- Provide Help text on the status line

To add a new menu, for example, you can modify the appropriate section of *acad.mnu* and save it under a new name, or you can create a new menu file.

In the following example, a drawing project requires frequent use of circles with a radius of 1, 2, or 3 units. To work more efficiently, you can create a menu file that defines a pull-down menu with three options, each of which draws a circle with a different radius. To create the menu file, you type the lines into Notepad (or any other text editor that saves in ASCII format) and then save the file with an *.mnu* extension. The menu file in this example could be named *circles.mnu*.

```
***MENUGROUP=CIRCLES
***POP13
//Creates circles with radii 1, 2, 3
Circle-1 [Radius-1 ]^C^C_circle \1
Circle-2 [Radius-2 ]^C^C_circle \2
Circle-3 [Radius-3 ]^C^C_circle \3
```

The first line in a menu file is always the Menugroup section, which provides a unique name for the menu. In the example, the Menugroup name is Circles. The Menugroup name does not need to be the same as the file name.

The second line in the example is a section label. This menu is a pull-down menu and therefore uses a Pop section with a number from 1 through 16 (POP13). For more information about pull-down and shortcut menus, see “Create Pull-Down and Shortcut Menus” on page 70.

The third line, preceded by two slashes (//), is an optional comment line, which is ignored when the menu is compiled. You can use comments in menu files for copyright notices, documentation, or notes.

The next three lines define the items on the menu. In the first menu item, circle-1 is the name tag assigned to the menu item. The text in brackets, [Radius-1], is the menu item label, which defines what is displayed on the menu for this item. The remainder is the menu macro, which uses AutoCAD commands and special characters to draw a circle with a radius of 1 unit. For information about creating menu macros, see “Create Menu Macros” on page 54.

In order to use the new menu in AutoCAD, you load the menu file, *circles.mnu*, with the MENULOAD command. For more information about loading menu files, see “Load and Unload Menu Files” on page 50.

Menu File Structure

Menu files are divided into sections. The Menugroup section is always first, and it assigns a unique menu group name to the menu file. A menu group name is a string of up to 32 alphanumeric characters with no spaces or punctuation marks.

The subsequent sections define specific areas of the AutoCAD interface and contain menu items, which usually consist of a name tag, a label, and a menu macro. For information specific to each section, see the topic for that section.

Menu file sections are identified by section labels that use the format *****section_name**. The multiple Buttons, Aux, Pop, and Tablet sections are numbered, for example, *****POP5**.

Section labels and associated elements in the user interface	
Section label	User interface area
***MENUGROUP	Menu group name
***BUTTONSn	Pointing-device button menu
***AUXn	System pointing device menu
***POPn	Pull-down and shortcut menus
***TOOLBARS	Toolbar definitions

Section labels and associated elements in the user interface (continued)

Section label	User interface area
***IMAGE	Image tile menus
***SCREEN	Screen menus
***TABLET _n	Tablet menus
***HELPSTRINGS	Text that is displayed in the status bar when a pull-down or shortcut menu item is highlighted, or when the cursor is over a toolbar button
***ACCELERATORS	Shortcut (or accelerator) key definitions

A menu file does not need to include every possible menu section. It is recommended that you create small menu files that can be loaded and unloaded on demand (with `MENULOAD` and `MENUUNLOAD`). Working with smaller files gives you better control of your system resources and makes customization easier.

Menu Items

The syntax that you use to create menu items is the same for all menu sections that use menu items. Each menu item can consist of a name tag, a label, and a menu macro. (Some sections do not use name tags, and some do not use labels.)

Name tag	Identifies the menu item. A menu item name tag is a string of alphanumeric and underscore (_) characters that uniquely identifies an item within a menu group.
Label	Defines what is displayed or presented to the user. The label is contained within square brackets ([and]).
Menu macro	Defines the action the menu item performs. Menu macros also define, for example, the appearance and location of toolbar buttons. Menu macros can be simple recordings of keystrokes that accomplish a task or a complex combination of commands and programming code.

A menu item normally resides on one line of the menu file and has the following format.

```
name_tag label menu_macro
```

In the following example from a Pop section, `ID_Quit` is the name tag. The label, `[Exit]`, displays Exit in the menu. When this menu item is selected, the menu macro, `^C^C_quit`, cancels any running commands and starts the QUIT command.

```
ID_Quit [Exit]^C^C_quit
```

Menu Item Labels

The format and use of menu item labels differ for each menu section. Menu sections that have no interface for displaying information (for example, the Buttons, Aux, and Tablet sections) do not require labels; however, labels can be used for internal notes in these sections. The following table describes how menu item labels are used in different sections of the menu file.

Use of menu section labels	
Menu section	Use of label
POP <i>n</i>	Defines the content and format of pull-down and shortcut menus
TOOLBARS	Defines the toolbar name, status (floating or docked and hidden or visible), and position; also defines each button and its properties
IMAGE	Defines the text and image displayed in the image tile menus
SCREEN	Defines the text displayed in the screen menus
HELPSTRINGS	Defines the status line Help related to menu items in the Pop and Toolbars sections
ACCELERATORS	Associates keyboard action with menu macros

Menu Macros

A menu macro defines the action that results when a menu item is chosen. You can use commands, special characters, and DIESEL or AutoLISP programming code to create a menu macro. If you intend to include a command in a menu item, you must know the sequence of prompts and the default options for each.

Note As AutoCAD is revised and enhanced, the sequence of prompts for various commands (and sometimes even the command names) might change. Therefore, your custom menus might require minor changes when you upgrade to a new release of AutoCAD.

See Also

- “Load and Unload Menu Files” on page 50
- “Create Menu Macros” on page 54
- “Create Pull-Down and Shortcut Menus” on page 70
- “Customize Toolbars” on page 82
- “Create Image Tile Menus” on page 87
- “Create Tablet Menus” on page 96
- “Create Status Line Help Messages” on page 97
- “Create Shortcut Keys” on page 98

Load and Unload Menu Files

Before you can use a menu file, it must be loaded into the program.

The *base menu* is automatically loaded when you start AutoCAD. In AutoCAD, the default base menu file is *acad.mnu*. If you modify the default menu or create a new menu file that you want to use as the base menu, you use MENU to load it. When you start AutoCAD again, the new base menu is automatically loaded.

The term *partial menu* refers to any menu file loaded after the base menu. You can use MENULOAD and MENUUNLOAD to load and unload partial menus as you need them during an AutoCAD session.

Any menu file can serve as a base menu or a partial menu, but it is recommended that you use a menu file that includes most sections as the base file and load additional smaller menu files as needed.

Load Menu Files

You can use MENULOAD and MENUUNLOAD to load and unload partial menus and add or remove individual pull-down menus from the menu bar.

AutoCAD stores the name of the last loaded base menu in the system registry. This name is also saved with the drawing, but it is used only for backwards compatibility. When you start AutoCAD, the last base menu used is loaded.

Change or Remove Menus

Frequent changes to the contents of a menu bar can be confusing. It is not recommended that you change the state of the menu bar visually except on explicit request. For example, if someone wants to unload an application, menus referenced specifically by that application could be removed as well.

To completely reinitialize the menu, remove all partial menus that are currently loaded by executing `MENULOAD` and selecting **Replace All** in the **Menu Customization** dialog box. This procedure removes all partial menus as well as their associated tag definitions and is equivalent to specifying a new menu file on the **Files** tab of the **Options** dialog box.

Restore or Alternate Menus

You can use a customized menu for some tasks while keeping the standard menu easily available. To load your custom menu, on the **System** tab of the **Options** dialog box, enter the custom menu name next to **Menu File**.

When you use `MENULOAD` or `MENUUNLOAD` to alter the loaded menus or customize the menu bar with **Pop** and **Toolbar** menus, the changes are saved to the registry. The next time you start AutoCAD, the menus that were loaded last and the menu bar configuration are restored. You can load and unload up to 8 partial menus and up to 16 **Pop** menus.

Work with Menu File Types

When you edit or create an MNU file, the next time it is loaded, AutoCAD compiles it and generates certain files. The term *menu file* is used to refer to any of the group of files that work together to define the user interface, as described in the following table. When you load or unload a menu file, the **Select Menu File** dialog box may list MNU, MNC, or MNS files. You can select any of these files to load the menu you want.

The menu file types and their origins are described in the table.

Menu file types	
File type	Description
MNU	Original ASCII menu file, the file you normally edit or create.
MNC	Compiled menu file; a binary file that contains the command strings and menu syntax that define the functionality and appearance of the menu or other interface element. AutoCAD compiles this file when you load an MNU file for the first time.
MNR	Menu resource file; a binary file that contains the bitmaps used by the menu or other interface element. AutoCAD generates this file each time it compiles an MNC file.
MNS	Source menu file; an ASCII file that is the same as the MNU file but does not include comments or special formatting. AutoCAD modifies this file each time the contents of the menu file change.
MNT	Menu resource file. This file is generated only when the MNR file is unavailable, for example, read-only.

Menu file types (continued)

File type	Description
MNL	Menu LISP file; contains AutoLISP expressions that are used by the menu file. This file is loaded into memory when a menu file with the same file name is being loaded.

AutoCAD finds and loads the specified file according to the following sequence. This same sequence is used when AutoCAD loads a new menu.

- 1 AutoCAD looks for a menu source (MNS) file of the given name, following the library search procedure.
 - If an MNS file is found, AutoCAD looks for a compiled menu (MNC) file of the same name in the same directory. If AutoCAD finds a matching MNC file with the same or later date and time as the MNS file, it loads the MNC file. Otherwise, AutoCAD compiles the MNS file, generating a new MNC file in the same directory, and loads that MNC file.
 - If an MNS file is not found, AutoCAD looks for a compiled menu (MNC) file of the given name, following the library search procedure. If AutoCAD finds the MNC file, it loads that file.
 - If neither an MNS nor an MNC file is found, AutoCAD searches the library path for a menu template (MNU) file of the given name. If this file is found, AutoCAD compiles an MNC file, generates an MNS file, and then loads the MNC file.
 - If no files of the given name are found, AutoCAD displays an error message and prompts you for another menu file name.
- 2 After finding, compiling, and loading the MNC file, AutoCAD looks for a menu LISP (MNL) file, using the library search procedure. If AutoCAD finds this file, it evaluates the AutoLISP expressions within that file.

The *acad.mnl* file contains AutoLISP code used by the standard menu file, *acad.mnu*. The *acad.mnl* file is loaded each time the *acad.mnu* file is loaded.

Each time AutoCAD compiles an MNC file it generates a menu resource (MNR) file, which contains the bitmaps used by the menu, and an MNS file, an ASCII file that is initially the same as the MNU file (without comments or special formatting). The MNS file is modified by AutoCAD each time you change the contents of the menu file through the interface (for example, when you use CUSTOMIZE to modify the contents of a toolbar).

Although the initial positioning of the toolbars is defined in the MNU or MNS file, changes to the show/hide and docked/floating status or changes to the toolbar positions are recorded in the system registry. After an MNS file

has been created, it is used as the source for generating future MNC and MNR files. If you modify the MNU file after an MNS file has been generated, you must use the **OPTIONS** command to explicitly load the MNU file so that AutoCAD will generate new menu files and your changes will be recognized.

Note If you use the interface to modify the toolbars, you should cut and paste the modified portions of the MNS file to the MNU file before deleting the MNS file.

To load a partial menu

- 1 At the Command prompt, enter **menuload**.
- 2 In the Menu Customization dialog box, Menu Groups tab, enter the menu file name (or click Browse to select the file in the Select Menu File dialog box).
- 3 Click Load.
- 4 If the menu group includes pull-down menus, click the Menu Bar tab.
- 5 Under Menu Group, select the menu group you loaded.
On the left, Menus lists the pull-down menus in the selected menu group. On the right, Menu Bar lists the menus currently displayed on the menu bar in order from left to right.
- 6 In the Menus list, select the menu you want to place on the menu bar.
- 7 In the Menu Bar list, select a menu and click Insert to place the new menu above the selected menu in the list.
If no menu is selected in the Menu Bar list, the menu selected in the Menus list is inserted at the top of the Menu Bar list.
- 8 Click Close.

To unload a partial menu

- 1 At the Command prompt, enter **menuunload**.
- 2 In the Menu Customization dialog box, Menu Groups tab, select the menu file that contains the menus you want to unload.
- 3 Click Unload.
- 4 Click Close.

To load the base menu

- 1 At the Command prompt, enter **menu**.
- 2 In the Select Menu File dialog box, select the menu file you want to load.
- 3 Click Open.

Create Menu Macros

An item in the MNU file can have three parts: name tag, label, and menu macro. The macro defines the action to be executed when a user chooses the menu item.

Overview of Menu Macros

The macro in a menu item can be as simple as a command:

```
Line [Line]line
```

The name tag is `Line`, the label is `[Line]`, and the menu macro is `line`, which starts the `LINE` command.

In the next menu item example, the menu macro, `^C^C_circle \1`, draws a circle with a radius of 1 unit.

```
Circle-1 [Radius-1]^C^C_circle \1
```

- The special characters `^C^C` cancel any running commands.
- The special character underscore (`_`) automatically translates the command that follows into other languages.
- The entry `circle` starts the `CIRCLE` command.
- The special character backslash (`\`) creates a pause for the user to specify the center point.
- The entry `1` responds to the prompt for the radius.

For a list of special characters that you can use in menu macros, see “Use Special Control Characters in Menu Macros” on page 58.

In addition to commands and special characters, you can use DIESEL (Direct Interpretively Evaluated String Expression Language) and AutoLISP. See “DIESEL” on page 103.

Cancel Running Commands

To make sure that you have no AutoCAD commands currently in progress before executing a menu macro, use `^C^C` at the beginning of the menu macro. This is the same as pressing `ESC` twice. Although a single `^C` cancels most commands, `^C^C` is required to return to the Command prompt from a dimensioning command; therefore, it is good practice to use `^C^C`.

Terminate Menu Macros

Every character in a menu macro is significant, even a blank space.

When a menu item is selected, AutoCAD places a space at the end of the macro before processing the command sequence. AutoCAD processes the macro in the following menu item as though you had entered **line** and then pressed SPACEBAR to complete the command:

```
Line [Line]line
```

In some situations, macros require special terminators. Some commands, for example, TEXT, must be terminated by pressing ENTER, not SPACEBAR. Some commands require more than one space (or ENTER) to complete, but some text editors cannot create a line with trailing blanks.

Two special conventions get around these problems.

- A semicolon (;) in a menu macro issues ENTER.
- If a line ends with a control character, a backslash (\), a plus sign (+), or a semicolon (;), AutoCAD does not add a blank space after it.

If the menu item in the following example simply ended with the backslash (which pauses the macro for user input), it would fail to complete the ERASE command, because AutoCAD does not add a blank after the backslash. Therefore, the menu macro uses a semicolon (;) to issue ENTER after the user input.

```
Erase 1 [Erase 1]erase \;
```

Compare the following menu items:

```
UCS [UCS ]ucs  
UCS World [UCS W ]ucs ;
```

The first example enters **ucs** on the command line and presses SPACEBAR. The following prompt is displayed.

Origin/ZAxis/3point/Entity/View/X/Y/Z/Prev/Restore/Save/Del/?/ <World>:

The second example enters **ucs**, presses SPACEBAR, and presses ENTER, which accepts the default value, World.

Warning! As AutoCAD is revised and enhanced, the sequence of prompts for various commands (and sometimes even the command names) might change. Therefore, your custom menus might require minor changes when you upgrade to a new release.

Most menu macros work equally well in all sections of the menu file.

Suppress Echoes and Prompts in Menu Macros

Normally, characters that are read from a menu macro appear in the command window just as if you had entered them through the keyboard (that is, they are echoed). Prompts are displayed even if a menu item provides the responses. You can suppress these displays using the `MENUECHO` system variable. If echoes and prompts from menu item input are turned off, a `^P` in the menu item turns them on.

Create Long Menu Macros

If an item in the menu macro does not fit on one line, you can continue it on the next line. To do this, add a plus sign (+) as the last character of the line to be continued. The example below, which you might use to set initial conditions for a new drawing, continues onto a second line.

```
[Setup ]layer set ground-floor;;grid on; ... ;fill off;+
limits 0,0 12,9;status
```

Menu items can continue on as many lines as necessary.

Note Line breaks are not preserved when AutoCAD creates the MNS file.

See Also

- “Use Special Control Characters in Menu Macros” on page 58
- “Pause for User Input in Menu Macros” on page 56
- “Provide International Language Support in Menu Macros” on page 58
- “Repeat Commands in Menu Macros” on page 61
- “Create Pull-Down and Shortcut Menus” on page 70
- “Customize Toolbars” on page 82
- “Create Image Tile Menus” on page 87
- “Create Screen Menus” on page 91
- “Create Tablet Menus” on page 96
- “Create Status Line Help Messages” on page 97
- “Create Shortcut Keys” on page 98
- “DIESEL” on page 103

Pause for User Input in Menu Macros

To accept input from the keyboard or the pointing device in the middle of a menu macro, place a backslash (\) at the point where you want input.

```
Circle-1 [Circle-1]circle \1
Layoff [Layoff ]-layer off \;
```

The menu macro in Circle-1 pauses for the user to specify the center point and then reads a radius of 1. Note that there is no space after the backslash. The Layoff menu item starts LAYER on the command line, enters the Off option, and pauses for the user to enter one layer name. Layoff then turns that layer off and exits LAYER. LAYER normally prompts for another operation and exits only if you press SPACEBAR or ENTER. In the menu macro, the semicolon (;) is used for ENTER.

Normally, a menu macro resumes after one user input; for example, one point location. Therefore, you cannot construct a menu macro that accepts a variable number of inputs (as in object selection) and then continues. However, an exception is made for SELECT: a backslash suspends the menu item until object selection has been completed. Consider the following menu item example:

```
Make Red [Make Red ]select \change previous ;properties color red ;
```

In this menu item, SELECT creates a selection set of one or more objects. The macro then starts CHANGE, references the selection set using the Previous option, and changes the color of all selected objects to red.

Note Because the backslash character (\) causes a menu macro to pause for user input, you cannot use a backslash for any other purpose in a menu macro. When specifying file directory paths, use a forward slash (/) as the path delimiter: for example, /direct/file.

The following circumstances delay resumption of a menu macro after a pause:

- If input of a point location is expected, object snap modes may be used before the point is specified.
- If X/Y/Z point filters are used, the menu item remains suspended until the entire point has been accumulated.
- For SELECT only, the menu macro does not resume until object selection has been completed.
- If the user responds with a transparent command, the suspended menu macro remains suspended until the transparent command is completed and the originally requested input is received.
- If the user responds by choosing another menu item (to supply options or to execute a transparent command), the original macro is suspended, and the newly selected item is processed to completion before the suspended macro is resumed.

Note When command input comes from a menu item, the settings of the PICKADD and PICKAUTO system variables are assumed to be 1 and 0, respectively. This preserves compatibility with previous releases of AutoCAD and makes customization easier because you are not required to check the settings of these variables.

Provide International Language Support in Menu Macros

If you develop menu files that can be used with a non-English-language version of AutoCAD, the standard AutoCAD commands and options are translated automatically if you precede each command or option with the underscore character (_).

The following example shows a portion of a Pop menu.

```
[ ->Arc]
[ 3-point]^C^C_arc
[ Start, Cen, End]^C^C_arc;\_c
[ Start, Cen, Angle]^C^C_arc;\_c;\_a
[ Start, Cen, Length]^C^C_arc;\_c;\_l
[ Start, End, Angle]^C^C_arc;\_e;\_a
[ Start, End, Radius]^C^C_arc;\_e;\_r
```

In the example, the underscore precedes every command or an option used in the menu macros.

Use Special Control Characters in Menu Macros

Special characters, including control characters, can be used in menu macros. In a menu macro, the caret (^) maps to the CTRL key on the keyboard. You can combine the caret with another character to construct menu macros that do such things as turn the grid on and off (^G) or cancel a command (^c). Because brackets ([and]) identify menu labels, they cannot be used in menu macros.

```
[ GridFlip]^G
[ *Cancel*]^C
```

The nonalphabetic control characters are as follows:

```
^@ (ASCII code 0)
^[ (ASCII code 27)
^\ (ASCII code 28)
^] (ASCII code 29)
^^ (ASCII code 30)
^_ (ASCII code 31)
```

The macro in the Address menu item below uses the backslash (\) to pause for user input and the semicolon (;) for ENTER.

```
Address [Address ]text \.4 0 DRAFT Inc;;;Main St.;;;City, State;
```

The macro starts TEXT, pauses for the user to specify a start point, and then enters the address on three lines. In the triple semicolon (;;;), the first semicolon ends the text string, the second repeats TEXT, and the third accepts the default placement below the previous line.

You may want a menu macro to enter one or more characters but not submit them as final input. For example, you could create a series of menu macros to act as a numeric keypad.

```
[1]1x^H
[2]2x^H
[3]3x^H
```

When you choose one of these items, the appropriate digit is entered. Another character follows (the letter *x* in this case), and that character is removed by ^H. (CTRL+H is the ASCII code for a BACKSPACE.) Each of these menu items ends with a control character, and AutoCAD does not add a space or ENTER to such items. Thus, you can choose [2], [2], [3], [1] to construct the input 2231. Press ENTER to enter the completed number.

Menu macros use the special characters listed in the following table. Brackets ([and]) identify menu labels and cannot be used in menu macros.

Special characters used in menu macros

Character	Description
;	Issues ENTER
^M	Issues ENTER
^I	Issues TAB
[blank space]	Enters a space; a blank space between command sequences in a menu item is equivalent to pressing the SPACEBAR
\	Pauses for user input (cannot be used in the Accelerators section)
_	Translates AutoCAD commands and options that follow
+	Continues the menu macro to the next line (if last character)
=*	Displays the current top-level pull-down, shortcut, or image menu
*^C^C	Prefix for a repeating item
\$	Loads a menu section or introduces a conditional DIESEL macro expression (\$M=)
^B	Turns Snap on or off (CTRL + B)
^C	Cancels a command (ESC)
^D	Turns Coords on or off (CTRL + D)
^E	Sets the next isometric plane (CTRL + E)
^G	Turns Grid on or off (CTRL + G)
^H	Issues BACKSPACE
^O	Turns Ortho on or off
^P	Turns MENUCHO on or off
^Q	Echoes all prompts, status listings, and input to the printer (CTRL + Q)
^T	Turns tablet on or off (CTRL + T)
^V	Changes the current viewport
^Z	Null character that suppresses the automatic addition of SPACEBAR at the end of a menu item

See Also

“ASCII Codes” on page 149

Repeat Commands in Menu Macros

Once you have selected a command, you might want to use it several times before moving on to another command. In a menu macro, you can repeat a command until you choose another command. You cannot use this feature to choose options.

If a menu macro begins with `*^C^c` immediately following the item label, the macro is saved in memory. Subsequent Command prompts are answered by that macro until it is terminated by ESC or by selection of another menu item.

Note Do not use `^C` (Cancel) within a menu macro that begins with the string `*^C^C`; this cancels the repetition.

The menu macros in the following examples repeat the commands:

```
[Move ]*^C^Cmove Single  
[Copy ]*^C^Ccopy Single  
[Erase ]*^C^Cerase Single  
[Stretch]*^C^Cstretch Single Crossing  
[Rotate ]*^C^Crotate Single  
[Scale ]*^C^Cscale Single
```

Each of the macros in the example starts the command and prompts the user to select one object. Any other prompts necessary to complete the command are displayed, and then the command ends and starts again. For information about `single` and `single crossing`, see “Use Single Object Selection Mode in Menu Macros” on page 61.

Command repetition cannot be used in menu macros for image tile menus.

Use Single Object Selection Mode in Menu Macros

Single Object Selection mode cancels the normal repetition of the Select Objects prompt in editing commands. After you select one object and respond to any other prompts, the command ends. Consider the menu macro in the following example:

```
[Erase]*^C^Cerase single
```

This macro terminates the current command and starts ERASE in Single Object Selection mode. After you choose this menu item, you either select a single object to be stretched or click a blank area in the drawing and specify window selection. Any objects selected in this way are erased, and the menu item is repeated (due to the leading asterisk) so that you can erase additional objects.

Use Menu Macros to Swap Menus

You can use menu macros to replace the contents of an active Buttons, Aux, Pop, Screen, or Tablet menu section. The new menu content can be from another section or submenu in the base menu, or it can come from a partial menu. Menu-swapping information that is related to each menu section is included with the description of that menu section.

You can swap menus only of the same type—one Aux for another, one Pop for another, and so on. Trying to swap between types may result in unpredictable and undesired behavior. However, within a given type, you can swap any menu for any other menu. Swapping can lead to some strange behavior for Tablet menus, because they typically do not all have the same number of macros.

Use the following syntax in a menu macro to swap menus:

`$section=menugroup.menuname`

These are the descriptions:

\$	Loads a menu section.
section	Specifies the menu section. Valid names are A1–A4 for Aux menus 1 through 4 B1–B4 for Buttons menus 1 through 4 P0–P16 for Pop menus 0 through 16 I for the Image menu S for the Screen menu T1–T4 for Tablet menus 1 through 4
menugroup	Specifies the menu group that <i>menuname</i> is a member of (not necessary if <i>menuname</i> is in the base menu).
menuname	Specifies which section or submenu to insert. It is the main label or alias for the section to load.

The following menu items illustrate submenu referencing:

```
$$=PARTS
$T1=EDITCMDS
```

You can activate the submenu mechanism in the middle of a command without interrupting it. For example, the following command strings are equivalent:

```
$S=ARCSTUFF ARC  
ARC $S=ARCSTUFF
```

Each menu item starts the ARC command, switches to the ARCSTUFF screen submenu, and awaits the entry of arc parameters. A space must follow the submenu reference to separate it from subsequent commands in the menu item.

A Pop menu can be present either in the menu bar or on the active shortcut menu but not both.

Use Conditional Expressions in Menu Macros

You can use the `$M=` command within a menu macro to introduce macro expressions written in DIESEL (Direct Interpretively Evaluated String Expression Language). The format is

```
$M=expression
```

Introducing the macro with `$M=` tells AutoCAD to evaluate the following string as a DIESEL expression, and that *expression* is the DIESEL expression.

```
[Fillflip]FILLMODE $M=$((-1,$(getvar,fillmode))
```

`Fillflip` switches FILLMODE on and off by subtracting the current value of FILLMODE from 1 and returning the resulting value to the FILLMODE system variable. You can use this method to toggle system variables whose valid values are 1 or 0.

Termination of Macros That Contain Conditional Expressions

If you use the DIESEL string language to perform “if-then” tests, conditions might exist where you do not want the normal terminating space or semicolon (resulting in ENTER). If you add `^z` to the end of the menu macro, AutoCAD does not automatically add a space (ENTER) to the end of the macro expression. See “Overview of Menu Macros” on page 54 and “DIESEL” on page 103.

As with other control characters in menu items, the `^z` used here is a string composed of `^` (a caret) and `z` and is not equivalent to pressing CTRL+Z.

In the following examples, `^z` is used as a menu macro terminator.

```
[Model]^C^C$M=$((if,$(=,$(getvar,tilemode),0),$S=mview _mspace )^z  
[Paper]^C^C$M=$((if,$(=,$(getvar,tilemode),0),$S=mview _pspace )^z
```

If these menu macros did not end with `^z`, AutoCAD would automatically add a space (ENTER), repeating the last command entered.

See Also

“Overview of Menu Macros” on page 54

“Use Special Control Characters in Menu Macros” on page 58

“DIESEL” on page 103

Use AutoLISP in Menu Macros

You can use AutoLISP variables and expressions to create menu macros that perform complex tasks. To use AutoLISP efficiently in menu macros, you can place AutoLISP code in a separate MNL file. AutoCAD loads the MNL file when it loads a menu file with the same name.

AutoCAD accepts up to 255 characters of AutoLISP code in menu macros. To use more characters, break up the code into separate modules separated by semicolons (;) so that AutoCAD can read and execute the code in blocks.

Creating menu items that use AutoLISP is a more advanced way to use the AutoCAD custom menu feature. Carefully study the following examples and the information in the *AutoLISP Reference* and the *AutoLISP Developer's Guide* (on the Help menu, click Developer Help). Experimentation and practice will help you use this feature effectively.

Call a Menu Macro

To programmatically execute a Pop menu macro as though the user chose it, you can use the following syntax:

```
(menucmd "Gmenugroup.name_tag=|")
```

This works only if the Pop menu macro is part of a Pop menu that is on the AutoCAD menu bar and available for use. For more information about this syntax, see the *AutoLISP Reference*.

Preset Values

An application that uses block insertion presets could provide menu items like these:

```
[Set WINWID ]^C^C^P(setq WWID (getreal"Enter window width: ")) ^P  
[Set WALLTHK]^C^C^P(setq WTHK (getreal"Enter wall thickness: ")) ^P  
[Insert Window]^C^C_INSERT window XScale !WWID YScale !WTHK
```

This code inserts the block named window, scaling its *X* axis to the current window width and its *Y* axis to the current wall thickness. In this example, the actual values come from the user-defined AutoLISP symbols WINWID and WALLTHK. The rotation is up to the user to decide so that the window can be rotated in the wall.

Resize Grips

With the following menu items, grip size adjustment can be done on the fly:

```
[GRIP-up]^P(setvar"gripsize"(1+(getvar"gripsize")))(redraw)(princ)
[GRIP-dn]^P(setvar"gripsize"(1-(getvar"gripsize")))(redraw)(princ)
```

To add validity checking to these menu items, values less than 0 and greater than 255 cannot be used for the GRIPSIZE system variable.

Prompt for User Input

The following menu item prompts the user for two points and draws a rectangular polyline with the specified points as its corners.

```
[BOX ]^P(setq a (getpoint "Enter first corner: "));\+
(setq b (getpoint "Enter opposite corner: "));\+
pline !a (list (car a)(cadr b)) !b (list (car b)(cadr a)) c;^P
```

Customize Buttons on a Pointing Device

The Windows system pointing device uses the Aux menus in the MNU file; any other pointing device uses the Buttons menus.

Overview of Buttons and Aux Menus

The Buttons (**BUTONS_{*n*}) and Aux (**AUX_{*n*}) sections of the menu file are identical in format. The Windows system pointing device uses the Aux menus, and any other pointing device (for example, a puck or stylus for a digitizing tablet) uses the Buttons menus. The BUTONS₁ menu functions identically to the AUX₁ menu, and so on.

Buttons and Aux menu sections are only valid when the menu file is used as a base menu; Buttons and Aux sections are ignored in partial menus.

In the numbered Aux and Buttons sections of the menu file, each line represents a button. Your pointing device can recognize as many lines as it has assignable buttons. For example, in the `AUX1` and `BUTTONS1` sections, each line defines what happens when you click a particular button, and in the `AUX2` and `BUTTONS2` sections, each line defines what happens when you press `SHIFT` and click a particular button.

Buttons and associated menu sections	
Key/button sequence	Menu sections
Click	AUX1 and BUTTONS1
SHIFT + click	AUX2 and BUTTONS2
CTRL + click	AUX3 and BUTTONS3
CTRL + SHIFT + click	AUX4 and BUTTONS4

Although sections 1 through 4 are the only active sections, you can define additional sections and swap them into the active sections. See “Swap Buttons and Aux Menus” on page 67.

The following `AUX1` section example is similar to that found in the standard *acad.mnu* file:

```
***AUX1
;
^C^C
^B
^O
^G
^D
^E
^T
```

The first line after the menu section label, `***AUX1`, represents the next button after the pick button on your pointing device (button 2). The semicolon (;) assigns `ENTER` to button 2. The second line after the menu section label assigns `ESC` twice to button 3, and so on.

Note The first line after the menu section label `***AUX1` or `***BUTTONS1` is used only when the `SHORTCUTMENU` system variable is set to 0. If it is set to a value other than 0, the built-in menu is used. Similarly, the second line after the `***AUX1` or `***BUTTONS1` label is used only when the `MBUTTONPAN` system variable is set to 0.

You cannot reassign the pick button in the menu file. The pick button assignment is controlled by the operating system, or a device-specific configuration. The default pick button can be different on each pointing device, depending on the manufacturer.

Because labels in button menus are not displayed, you can use the labels as comments. The following example uses the label area to note the button number.

```
***AUX1
[button no.2];
[button no.3]$P0=*
[button no.4]^C^C
[button no.5]^B
```

The macro assigned to button number 3 in the example causes another menu to be displayed. It has the following format:

```
$Pn=*
```

The `$` is the special character code for loading a menu area; `Pn` specifies the Pop menu section; and `=*` displays what is currently loaded to the specified menu area.

Therefore, in the example from *acad.mnu*, clicking button number 3 displays the menu assigned to the P0 location. (The P0 menu location is the shortcut menu usually called from the Buttons or Aux menu. The P1 through P16 locations are left to right on the menu bar.) Typically, the Pop0 section of the menu file is assigned to the P0 location.

Each remaining line in that section assigns a command sequence to each subsequent button on the pointing device. For example, `^C^C` (ESC twice) is assigned to button 4, and `^B` (Snap mode toggle) is assigned to button 5.

See Also

“Create Menu Macros” on page 54

“Use Special Control Characters in Menu Macros” on page 58

Swap Buttons and Aux Menus

If necessary, you can swap the contents of the active Buttons and Aux menus with that of another menu section of the same type. In addition to the standard AUX1 through AUX4 (and BUTTONS1 through BUTTONS4) sections, you can define additional numbered Aux and Buttons sections for specific purposes.

Note Even though section labels such as `***AUXTEST` and `***BUTTONS1-2` are currently valid, numbered labels such as `***AUX10` or `***BUTTONS15` are preferred for their long-term compatibility.

These additional sections can use aliases with the syntax `**label`. The labels must come between the `***section` line and the first menu item line for that section. The alias `label` string can be any string. It does not need to contain any keyword. You can have as many aliases as you like for each section. The alias labels, as well as the section labels, can be used to identify the menu for swapping purposes. In the example below, `Aux12` has two aliases; the section label or either of the aliases is sufficient to identify the menu.

For example, if you want to change the standard right-click action so that AutoCAD displays a different shortcut menu, you can use the following menu syntax.

```
***AUX2
// Shift + button
$P0=SNAP $p0=*

***AUX4
// Control + Shift + button - Toggles to custom A2
$A2=CUSTOM_A2 $A4=CUSTOM_A4 ^P(princ ">> Custom A2 <<")(princ) ^P

***AUX12
**CUSTOM_A2
**MYPOP
// Shift + button - Displays the MYPOP menu
$P0=MYPOP $p0=*

***AUX14
**CUSTOM_A4
// Control + Shift + button - Toggles back to default A2
$A2=AUX2 $A4=AUX4 ^P(princ ">> Default A2 <<")(princ) ^P
```

Replacing the standard `Aux4` definition with the one shown in the example allows you to swap the content of the `A2` and `A4` menus. After reloading the `MNU` file, `CTRL+SHIFT+click` loads the contents of the menus defined by the alias `CUSTOM_A2` into the `A2` menu position and `CUSTOM_A4` into the `A4` position. Then, using AutoLISP, AutoCAD displays a message on the command line. Now when you `SHIFT+click`, the `MYPOP` menu is loaded into the `P0` menu position and is used as the shortcut menu. To return to the default `P0` menu, use `CTRL+SHIFT+click` (which now calls the `CUSTOM_A4` menu) to load the default `Aux2` menu back into the `A2` menu position and the `Aux4` menu back into the `A4` position.

You can also use the AutoLISP `menucmd` function to swap Aux and Buttons menus. Assuming that the previous examples are in an MNU file with a menugroup of `MYGROUP`, the following function call loads the `CUSTOM_A2` menu into the A2 menu position.

```
(menucmd "A2=mygroup.custom_a2")
```

Note Previous AutoCAD releases allowed the `**label` syntax after menu items within a menu section; these were known as submenus. The submenu syntax is still accepted by AutoCAD, but it is converted into a `***section` label in the MNS file. This syntax is not guaranteed to be valid in future releases. It is recommended that you change all `**label` submenus into `***section` labels. The `**alias` label is valid, provided that it occurs after a section label and before any menu items.

Accept Coordinate Entry in Button Menus

When you click one of the buttons on a multibutton pointing device, AutoCAD reads not only the button number but also the coordinate of the crosshairs at the time you click. By carefully constructing the macros in the Buttons and Aux sections of the menu file, you can choose to either ignore the coordinate or use it with the command activated by the button.

As described in “Pause for User Input in Menu Macros” on page 56, you can include a backslash (\) in a menu item to pause for user input. For the Aux and Buttons menus, the coordinate of the crosshairs is supplied as user input when the button is clicked. This occurs only for the first backslash in the menu item; if the item contains no backslashes, the crosshairs coordinate is not used. Consider the following menu items:

```
***AUX2  
line  
line \
```

The first button starts the `LINE` command and displays the Specify First Point prompt in the normal fashion. The second button also starts the `LINE` command, but AutoCAD uses the current crosshairs location at the Specify First Point prompt and displays the Specify Next Point prompt.

Create Pull-Down and Shortcut Menus

Pull-down menus are pulled down from the menu bar. Shortcut menus (also called context menus) are displayed at or near the crosshairs or cursor in the drawing window, in the text window or command window, or in toolbar areas. Both are defined in the Pop sections of the MNU file.

Overview of Pull-Down and Shortcut Menus

The pull-down and shortcut menus are displayed as cascading menus (also known as walking or hierarchical menus).

There are two types of shortcut menus; both are displayed at the cursor location. The context menus, displayed when you right-click, provide commands appropriate to your current activity or the location of the cursor on the screen. The object snap menu, displayed when you hold down SHIFT and right-click, provides object snaps and tracking options.

Pull-down menus are defined in the `***POP1` through `***POP499` menu sections, and shortcut menus are defined in the `***POP0` and `***POP500` through `***POP999` sections. A pull-down menu can contain up to 999 menu items. A shortcut menu can contain up to 499 menu items. Both limits include all menus in a hierarchy. If menu items in the menu file exceed these limits, AutoCAD ignores the extra items. If a pull-down or shortcut menu is longer than the available display space, it is truncated to fit.

Pull-down menus are always *pulled down* from the menu bar, but the shortcut menu is always displayed at or near the crosshairs or cursor in the graphics area, in the text window or command window, or in the toolbar areas. The syntax for both of these `POPn` menu sections is the same except that the shortcut menu title is not included in the menu bar. The shortcut menu title is not displayed at all (but you must still enter a dummy title).

Access to the shortcut menu is through the `$P0=* menu` command, which can be issued by another menu item (such as a `BUTTONSn` menu item) or by an AutoLISP or ObjectARX® program. While the shortcut menu is active, the menu bar is not available.

Create Separator Lines on Menus

To create a horizontal line between items displayed on a menu, use a menu item label containing two hyphens:

```
[ -- ]
```

The width of each pull-down and shortcut menu is determined by its widest label, and the separator line expands to fill the entire width of the menu. (A menu macro on a line with a separator line label is ignored.)

Create Cascading Submenus

Pull-down and shortcut menu item labels use special characters (such as →, ←, and <←) to control the hierarchy of cascading menus. These special characters indicate submenus and last items in submenus and can also terminate all parent menus. Each special character string must come first in a menu item label.

The special character → indicates that this item has a submenu, as in the following example:

```
[ →&Zoom ]
```

If you pull down the View menu and click Zoom or move the cursor to the right end of the item, the Zoom submenu is displayed.

The special character ← indicates that the item is the last item in a submenu, as in the following example:

```
[ ←&Extents ]
```

Special characters <← indicate that the item is the last item of a submenu and also of its parent menu, as in the following examples of some labels from the Modify menu:

```
[ &Modify ]  
[ →&Object ]  
[ →&Text ]  
[ <←&Justify ]
```

The first example is the label for the Modify menu. The label for Object uses a special character to indicate that it has a submenu. Text is part of the Object submenu and has a submenu of its own. Justify is the last item on the Text submenu and also ends the Modify menu.

The characters described in the following table are the only nonalphanumeric characters that can be used in a Pop section label. Nonalphanumeric characters not listed are reserved for future use as special menu characters.

Special characters for labels in Pop menu sections	
Character	Description
--	Expands to become a separator line in the pull-down and shortcut menus (when used with no other characters).
->	Indicates that the pull-down or shortcut menu item has a submenu.
<—	Indicates that the pull-down or shortcut menu item is the last item in the menu or submenu.
<—<—	Indicates that the pull-down or shortcut menu item is the last item in the submenu and terminates the parent menu. (One <— is required to terminate each parent menu.)
\$(	Enables the pull-down or shortcut menu item label to evaluate a DIESEL string macro if \$(are the first characters.
~	Makes a menu item unavailable.
! .	Marks a menu item with a check mark.
&	Placed directly before a character, specifies that character as the menu access key in a pull-down or shortcut menu label. For example, S&le displays S <u>a</u> mple (with the letter <i>a</i> underlined).
/c	Specifies the menu access key in a pull-down or shortcut menu label. For example, /aSample displays S <u>a</u> mple (with the letter <i>a</i> underlined).
\t	Pushes all label text to the right of these characters to the right side of the menu.

Create Pull-Down Menus

The POP1 through POP499 menu file sections define menus that are pulled down from the menu bar.

The following example illustrates the syntax that is used to create a pull-down menu.

```
***POP13
**MYTOOLS
M_Tools  [&MyTools]
M_Save   [&Save\tCtrl+S]^C^C_qsave
```

```

[--]
M_ShwbTB [Show MyToolbar]^C^C_-toolbar mytools s
M_HidTB [Hide MyToolbar]^C^C_-toolbar mytools h
[--]
M_EMenu [Edit MyMenu]^C^C^P(command"notepad"(findfile"my.mnu")) ^P
M_LMenu [Reload MyMenu]^C^C^P(command"_menu"my.mnu") ^P
[--]
M_EPgp [Edit PGP]^C^C^P(command"notepad"(findfile"acad.pgp")) ^P
M_LPgp [Reload PGP]^C^C_re-init 16

```

Each menu section can have one or more aliases that are defined by ****alias** labels following the *****POP_n** menu section label. In the previous example, ****MYTOOLS** is an alias for the POP13 menu. For additional information about menu aliases, see “Swap Button and Auxiliary Menus” on page 30.

Note The POP_n menu sections no longer support the ****submenu** syntax used in previous releases. The ****alias** syntax is valid, provided that it occurs after a section label and before any menu items.

For the POP1 through POP16 menu sections, AutoCAD constructs a menu bar containing the titles of those sections. If no POP1 through POP16 sections are defined, AutoCAD inserts default File and Edit menus.

Note If no active pull-down menus are defined (POP1 through POP16), the shortcut menu, POP0, does not function.

Pop menu sections numbered higher than POP16 and lower than POP500 are available to be inserted into the menu bar with MENULOAD or through the menu swapping process.

Define the Menu Bar Title

For pull-down menus, the first label defines the title that is displayed in the menu bar. The following example is the top portion of the POP2 pull-down menu section.

```

***POP2
ID_MnEdit [&Edit]
ID_U [&Undo\Ctrl+Z]_u
ID_Redo [&Redo\Ctrl+Y]^C^C_redo

```

On the first line after the section label, `***POP2`, the label `[&Edit]` causes Edit to be displayed as a menu bar title. The ampersand (&) preceding the letter *E* underlines that letter to indicate that it is the access key. The name tag associated with the menu title, `ID_MnEdit`, can be used to enable and disable this entire menu. (The line that defines the menu bar title cannot have a menu macro.)

To access a menu or a menu item from the keyboard, you hold down ALT and press the access key. Any letter in the label can be the access key, but an access key must be unique for its menu or submenu. For example, on the Modify menu in AutoCAD, *m* is used for Match Properties, so Mirror and Move must use other letters. The letter *t* can be used for both Trim and Text because Text is on the Object submenu.

See Also

“Swap Buttons and Aux Menus” on page 67

“Swap and Insert Pull-Down Menus” on page 80

Create Shortcut Menu

Shortcut menus are defined in the MNU file using the same syntax as pull-down menus. The `POP0` menu section defines the default Object Snap shortcut menu, and the menu sections from `POP500` through `POP999` are used for context shortcut menus.

AutoCAD references the context shortcut menus by their alias (for example, `**GRIPS`) and uses them in specific situations. The actual `POPn` number is not important, but the alias names must follow the proper naming conventions in order to be used. The following aliases are reserved for use by AutoCAD:

GRIPS	The content of this menu defines the Hot Grip shortcut menu (right-click in the drawing area while a grip on an object is selected).
CMDEFAULT	The content of this menu defines the Default mode shortcut menu (right-click in the drawing area while no command is active and no objects are selected).
CMEDIT	The content of this menu defines the Edit mode shortcut menu (right-click in the drawing area while one or more objects are selected, no grips are selected, and no command is active).

CMCOMMAND The content of this menu defines the Command mode menu (right-click in the drawing area while a command is active). In addition to the content of the **CMCOMMAND** menu, the command line options (keywords within the square brackets) are inserted into this menu.

Context-Sensitive Shortcut Menus

The **CMEDIT** and **CMCOMMAND** shortcut menus can be made context-sensitive. In addition to the content of the **CMEDIT** menu, the appropriate object menu (if it exists) is inserted into this menu when one or more of a specific object type are selected. Object menus use the following naming convention:

OBJECT_*objectname* OR **OBJECTS_***objectname*.

If a single object is selected, the **OBJECT_***objectname* menu is used, and if more than one of the same object is selected, the **OBJECTS_***objectname* menu is used. If no **OBJECT_***objectname* is available, AutoCAD uses the **OBJECTS_***objectname* menu (if it exists).

The object name is the drawing interchange format (DXF)[™] name of the object in all cases except the insert object. To differentiate between a block insertion and an xref, use the names **BLOCKREF** and **XREF**.

The following AutoLISP[®] code defines the command **OTYPE**, which reports the selected object's DXF name.

```
(defun C:OTYPE()  
  (cdr (assoc 0 (entget (car (entsel))))))
```

For example, to support an object-specific shortcut menu item for one or more selected block references, insert the following:

```
***POP512  
**OBJECTS_BLOCKREF  
[shortcut menu for block objects]  
ID_BLOCK [Explode] ^C^C_explode
```

Like the **CMEDIT** menu, the **CMCOMMAND** menu can have context-sensitive information added to it. Any menu named **COMMAND_***commandname* is appended to the **CMCOMMAND** menu. The text of *commandname* can be any valid AutoCAD command, including any custom-defined or third-party commands.

To make this work with a hyphen-prefixed command (such as **-INSERT**), you need to name the menu **COMMAND_-INSERT**.

Control Display of Menu Item Labels

You can control the way that item labels are displayed. Display them as gray, making them unavailable to the user, or mark them with a check mark.

Labels can also contain DIESEL string expressions to modify the contents of the label. This can disable, mark, or interactively change the text of the displayed label. See “DIESEL Expressions in Menu Macros” on page 108.

When disabling and marking menu item labels, be sure to use an appropriate technique that keeps track of changes that affect the state of the label.

Disable Menu Item Labels

To gray out a menu item, begin the menu item label with a tilde (~). By convention, this indicates that the item is disabled, that is, not available. Any commands associated with the item are not issued, and any submenus are inaccessible.

For example, the following menu items are disabled.

```
[~Line]
[~->Pline]
```

Menu item labels can contain DIESEL string expressions that conditionally disable or enable menu item labels each time they are displayed. For example, the DIESEL string expression within the following menu item label disables the item while a command is active.

```
[$(if,$(getvar,cmdactive),~)MOVE]^C^C_move
```

The AutoLISP `menucmd` function can be used to disable and enable items from a menu macro or application. For examples, see “Reference Pull-Down or Shortcut Menus”.

Mark Menu Item Labels

You can mark a menu item label with a leading check mark by including an exclamation point and a period (!.). Marking a menu item does not restrict your ability to choose the item, although a marked item can be disabled.

In the following example, the Line menu item is marked with a check.

```
[!.Line]
```

Menu item labels can contain DIESEL string expressions that conditionally mark menu item labels each time they are displayed. The following example places a check mark to the left of menu labels whose related system variables are currently enabled.

```
[$(if,$(getvar,orthomode),!.)Ortho]^O
[$(if,$(getvar,snapmode),!.)Snap]^B
[$(if,$(getvar,gridmode),!.)Grid]^G
```

The AutoLISP `menucmd` function can be used to mark labels from a menu macro or application. For examples, see “Reference Pull-Down or Shortcut Menus”.

Simultaneously Disable and Mark Menu Item Labels

You can mark and disable menu items at the same time. This is the format:

```
[~!. labeltext]
```

or

```
[!.~ labeltext]
```

The ~ is the special character code to disable a menu item and !. is the special character code to mark a menu item.

In the example that follows, the Line menu item is disabled and marked with a check mark. As with the previous examples, a DIESEL expression can be used to simultaneously disable and mark a menu item label.

```
[~!.Line]
```

See Also

“DIESEL Expressions in Menu Macros” on page 108

Reference Pull-Down or Shortcut Menus

Using a method similar to that used to activate regular submenus, you can activate or deactivate a pull-down or pop-up submenu.

The two methods for referencing a pull-down or shortcut menu are relative and absolute. *Relative referencing* uses the menu group and name tag; *absolute referencing* uses the absolute position of the menu item in the menu hierarchy. The relative referencing method is recommended because of its dynamic nature, which allows it to function properly regardless of the current state of the menu.

Relative Referencing of Pull-Down and Shortcut Menu Items

To reference a pull-down or shortcut menu item based on its menu group and name tag, use the AutoLISP **menucmd** function. The following syntax references a menu item based on its name tag.

```
(menucmd "Gmenugroup.name_tag=value")
```

The following example disables the menu item ID_Line in the sample menu group. It works regardless of the menu item's location in the menu.

```
[Disable Line](menucmd "Gsample.ID_Line=~")
```

If the author of a partial menu is aware of the contents of the base menu, the syntax of a menu item can reference a tag from the base file. An excerpt from the base file *acad.mnu* might look something like the following:

```
***MENUGROUP=ACAD  
***POP0 (and so forth...)
```

```

.....
***POP6
ID_MnHelp    [Help]
ID_Contents  [Contents]^C^C_HELP
ID_About     [About]^C^C_ABOUT

```

A menu item in a partial menu can be modified to have an additional menu item that references the tag in the base menu.

```

***POP2
[Title2]
[Disable Help Contents]^P(menucmd "Gacad.ID_Contents=~") ^P

```

In this manner, multiple partial menu files and specific base files can work together. AutoCAD enforces strict menu group definition so that no two menus can define the same menu group. Attempts to load a menu with a conflicting menu group results in cancellation of the MENULOAD request.

Absolute Referencing of Pull-Down and Shortcut Menu Items

In addition to referencing a menu item, you can activate or deactivate a menu item with the `Pn=xxx` syntax. This is the format:

```
Pn.i=xxx
```

The `Pn` specifies the active `POPn` menu section (0 through 16 are valid values); `i` specifies the menu item number; and `xxx` (if present), specifies a string defining the action.

The following example uses the AutoLISP `menucmd` function to reference a pull-down or shortcut menu item. Because AutoCAD menu files are dynamic (through the loading of partial menus), the following syntax won't work in all cases.

```
[Disable Line Old Way](menucmd "Pl.2=~")
```

This syntax relies on the location of the menu item and does not work if a new item was inserted into the `POP1` section by the menu author or if a new pull-down menu is inserted before `POP1` by the MENULOAD command.

You can use the `Pn=xxx` syntax from a menu macro if it follows the `$` command. The following example disables item 4 in the `POP3` section.

```
$P3.4=~
```

The following example adds a check mark to item 1 in the `POP7` section.

```
$P7.1=!
```

The following example removes any disabling or mark character from item 1 in the `POP7` section.

```
$P7.1=
```

Menu item numbering is consecutive without regard to the hierarchy of the menu file; item 1 is the first item following the title.

```
***POP5
[Assist   ]                               Title
[Help!    ] ' ?                           Item 1
[Cancel   ] ^C^C^C                         Item 2
[--]      ]                               Item 3
[Undo     ] ^C^C_U                         Item 4
[Redo     ] ^C^C_redo                     Item 5
[--]      ]                               Item 6
[->Osnap  ]                               Item 7
[Center   ] center                       Item 8
```

To make it easy for an item to address itself without regard to location in the menu hierarchy, use these forms:

<code>\$P@. @=xxx</code>	References the current or most recently chosen menu item.
<code>\$P@. n=xxx</code>	References item <i>n</i> in the current or most recently chosen menu.

AutoLISP Access to Label Status

The AutoLISP **menucmd** function accepts `$Pn=xxx` command strings but without the leading `$`. For these functions, the `xxx` portion of the command string can have special values.

<code>Pn.i=?</code>	Returns the current disabled and marked status for the specified item as a string (for example, <code>~</code> for a disabled item, <code>!</code> for an item with a check mark, and <code>"</code> for an item that is neither grayed out nor marked).
<code>Pn.i=#?</code>	Returns the same type of string as described for <code>Pn.i=?</code> , but with the <code>Pn.i=</code> prefix. This is useful in conjunction with the <code>@</code> forms, because the actual menu and item number are returned.

For example, if the fifth item in the `POP6` section is disabled, the following **menucmd** code returns the following string values.

```
(menucmd "P6.5=?")      returns  "~"
(menucmd "P6.5=#?")     returns  "P6.5=~"
```

See "Use of AutoLISP in Menu Macros" in the *AutoLISP Developer's Guide*.

Swap and Insert Pull-Down Menus

Because AutoCAD pull-down menus are the cascading type, you usually don't need to swap menus. Also, swapping menus can detract from the consistency of the user interface. However, inserting and removing a pull-down menu can be appropriate when the user specifically loads or unloads an application that requires an additional menu.

Swap Pull-Down Menus

Using \$ commands in menu macros, you can swap pull-down menus in specific `POPn` locations. This method, however, is not recommended unless you can verify that the menu you are replacing is really the one you think it is. Because of the dynamic nature of AutoCAD menus, a menu you inserted at position `P6` might not actually be at that location. If you try to swap this menu for another, you might not remove the correct menu. An alternative method for menu swapping involves relative (or global) referencing (see "Insert and Remove Pull-Down Menus"). Using this method, you can insert the new menu in front of a known menu and then remove the known menu.

For menu-swapping purposes, the active pull-down menu areas are named `P1` through `P16`. The following menu macro replaces the menu at position `P3` with the menu named `JoesMenu` in the menu group `MYMENU`.

```
$P3=MyMenu.JoesMenu
```

The same thing can be done with the AutoLISP `menucmd` function as follows:

```
(menucmd "P3=MyMenu.JoesMenu")
```

You can use the `$Pn=*` special command from within any menu macro to force the menu currently assigned to area `POPn` to be displayed.

Note The swapping of pull-down menus does not conform to the Microsoft® user interface guidelines and is not guaranteed to be available in future releases of AutoCAD.

Insert and Remove Pull-Down Menus

You can use the AutoLISP `menucmd` function to insert or remove a pull-down menu. The syntax is similar to that used to swap pull-down menus except that the left side of the assignment is the pull-down menu before which you want the new menu to be inserted. The right side of the assignment is a plus sign (+), followed by the name of the menu group, followed by a period and the menu's alias, as shown in the following syntax:

```
(menucmd "Gmenugroup1.menuname1+=menugroup2.menuname2")
```

You can also insert a menu with the `Pn=` syntax. The following menu macro inserts a menu after the `P5` menu. (You can also use the `menucmd` function with this format.)

```
(menucmd "P5+=mymenu.new3")
```

If you use this method to insert a menu, remember that you cannot rely on its being inserted at the `P6` menu location as you might expect. There are two reasons that this may not be the case:

- If the current menu bar has only three menus, inserting a menu after menu `P5` results in the new menu's location being `P4`.
- If the user inserts or removes a menu with the `MENULOAD` command or when another application inserts or removes menus, menu numbering can get out of sync.

This is the syntax for removing a menu:

```
(menucmd "Gmenugroup.menuname=-")
```

The following example removes the menu `NEW3` that is a member of the `MyMenu` group.

```
(menucmd "Gmymenu.new3=-")
```

As you might expect, the preceding format is preferable to the `Pn=` format because it removes only the specified menu. The following example removes the menu at the `P4` location (whatever it is).

```
$P4=-
```

Note Use the `Pn` syntax as part of the syntax for a `menucmd` statement only. Use the `$Pn` syntax for menu macro-specific statements.

Controlling Toolbars Across Partial Menus

To control toolbars across partial menus, use the following syntax at the Toolbar Name prompt of the `-TOOLBAR` command on the command line.

```
menugroup.subsection-name
```

This syntax accesses the toolbar identified by `menugroup.menuname` and allows you to use the full spectrum of `-TOOLBAR` command options on that toolbar.

If the menu group is left out of any of these commands and functions, then AutoCAD defaults to the base menu.

You should be aware of the following:

- You cannot swap into the `POP0` menu position. However, you can swap a `POP0` menu into any other Pop menu position.
- Image menus cannot be swapped from external menu files.
- You can swap menus only of the same type, that is, one Aux for another, one Pop for another, and so on. Trying to swap between types may result in unpredictable behavior. However, within a given type, you can swap any menu for any other menu. This can lead to strange behavior for Tablet menus, because they typically do not all have the same number of macros.

Customize Toolbars

The Toolbars section of the MNU file specifies the default layout and contents of the toolbars. It contains a submenu for each toolbar defined by the menu.

Create Toolbars

In the Toolbars section of an MNU file, you can create toolbars with buttons, flyouts, and special control elements and use your own bitmaps for the button icons.

If you just want to create or change toolbars or create, rearrange, add, or remove buttons and flyouts, you can use `CUSTOMIZE`. See “Create Custom Toolbars”.

The menu item syntax for the Toolbars section of the MNU file is shown in the following example. All lines other than the separator begin with a standard name tag, which is used to associate help information with the item. In the example, `**TOOLS1` is a submenu that uses the alias `TOOLS1` as a label to reference the subsequent toolbar definition.

```
***TOOLBARS
**TOOLS1
TAG1 [Toolbar ("tbname", orient, visible, xval, yval, rows)]
TAG2 [Button ("btnname", id_small, id_large)]macro
TAG3 [Flyout ("flyname", id_small, id_large, icon, alias)]macro
TAG4 [Control (element)]
[--]
```

The first line of a toolbar submenu is the toolbar definition (TAG1 in the example), which defines the characteristics of the toolbar. The remaining lines in the submenu can be a mix of the remaining toolbar items. The second line in the example above (TAG2) defines a button. The third line (TAG3) defines a flyout control, and the fourth line (TAG4) defines a special control element. The fifth line defines a separator (--).

The toolbar definition includes the keyword `Toolbar` and a series of options that are contained in parentheses. The options define the display characteristics of the toolbar.

TAG1 [`Toolbar ("tbarname", orient, visible, xval, yval, rows)`]

The options are as follows:

<code>tbarname</code>	The string that names the toolbar. The string must include alphanumeric characters with no punctuation other than a dash (-) or an underscore (_). With this name (along with the alias) the toolbar can be referenced programmatically.
<code>orient</code>	The orientation of the toolbar. The values are <code>floating</code> , <code>top</code> , <code>bottom</code> , <code>left</code> , and <code>right</code> and are not case-sensitive.
<code>visible</code>	The visibility of the toolbar. The values, <code>show</code> and <code>hide</code> , are not case-sensitive.
<code>xval</code>	A numeric value specifying the X coordinate in pixels. This value is measured from the left edge of the screen to the left side of the toolbar.
<code>yval</code>	A numeric value specifying the Y coordinate in pixels. This value is measured from the top edge of the screen to the top of the toolbar.
<code>rows</code>	A numeric value specifying the number of rows.

The following example is the first few lines of the *Zoom* toolbar in *acad.mnu*:

```

**TB_ZOOM
ID_TbZoom [_Toolbar("Zoom", _Floating, _Hide, 100, 380, 1)]
ID_ZoomWindo [Button("Zoom Window", ICON_16_ZOOWIN,
ICON_16_ZOOWIN,)]'_zoom_w
ID_ZoomDynam [Button("Zoom Dynamic", ICON_16_ZOODYN,
ICON_16_ZOODYN,)]'_zoom_d
ID_ZoomScale [Button("Zoom Scale", ICON_16_ZOOSCA,
ICON_16_ZOOSCA,)]'_zoom_s

```

Note Each line begins with `ID` and there are no line breaks.

To control Toolbars with partial menus, use the following syntax at the Toolbar Name prompt of the -TOOLBAR command on the command line:

```
menugroup.toolbarname
```

The following AutoLISP code displays the toolbar MYBAR in the menu group MYGROUP. (This code assumes that the MYGROUP menu is already loaded.)

```
(command "toolbar" "mygroup.mybar" "show")
```

If *menugroup* is not included, then AutoCAD defaults to the base menu.

See Also

“Create Toolbar Buttons” on page 84

“Create Toolbar Flyouts” on page 85

“Define Controls for a Toolbar” on page 86

“Specify User-Defined Bitmaps” on page 87

Create Toolbar Buttons

Toolbar button definitions in the MNU file contain the keyword `Button` followed by options that are contained in parentheses.

```
TAG2 [Button ("btnname", id_small, id_large)]macro
```

The options are as follows:

<code>btnname</code>	The string that names the button. The string must include alphanumeric characters with no punctuation other than a hyphen (-) or an underscore (_). This string is displayed as a tooltip when the cursor is placed over the button.
<code>id_small</code>	The string that names the ID string of the small-image resource (16 × 15 bitmap). The string must include alphanumeric characters with no punctuation other than a hyphen (-) or an underscore (_). It can also specify a user-defined bitmap (see “Specify User-Defined Bitmaps” on page 87).
<code>id_big</code>	The string that names the ID string of the large-image resource (32 × 30 bitmap). If the specified bitmap is not 32 × 30, AutoCAD scales it to that size. The string must include alphanumeric characters with no punctuation other than a hyphen (-) or an underscore (_). This can also specify a user-defined bitmap (see “Specify User-Defined Bitmaps” on page 87).

macro The menu item macro. It follows the standard menu macro syntax.

See Also

“Create Toolbars” on page 82

Create Toolbar Flyouts

Toolbar flyout definitions in the MNU file contain the keyword `Flyout` followed by options that are contained in parentheses.

```
TAG3 [Flyout ("flyname", id_small, id_large, icon, alias)]macro
```

The options are as follows:

flyname The string that names the flyout. The string must include alphanumeric characters with no punctuation other than a hyphen (-) or an underscore (_). This string is displayed as a tooltip when the cursor is placed over the flyout.

id_small The string that names the ID string of the small-image resource (16 × 15 bitmap). The string must include alphanumeric characters with no punctuation other than a hyphen (-) or an underscore (_). This can also specify a user-defined bitmap (see “Specify User-Defined Bitmaps” on page 87).

id_big The string that names the ID string of the large-image resource (32 × 30 bitmap). If the specified bitmap is not 32 × 30, AutoCAD scales it to that size. The string must include alphanumeric characters with no punctuation other than a hyphen (-) or an underscore (_). It can also specify a user-defined bitmap (see “Specify User-Defined Bitmaps” on page 87).

icon The keyword that controls whether to display either its own icon or the last icon selected (other). The acceptable values, `ownIcon` and `otherIcon`, are not case-sensitive.

alias The reference to the toolbar to display as the flyout. The alias refers to a toolbar submenu defined with the standard `**aliasname` syntax.

macro The menu item macro. It follows the standard menu macro syntax.

See Also

“Create Toolbars” on page 82

Define Controls for a Toolbar

Toolbar control definitions in the MNU file contain the keyword `control` followed by a name specifying the type of control element you want contained in parentheses. For examples of the `Color`, `Linetype`, and `Lineweight` controls, see the Properties toolbar in AutoCAD.

`TAG4 [Control (element) ]`

The values for the *element* parameter specify the following controls (the values are not case-sensitive):

<code>_Color</code>	Color control element. This element is a drop-down list that provides specification of the current color.
<code>_Dimstyle</code>	Dimension style control element. This element is a drop-down list that provides specification of the current dimension style.
<code>_Layer</code>	Layer control element. This element is a drop-down list that provides control of the current layers in the drawing.
<code>_Linetype</code>	Linetype control element. This element is a drop-down list that provides specification of the current linetype.
<code>_Lineweight</code>	Lineweight control element. This element is a drop-down list that provides specification of the current lineweight.
<code>_PlotStyle</code>	Plot style control element. This element is a drop-down list that provides specification of the current plot style.
<code>_Refblkname</code>	Xref name control element. This element displays the current xref name in edit mode.
<code>_UCSManager</code>	UCS control element. This element is a drop-down list that provides specification of the current UCS.
<code>_View</code>	View control element. This element is a drop-down list that provides specification of the current standard 3D views.
<code>_ViewportScale</code>	Viewport scale control element. This element is a drop-down list that provides specification of viewport scaling in layouts.

See Also

“Create Toolbars” on page 82

Specify User-Defined Bitmaps

User-defined bitmaps can be used in place of the *id_small* and *id_big* image resource names in the button and flyout menu items.

A user-defined bitmap must be of the proper size for the *id_small* parameter (16 pixels wide by 15 pixels high) and must reside in the library search path. For the *id_big* parameter, if the specified bitmap is not 32×30 , AutoCAD scales it to that size. Specify a user-defined bitmap with the file name and *.bmp* extension as shown in the following example:

```
TAG34 [Button ("My Command", mycmd16.bmp, mycmd32.bmp) ] ^C^CMYCMD
```

See Also

“Create Toolbars” on page 82

Create Image Tile Menus

The main purpose of an image tile menu is to provide an image when the user must select a graphical symbol instead of text.

Overview of Image Tile Menus

You define an image tile menu by including an Image section in the menu file. AutoCAD displays images in groups of 20, along with a scrolling list box containing the associated slide file names or related text. Image tile submenus are unlimited in length. If an image tile submenu contains more than 20 slides, AutoCAD provides Next and Previous buttons so that the user can leaf through pages of images.

The Image section uses submenus similar to the Toolbars and Screen sections. As with Pop menu sections, the first line of the submenu is its title. The title is displayed as the label of the dialog box that contains the images. Submenus should be separated by at least one blank line to clear out items from a previous submenu.

Image tile menu items use item labels to define the text of the scrolling list and the image itself. The label is followed by an associated menu macro. Image tile menus cannot contain name tags.

Image Tile Item Labels

Labels in an image tile menu generally refer to slide file names instead of text labels that are displayed on the screen. The slide file contains the image to show for that selection. The name of the slide file, which can be a single slide or part of a library, should appear exactly as you would enter it at the VSLIDE command.

When successive slides from the same library are displayed, the library file remains open. Therefore, the time required to display an image menu is significantly reduced. The SLIDELIB utility can be used to combine multiple slide files into a slide library.

Image tile menu labels are displayed in a scrolling list box that can accommodate up to 19 characters per label. The slide file name is typically displayed; however, the following icon menu-labeling options are also available.

[*sldname*] The slide name *sldname* is displayed in the list box, and the slide *sldname* is displayed as an image.

[*sldname, labeltext*] The text *labeltext* is displayed in the list box, and the slide *sldname* is displayed as an image.

[*sldlib(sldname)*] The slide name *sldname* is displayed in the list box, and the slide *sldname* in the slide library *sldlib* is displayed as an image.

[*sldlib(sldname, labeltext)*] The text *labeltext* is displayed in the list box, and the slide *sldname* in the slide library *sldlib* is displayed as an image.

[blank] When you supply the text blank as an icon label, a separator line is displayed in the list box and a blank image is displayed.

[*labeltext*] When the first character of an item label is a space, the text supplied as *labeltext* is displayed in the list box and no image is displayed. In this case you can include related commands and simple items such as Exit without needing to make slides that contain those words.

Image Tile Menu Macros

Image tile menu macros can perform the same function as other menu macros; however, you cannot use the menu macro repetition feature. These menu macros can contain menu commands, including `$I=` commands. It is possible, therefore, to construct hierarchical image tile menus in which a selection displays another image tile menu, and so on. Because the activation of these menus is sequential rather than nested, the complexity of the structures you can create has no limits.

Display of Image Tile Menus

The `$I=` macro command calls the image tile menu. Before you can display an image tile menu you must load it using the following syntax:

```
$I=[menugroup.]menuname
```

The `$I=*` macro command displays the currently loaded image tile menu. For example, the following macros load and display the `IMAGE_POLY` image tile menu in `acad.mnu`.

```
$I=image_poly $I=*
```

The following example loads and displays the `MYBLOCKS` image menu from a partially loaded menu group `MYGROUP`.

```
$I=mygroup.myblocks $I=*
```

You can also use the AutoLISP `menucmd` function to load and display image tile menus. The following code produces the same result as the previous example.

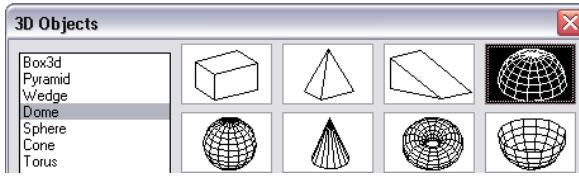
```
(menucmd "I=mygroup.myblocks")  
(menucmd "I=*")
```

Sample Image Tile Menus

This example shows an image tile submenu named `3DObjects`.

```
**3DOBJECTS  
[ 3D Objects]  
[acad(box3d)]^c^cai_box  
[acad(Pyramid)]^c^cai_pyramid  
[acad(Wedge)]^c^cai_wedge  
[acad(Dome)]^c^cai_dome  
[acad(Sphere)]^c^cai_sphere  
[acad(Cone)]^c^cai_cone  
[acad(Torus)]^c^cai_torus  
[acad(Dish)]^c^cai_dish  
[acad(Mesh)]^c^cai_mesh
```

The resulting figure is a portion of the image tile menu.



3D Objects image tile menu sample

In the next example, an image tile menu is used to insert various electronic parts. The text label is an item that swaps to another image tile submenu that contains various fasteners.

```
***IMAGE
**IPARTS
[Electronic Parts]
[cap]^Cinsert cap
[res]^Cinsert res
[neon]^Cinsert neon
[triode]^Cinsert triode
[tetrode]^Cinsert tetrode
[ Fasteners]$I=ifast $I=*

**IFAST
[Fasteners]
[nut632]...
```

To activate this image tile menu, you could choose a menu item such as the following from any menu.

```
[Electronic parts]$I=iparts $I=*
```

In the following variation, the images are retrieved from a slide library named *elib*; only the slide name appears in the list box.

```
***IMAGE
**IPARTS
[Electronic Parts]
[elib(cap)]^Cinsert cap
[elib(res)]^Cinsert res
[elib(neon)]^Cinsert neon
[elib(triode)]^Cinsert triode
[elib(tetrode)]^Cinsert tetrode
```

Prepare Slides for Image Tile Menus

You can use any slide generated by AutoCAD as an image. However, the optimal use of image tile menus requires that you take care in preparing slides that will serve as images. Keep the following suggestions in mind as you prepare slides for an image tile menu.

- Keep the image simple. When an image tile menu is displayed, the user must wait for all images to be drawn before making a selection. If you are showing the user numerous complex symbols, make the images simple versions rather than full renditions. An image should be as simple as possible and yet immediately recognizable.
- Fill the box. Screen space is limited, and images appear in small portions of the full screen. When making a slide for an image, be sure to fill the screen with the image before starting MSLIDE. If the image is very wide and short, or long and thin, the image tile menu will look best if you use PAN to center the image on the screen before making the slide.

Images are displayed with an aspect ratio of 3:2 (3 units wide by 2 units high). If your drawing area has a different aspect ratio, it can be difficult to produce image slides that are centered in the image tile menu. If you work within a layout viewport that has an aspect ratio of 3:2, you can position the image and be assured that it will look the same when it is displayed in the image tile menu.

- Use the -SHADEMODE command for solid-filled areas before you generate the slide. Otherwise, objects such as wide polylines and traces are displayed as outlines.
- Remember the purpose of these images. Do not make use of images to encode abstract concepts into symbols. Image tiles are useful primarily when the user must select a graphic symbol.

Create Screen Menus

The screen menu section of the MNU file controls the screen menu area.

By default, the screen menu is disabled. To enable the screen menu, click Display Screen Menu on the Display tab in the Options dialog box.

In the MNU file, the ***SCREEN section label represents the beginning of the AutoCAD screen menus. The submenu section label shown here is identified by the string **s. A simple, concise name, such as this, is convenient when many separate items reference this submenu, as shown in the following example:

Screen menu

Screen menu file section

Screen Menu X	
AutoCAD	***SCREEN
****	**S
FILE	[AutoCAD] ^C^C^P(ai_rootmenus) ^P
EDIT	[* * * *] \$\$=ACAD.OSNAP
VIEW 1	[FILE] \$\$=ACAD.01_FILE
VIEW 2	[EDIT] \$\$=ACAD.02_EDIT
INSERT	[VIEW 1] \$\$=ACAD.03_VIEW1
FORMAT	[VIEW 2] \$\$=ACAD.04_VIEW2
TOOLS 1	[INSERT] \$\$=ACAD.05_INSERT
TOOLS 2	[FORMAT] \$\$=ACAD.06_FORMAT
DRAW 1	[TOOLS 1] \$\$=ACAD.07_TOOLS1
DRAW 2	[TOOLS 2] \$\$=ACAD.08_TOOLS2
DIMENSION	[DRAW 1] \$\$=ACAD.09_DRAW1
MODIFY1	[DRAW 2] \$\$=ACAD.10_DRAW2
MODIFY2	[DIMENSION] \$\$=ACAD.11_DIMENSION
	[MODIFY1] \$\$=ACAD.12_MODIFY1
	[MODIFY2] \$\$=ACAD.13_MODIFY2
HELP	[HELP] \$\$=ACAD.14_HELP
ASSIST	
LAST	[ASSIST] \$\$=ACAD.ASSIST
	[LAST] \$\$=ACAD.

Screen Submenus

Screen menu submenu labels have the following format:

****menuname [startnum]**

The *menuname* is a string of up to 33 characters containing letters, digits, and the dollar (\$), hyphen (-), and underscore (_) characters. The submenu label must reside on a menu line by itself and must not contain embedded blanks. An optional integer *startnum*, which specifies the start line of the submenu, can follow *menuname*.

A submenu can contain any number of items, but the total size of the screen menu is limited by the setting of the SCREENBOXES system variable (typically set to 28). For instance, if a screen menu submenu has 21 items, but the screen can display only 20 items at a time, the last item in the submenu is inaccessible.

When a submenu is activated, its items normally replace those of the previous menu starting at the beginning (menu box 1) and continuing through all items of the submenu. Thus, a submenu can replace only a portion of the previous menu. You can add an item number after the section or submenu label to specify a replacement starting with a menu item other than 1, as shown in the following example:

****SAMPLE 3**

When the **SAMPLE** submenu is activated, the first two menu boxes are unchanged and submenu replacement begins with menu box 3.

To restore the previous screen items, a menu item must issue the following code without a submenu label.

\$S=

AutoCAD keeps track of the last eight submenus. If you exceed eight, the first menus are discarded.

The following sample screen menu section demonstrates the use of submenus.

```
***SCREEN
[EASYmenu]
```

Blank line

```
[DRAW... ]$S=Draw_Root
[EDIT... ]$S=Edit_Root
```

Blank line

```
[Bye      ]end
```

Three blank lines fill out this page of the menu to 10 lines and blank out items displayed by the submenus. Because no submenu extends below this line, it is displayed in all menus. It recalls the main menu.

```
[ -MAIN- ]$S=SCREEN
```

```
**Draw_Root 2
```

The 2 after the submenu name starts this menu on the line after [EASYmenu].

```
[Line      ]line
[Circle    ]circle
[Arc        ]arc
```

At least one blank line.

```
**Edit_Root 2
```

```
[Erase ]$S=Sel_obj erase
[Copy  ]$S=Sel_obj copy
[Move  ]$S=Obj_sel move
```

Notice the use of a menu alias. At least two blank lines cover up the Sel_obj menu items.

<pre> **Obj_sel 2 **Sel_obj 2 [Last]last [Previous]previous [Window]window [Crossing]crossing </pre>	<i>You can use both Obj_sel and Sel_obj to call this menu.</i>
<pre> [-PREV-]\$\$= </pre>	<i>The \$\$= calls the previous menu.</i>
<pre> ***BUTTONS1 ; redraw </pre>	<i>Pointing-device button menu. Assigns ENTER to button 2. Assigns REDRAW command to button 3.</i>

The previous example contains three submenus: Draw_Root, Edit_Root, and Sel_obj.

Draw_Root and Edit_Root are called from the main screen menu when you select the Draw or Edit menu items. The Draw_Root submenu provides three selection items that correspond to AutoCAD commands. The Edit_Root submenu also contains three selection items, each of which calls the submenu Sel_obj before executing the appropriate command.

In all cases, a –MAIN– selection item recalls the main screen menu. A screen menu writes over (erases) only as many lines of the previous screen menu as it contains. If a screen menu contains more items than boxes on the screen, or if a buttons menu contains more items than buttons available, the excess items are ignored.

You can use blank lines in menu files to lengthen submenus so that they cover up previous menus. You can also include blank lines to improve the readability of the file.

Selecting a menu item called Zoom from the main screen menu can activate a submenu containing the options for the ZOOM command.

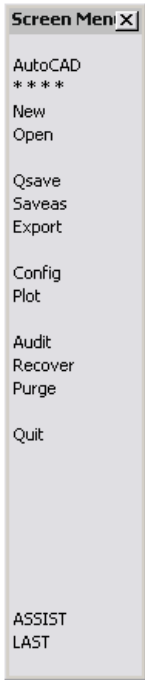
For an alternative method of calling a command submenu, see “Automatic Swapping of Screen Submenus” on page 96.

The following example references the submenu **01_FILE in the ACAD menu group.

```
[FILE      ]$$=ACAD.01_FILE
```

Most screen menus in *acad.mnu* are loaded at menu box 3, enabling the menu labels [AutoCAD] and [* * * *] to remain on the screen.

The following example shows how the `**01_FILE` submenu is displayed on the screen. Notice that the first line (for New) is displayed at menu box 3.

Screen menu	Screen menu file section
	<pre>**01_FILE 3 [New] ^C^C_new [Open] ^C^C_open [Qsave] ^C^C_qsave [Saveas] ^C^C_saveas [Export] ^C^C_export [Config] ^C^C_config [Plot] ^C^C_plot [Audit] ^C^C_audit [Recover] ^C^C_recover [Purge] ^C^C_purge [Quit] ^C^C_quit ASSIST LAST</pre>

The menu items Assist and Last are displayed at the bottom of the screen menu area because they are part of the `**s` submenu that is not overwritten by the `**01_FILE` submenu.

Screen Menu Item Labels

If a screen menu item does not contain an item label, the first eight characters of a menu macro appear on the screen menu. The command in the following example would be displayed as SNAP 0.001.

SNAP 0.001

If a label is provided, the first eight characters of the label are displayed in the appropriate screen menu box. Any additional characters can serve as comments.

Note The maximum number of menu items depends on your system. You can retrieve the number of screen menu boxes with the SCREENBOXES system variable.

Automatic Swapping of Screen Submenus

The MENUCTL system variable controls the automatic swapping of screen submenus when a corresponding command is issued. When MENUCTL is set to 1 (on) and an AutoCAD command is called from a menu item, AutoCAD issues a `$S=cmdname` (where *cmdname* is the name of the command), which calls a screen submenu of the same name as the command. The Standard menu, *acad.mnu*, takes advantage of this feature by setting MENUCTL to 1 from the *acad.mnl* file. Setting MENUCTL to 0 (off) affects the operation of the Standard menu but may be preferable for older custom menus.

To display the screen menu

- 1 On the Tools menu, click Options.
- 2 Click the Display tab.
- 3 On the Display tab under Window Elements, select Display Screen Menu.
- 4 Click OK.

Create Tablet Menus

You can configure up to four areas of your digitizing tablet as menu areas for command input. The sections of the menu file labeled `TABLET1` through `TABLET4` define the menu macros associated with tablet selections in these areas.

The menu items in Tablet sections use the same syntax as those in the other sections. Item labels can be used as comments and are not displayed.

The tablet menu areas that you define with the Cfg option of the TABLET command are divided into equal-sized menu selection boxes, which are determined by the number of columns and rows you specify in each area. These tablet menu selection boxes correspond directly to the lines that follow the Tablet section labels from left to right and top to bottom (whether or not they contain text).

For example, if you configure a menu area for five columns and four rows, the menu item on the line immediately following the section label corresponds to the leftmost selection box in the top row. Similarly, the menu item on the eighth line following the section label corresponds to the third box from the left in the second row. AutoCAD can recognize up to 32,766 menu items in each tablet section, which should be more than enough for any tablet menu.

You can add your own menu macros to the ***TABLET1 section of *acad.mnu*. The menu item labels in this area correspond to the 225 boxes at the top of your tablet template (rows A through I and columns 1 through 25). You can add your menu macro after the corresponding [row-column] menu label, using standard menu item syntax.

```
***TABLET1
[A-1 ]
[A-2 ]
[A-3 ]
.
.
.
[I-25]
```

It is not recommended that you modify any lines following box I-25.

See Also

“Create Menu Macros” on page 54

Create Status Line Help Messages

Status line help messages are the simple, descriptive messages that are displayed on the status line when a menu item is chosen. The Helpstrings menu section defines these messages.

The following example shows a simple menu file that makes use of the Helpstrings menu section.

```
***MENUGROUP=sample
***POP1
ID_Title  [/TTitle]
```

```

ID_Cancel [Cancel Command]^C^C
ID_Line   [/LLine]^C^C_line
          [Disable Line](menucmd "Gsample.ID_Line=~")
          [Check Line](menucmd "Gsample.ID_Line=!.")

***POP2
[/2Title2]
[Another Pull Down](menucmd "Gsample.ID_Line=~")

***HELPSTRINGS
ID_Title  [This is the Title menu]
ID_Cancel [This item cancels the previous command]
ID_Line   [This draws a simple line]

```

The syntax for the Helpstrings section is a name tag followed by a label. When a menu item is highlighted, the name tag for that item is queried for a corresponding entry in the Helpstrings section. If a match occurs, the string contained within the label is displayed on the status line.

Create Shortcut Keys

You can define your own shortcut keys (or accelerator keys). The following is a short example of an Accelerators section in an MNU file.

```

***ACCELERATORS
ID_Line   [SHIFT+CONTROL+"L"]
[CONTROL+"Q"]^C^C_quit
[CONTROL+SHIFT+"Z"]^C^Czoom extents

```

The Accelerators section contains menu items in one of two formats, both illustrated in the above example:

- The first menu item maps a key sequence to a menu item: `ID_Line [SHIFT+CONTROL+"L"]`. A name tag (`ID_Line`) is followed by a label containing modifiers: `SHIFT+CONTROL+`. The modifiers are followed by either a single-character key string (`L` in the example) or a special virtual key string (such as `F12`) enclosed in quotation marks. When a special key sequence is recognized, the menu item associated with the name tag is executed as if the user had chosen the menu item.
- The second and third menu items map a key sequence to a menu macro, not a menu item. A label containing a modifier and key string (`CONTROL+"Q"`) is followed by a menu macro (`^C^C_quit`). The menu macro uses the standard format and special characters, except that the backslash character (`\`) cannot be used to pause for user input.

If you want a pause for user input, use the first method to map a key sequence to a menu item that includes the desired pause.

You can concatenate modifiers by using the plus symbol (+), as in the first and third examples. The valid modifiers are **CONTROL** (the CTRL key) and **SHIFT** (the SHIFT key).

The following table lists the special virtual keys that you can use with modifiers. Both standard keyboard letters and numbers and these special virtual keys must be enclosed in quotation marks.

Special virtual keys		
String	Description	Exceptions
F1	F1 key	It is not recommended that you assign a menu macro to the F1 key, because F1 is generally associated with Help. Using a modifier with this key is acceptable.
F2	F2 key	Unmodified, switches the state of the text window.
F3	F3 key	Unmodified, runs OSNAP.
F4	F4 key	Unmodified, turns TABMODE on or off.
F5	F5 key	Unmodified, turns ISOPLANE on or off.
F6	F6 key	Unmodified, turns COORDS on or off.
F7	F7 key	Unmodified, turns GRIDMODE on or off.
F8	F8 key	Unmodified, turns ORTHOMODE on or off.
F9	F9 key	Unmodified, turns SNAPMODE on or off.
F10	F10 key	Unmodified, turns Polar Tracking on or off.
F11	F11 key	Unmodified, turns Object Snap Tracking on or off.
F12	F12 key	None
INSERT	INS key	Must be used with the CONTROL modifier.
DELETE	DEL key	Must be used with the CONTROL modifier.
ESCAPE	ESC key	It is not recommended that you assign a menu macro to the ESC key, because ESC is generally associated with Cancel. CONTROL+ESCAPE and CONTROL+SHIFT+ESCAPE cannot be assigned a menu macro; these sequences are controlled by Windows. Using the SHIFT modifier with this key is acceptable.

Special virtual keys (continued)

String	Description	Exceptions
UP	UPARROW key	Must be used with the CONTROL modifier.
DOWN	DOWNARROW key	Must be used with the CONTROL modifier.
LEFT	LEFTARROW key	Must be used with the CONTROL modifier.
RIGHT	RIGHTARROW key	Must be used with the CONTROL modifier.
NUMPAD0	0 key	None
NUMPAD1	1 key	None
NUMPAD2	2 key	None
NUMPAD3	3 key	None
NUMPAD4	4 key	None
NUMPAD5	5 key	None
NUMPAD6	6 key	None
NUMPAD7	7 key	None
NUMPAD8	8 key	None
NUMPAD9	9 key	None

Coordinate Entry with Shortcut Keys (Example)

Users who make extensive use of coordinate entry might find the following menu enhancement very useful.

```
[ "NUMPAD5" ]@x^h  
[ "NUMPAD6" ]<0  
[ "NUMPAD9" ]<45  
[ "NUMPAD8" ]<90  
[ "NUMPAD7" ]<135  
[ "NUMPAD4" ]<180  
[ "NUMPAD1" ]<-135  
[ "NUMPAD2" ]<-90  
[ "NUMPAD3" ]<-45
```

If this code is added to the Accelerators section, your numeric keypad is modified as follows: the ENTER key enters the @ symbol and the other number keys enter the less-than symbol (<), followed by the angular value represented by its location in the keypad. For example, if you wanted to draw a square that was 3 units on each side, you would enter the following:

Command: **line**

From point: *(specify start point)*

To point: *(press number 5)* **3** *(press number 6)*

To point: *(press number 5)* **3** *(press number 2)*

To point: *(press number 5)* **3** *(press number 4)*

To point: **c**

DIESEL

You can use DIESEL (Direct Interpretively Evaluated String Expression Language) to alter the AutoCAD status line through the MODEMACRO system variable. You can also use DIESEL in menu items as a macro language instead of AutoLISP. DIESEL expressions accept strings and generate string results.

Because DIESEL expressions handle strings exclusively, the USERS1-5 system variables are useful for passing information from an AutoLISP routine to a DIESEL expression. DIESEL expressions are evaluated by AutoLISP routines through the use of the AutoLISP `menucmd` function.

5

In this chapter

- Customize the Status Line
- DIESEL Expressions in Menu Macros
- Catalog of DIESEL Functions
- DIESEL Error Messages

Customize the Status Line

You can use the MODEMACRO system variable to display information on the status line.

Overview of the MODEMACRO System Variable

The status line can provide the user with important information without interrupting the work flow. The MODEMACRO system variable controls the user-defined area on the status line. The calculated value of the MODEMACRO system variable is displayed in a left-aligned panel in the status bar at the bottom of the AutoCAD® window. This variable is set to the null string when you start AutoCAD. Its value is not saved in the drawing, the configuration file, or anywhere else.

The number of characters displayed on the status line is limited only by the size of the AutoCAD window (and your monitor). The default panels move to the right as the content of the MODEMACRO panel grows. It is possible to push the default panels completely off the screen (if you want to).

You can use the MODEMACRO system variable to display in the status line most data known to AutoCAD. With its calculation, decision, and editing facilities you can compose the status line to your precise specifications.

MODEMACRO is a user-string variable. It can be set to any string value. A maximum string length of 460 characters should work on all systems. You can set MODEMACRO with SETVAR or by entering **modemacro** at the Command prompt. If you modify the MODEMACRO setting, you can experiment with various status line formats; however, the maximum number of characters you can enter in this manner is 255.

If you set MODEMACRO to the null string by entering a period (.), AutoCAD displays the standard status line.

Set MODEMACRO Values

The value of MODEMACRO determines what is displayed in the mode status line. The simplest (and least useful) MODEMACRO consists of constant text. For example, to display a company name in the status line, you enter the following:

Command: **modemacro**

New value for MODEMACRO, or . for none <"">: **Joe's Bank and Grill**

This MODEMACRO value always displays the same text; the status line does not reflect changes to the AutoCAD internal state. It doesn't change until you change MODEMACRO.

To make the status line reflect the AutoCAD current state, enter macro expressions using the DIESEL language in the following format:

```
$(somefun, arg1, arg2, ...)
```

In the macro expression, **somefun** is the name of the DIESEL function (similar to an AutoLISP function name) and *arg1*, *arg2*, and so on, are arguments to the function, interpreted according to the function's definition. Unlike AutoLISP, DIESEL macro expressions have only one data type: strings. Macros that operate on numbers express the numbers as strings and convert back and forth as required.

For descriptions of the DIESEL functions, see "Catalog of DIESEL Functions" on page 111.

Now define a more interesting status line, for example, one that shows the current text style name:

Command: **modemacro**

New value for MODEMACRO, or . for none <"">: **Style: \$(getvar, textstyle)**

- **Style:** is a text string to be displayed on the status line.
- **\$(getvar,textstyle)** is a DIESEL function (**getvar**) and argument that retrieves the current value of the TEXTSTYLE system variable.

Note The examples in this topic may show the MODEMACRO string as more than one line of text. You enter it as one long string at the prompt.

You can retrieve any system variable by entering **\$(getvar, varname)**. The current setting of the system variable replaces the macro expression on the status line. Then, when you switch text styles, for example, MODEMACRO is reevaluated. If it changes, the new text style name is displayed on the status line.

Expressions can be nested, and they can be as complex as you want. The example that follows displays the current snap value and angle (in degrees) in the status line. It uses nested expressions to convert the snap angle from radians to degrees and truncates the value to an integer.

Command: **modemacro**

New value for MODEMACRO, or . for none <"">: **Snap: \$(getvar, snapunit) \$(fix,\$(*,\$(getvar,snapang),\$(/,180,3.14159)))**

You can also display the values in the current linear and angular units modes.

Command: **modemacro**

New value for MODEMACRO, or . for none <"">: **Snap: \$(rtos,\$(index,0,\$(getvar,snapunit))),\$(rtos,\$(index,1,\$(getvar,snapunit))\$(angtos,\$(getvar,snapang))**

DIESEL copies its input directly to the output until it comes to the dollar sign character (\$) or a quoted string. You can use quoted strings to suppress evaluation of character sequences that would otherwise be interpreted as DIESEL functions. You can include quotation marks in quoted strings by using two adjacent quotation marks. In the following example, the current layer is set to LAYOUT, and MODEMACRO is set to the string.

Command: **modemacro**

New value for MODEMACRO, or . for none <"">: **"\$(getvar,clayer)=""\$(getvar,clayer)""**

The status line displays the following:

\$(getvar,clayer)="LAYOUT"

Set MODEMACRO with AutoLISP

Another way to set MODEMACRO values is to use AutoLISP. You can save the code samples shown here as ASCII format text files and load them with the AutoLISP **load** function.

The following AutoLISP command defines a MODEMACRO string that provides similar information to that in the built-in status line. Because AutoLISP cannot continue strings from line to line, you use the AutoLISP **strcat** function to assemble the complete MODEMACRO string from shorter component strings.

```
(defun C:ACADMODE ( )
  (setvar "modemacro"
    (strcat
      "Layer $(substr,$(getvar,clayer),1,8)"
      "$(if,$(getvar,orthomode), Ortho)"
      "$(if,$(getvar,snapmode), Snap)"
      "$(if,$(getvar,tabmode), Tablet)"
      "$(if,$(=,$(getvar,tilemode),0),"
      "  $(if,$(=,$(getvar,cvport),1), P)"
      ")"
    )
  )
)
```

Save this AutoLISP routine in a file called *acadmode.lsp*. When you load the routine and execute it, it displays information on the status line. This is not the most useful application of this feature; it is provided only as an example.

The following sample *acad.lsp* file uses the **S::STARTUP** function to set the MODEMACRO variable to a string defined by the AutoLISP file *model1.lsp*.

```
;;; Sample acad.lsp file that uses S::STARTUP to load the
;;; file MODEL.LSP which defines a MODEMACRO string
(defun S::STARTUP ( )
  (load "model")
  (princ)
)
;;; Additional AutoLISP files can also be defined or
;;; loaded here
```

When the AutoLISP file (*model1.lsp*) is loaded, it uses the MODEMACRO system variable to define a status line that displays *L:* followed by the first eight characters of the layer name, the drawing name and a portion of the path, and the first letter of each name of the currently active modes. The position of the drawing name remains constant, regardless of the length of the layer name.

```
;;; MODEL.LSP
;;;
(setvar "modemacro"
  (strcat
    "L:$(substr,$(getvar,clayer),1,30)"
    "$(substr,          ,1,$(-,30,$(strlen,$(getvar,clayer)))) "
  ;;      ^^^^^^^^ Note the 8 spaces here
  "<.."
    "$(if,$(eq,$(getvar,dwgname),UNNAMED),UNNAMED,"
    "$(substr,$(getvar,dwgname),"
    "$(if,$(>,$(strlen,$(getvar,dwgprefix)),29),"
    "$(-,$(strlen,$(getvar,dwgprefix)),29),1"
    "),"
    "$(strlen,$(getvar,dwgname))"
    ")"
  ">"
    "$(if,$(getvar,orthomode), O, )"
    "$(if,$(getvar,snapmode), S, )"
    "$(if,$(getvar,tabmode), T, )"
    "$(if,$(and,"
    "$($($getvar,tilemode),0),$($($getvar,cvport),1)),P)"
  )
)
```

Indenting code improves the readability of AutoLISP files and DIESEL strings.

DIESEL Expressions in Menu Macros

You can implement DIESEL string expressions in menu files and use them as an additional method of creating macros. These expressions can return string values (text strings) in response to standard AutoCAD commands, AutoLISP and ObjectARX[®] routines, and other menu macros. They can also return string values to the menu itself, thereby altering the appearance or content of a menu label.

A DIESEL expression that you use in a menu item must follow the `$section=submenu` format where the section name is `M` and the submenu is the DIESEL expression you want. Frequently, you can implement a menu macro more easily with AutoLISP.

The following examples show two menu items that produce the same result; one uses DIESEL, and the other uses AutoLISP.

This menu item uses the DIESEL expression:

```
[Ps/Ms]^C^C^P$M=$(if,$(=,$(getvar,cvport),1),mspace,pspace)
```

This menu item uses the AutoLISP expression:

```
[-Ps/Ms-]^C^C^P(if (= (getvar "cvport") 1)(command "mspace")+  
(command "pspace"))(princ) ^P
```

Both menu items provide a way to switch between paper space and model space (if `TILEMODE` is set to 0), but the DIESEL expression is shorter and is evaluated transparently, not requiring the call to the AutoLISP **princ** function. If the special character `^P` (which switches `MENUECHO` on and off) is omitted in both cases, the DIESEL expression displays only the issued command, whereas the AutoLISP expression displays the entire line of code.

Because the value returned by a DIESEL expression is a text string, it can be used in response to an AutoLISP **getxxx** function call. This functionality enables menu items to evaluate current drawing conditions and to return a value to an AutoLISP routine.

The next example is based on four assumptions:

- The AutoLISP routine is loaded into memory.
- The menu excerpt is included in the current menu.
- The symbols to insert are one unit high by one unit wide.
- The `DIMSCALE` variable is set to the drawing's scale factor (that is, a drawing to be plotted at a scale of 1" = 10' would have a scale factor of 120, or a 1/4" = 1' scale drawing would have a scale factor of 48).

If you load and execute the sample AutoLISP routine, AutoCAD inserts the symbol at the size and location you have specified. When plotted, the symbols are the specified size (if the drawing is plotted at the same scale as that specified by DIMSCALE).

The following is a sample AutoLISP routine.

```
(defun C:SYMIN ( )
  (setq sym
    (getstring
      "\nEnter symbol name: ") ; Prompts for a symbol name
    )
  (menucmd "s=symsize") ; Switches the screen menu
                          ; to the symsize submenu
  (setq
    siz (getreal
      "\nSelect symbol size: ") ; Prompts for a symbol size
    pl (getpoint
      "\nInsertion point: ") ; Prompts for insertion point
    )
  (command "insert" ; Issues the INSERT command
    sym ; using the desired symbol
    pl siz siz 0) ; insertion point, and size
  (menucmd "s=") ; Switches to the previous
                  ; screen menu
  (princ) ; Exits quietly
)
```

Note An AutoLISP routine that you use regularly should include error checking to verify the validity of user input.

The DIESEL expressions in the following menu file multiply the current value of DIMSCALE by the specified value, and return an appropriate scale factor. This cannot be done with similar AutoLISP code; a value returned by an AutoLISP expression cannot typically be used as a response to a **getxxx** function call (such as, the **getreal** function in the preceding sample).

The following is a menu file excerpt.

```
**symsize 3
[ SIZES ]

[ 3/8" ]$M=$(*,$(getvar,dimscale),0.375)
[ 1/2" ]$M=$(*,$(getvar,dimscale),0.5)
[ 5/8" ]$M=$(*,$(getvar,dimscale),0.625)
```

DIESEL expressions can also return string values to pull-down menu item labels, so that you can make menus unavailable or otherwise alter the way they are displayed. To use a DIESEL expression in a pull-down menu label, make sure that the first character is the \$ character.

In the next example, the current layer is set to BASE and the following DIESEL expression is used as the label portion of a `***POPn` section in a menu file.

```
[$(eval,"Current layer: " $(getvar,clayer))]
```

The result is that the appropriate pull-down menu is displayed and updated whenever the current layer changes.

Current Layer: BASE

You can also use this method to interactively change the text displayed in a pull-down menu. You use an AutoLISP routine that sets the `USERS1-5` system variables to the selected text, which can be retrieved by a DIESEL macro in a menu label.

Note The width of pull-down and shortcut menus is determined when the menu file is being loaded. Menu labels generated or changed by DIESEL expressions after a menu is loaded are truncated to fit within the existing menu width.

If you anticipate that a DIESEL-generated menu label will be too wide, you can use the following example to ensure that the menu width will accommodate your labels. This example displays the first 10 characters of the current value of the `USERS3` (`USERS1-5`) system variable.

```
[$(eval,"Current value: " $(getvar,users3))+  
$(if, $(eq,$(getvar,users3),""), 10 spaces )]^C^Cusers3
```

You cannot use trailing spaces in a menu label to increase the menu width, because trailing spaces are ignored while the menu is being loaded. Any spaces you use to increase the width of a menu label must be within a DIESEL expression.

The next example uses the same DIESEL expression as the label and a portion of the menu item. It provides a practical way to enter the current day and date into a drawing.

```
[$(edtime,$(getvar,date),DDD", "D MON YYYY)]^C^Ctext +  
\\ $M=$(edtime,$(getvar,date),DDD", "D MON YYYY);
```

Also, you can use a DIESEL macro to mark pull-down menu labels or make them unavailable. The following pull-down menu label displays an unavailable `ERASE` while a command is active. The text is displayed normally when a command is not active.

```
[ $(if,$(getvar,cmdactive),~)ERASE]erase
```

You can use a similar approach to place a mark beside a pull-down menu item or to interactively change the character used for the mark.

Catalog of DIESEL Functions

Status retrieval, computation, and display are performed by DIESEL functions. All functions have a limit of 10 parameters, including the function name itself. If this limit is exceeded, you get a DIESEL error message.

+ (addition)

Returns the sum of the numbers *val1*, *val2*, ..., *val9*.

```
$(+, val1 [, val2, ..., val9])
```

If the current thickness is set to 5, the following DIESEL string returns 15.

```
$(+, $(getvar,thickness),10)
```

– (subtraction)

Returns the result of subtracting the numbers *val2* through *val9* from *val1*.

```
$(-, val1 [, val2 , ..., val9])
```

* (multiplication)

Returns the result of multiplying the numbers *val1*, *val2*, ..., *val9*.

```
$(*, val1 [, val2, ..., val9])
```

/ (division)

Returns the result of dividing the number *val1* by *val2*, ..., *val9*.

```
$(/, val1 [, val2, ..., val9])
```

= (equal to)

If the numbers *val1* and *val2* are equal, the string returns 1; otherwise, it returns 0.

```
$(=, val1, val2)
```

< (less than)

If the number *val1* is less than *val2*, the string returns 1; otherwise, it returns 0.

```
$(< , val1, val2)
```

The following expression gets the current value of HPANG; if the value is less than the value stored in the system variable USERR1, it returns 1. If the value 10.0 is stored in USERR1 and the current setting of HPANG is 15.5, the following string returns 0.

```
$( <, $(getvar, hpang), $(getvar, userr1) )
```

> (greater than)

If the number *val1* is greater than *val2*, the string returns 1; otherwise, it returns 0.

```
$( >, val1, val2 )
```

!= (not equal to)

If the numbers *val1* and *val2* are not equal, the string returns 1; otherwise, it returns 0.

```
$( !=, val1, val2 )
```

<= (less than or equal to)

If the number *val1* is less than or equal to *val2*, the string returns 1; otherwise, it returns 0.

```
$( <=, val1, val2 )
```

>= (greater than or equal to)

If the number *val1* is greater than or equal to *val2*, the string returns 1; otherwise, it returns 0.

```
$( >=, val1, val2 )
```

and

Returns the bitwise logical AND of the integers *val1* through *val9*.

```
$(and, val1 [, val2,..., val9] )
```

angtos

Returns the angular value in the format and precision specified.

```
$(angtos, value [, mode, precision] )
```

Edits the given *value* as an angle in the format specified by the *mode* and *precision* as defined for the analogous AutoLISP function. (The values for *mode* are shown in the following table.) If *mode* and *precision* are omitted, it uses the current values chosen by the UNITS command.

Angular units values	
Mode value	String format
0	Degrees
1	Degrees/minutes/seconds
2	Grads
3	Radians
4	Surveyor's units

edtime

Returns a formatted date and time based on a given picture.

\$(edtime, time, picture)

Edits the AutoCAD Julian date given by *time* (obtained, for example, from **\$(getvar,date)** according to the given *picture*). The *picture* consists of format phrases replaced by specific representations of the date and time. Characters not interpretable as format phrases are copied literally into the result of **\$(edtime)**. Format phrases are defined as shown in the following table. Assume that the date and time are Saturday, 5 September 1998 4:53:17.506.

edtime format phrases			
Format	Output	Format	Output
D	5	H	4
DD	05	HH	04
DDD	Sat	MM	53
DDDD	Saturday	SS	17
M	9	MSEC	506

edtime format phrases (continued)

Format	Output	Format	Output
MO	09	AM/PM	AM
MON	Sep	am/pm	am
MONTH	September	A/P	A
YY	98	a/p	a
YYYY	1998		

Enter the entire **AM/PM** phrase as shown in the preceding table; if **AM** is used alone, the **A** will be read literally and the **M** will return the current month.

If any **AM/PM** phrases appear in the picture, the **H** and **HH** phrases edit the time according to the 12-hour civil clock (12:00–12:59 1:00–11:59) instead of the 24-hour clock (00:00–23:59).

The following example uses the date and time from the preceding table. Notice that the comma must be enclosed in quotation marks because it is read as an argument separator.

```
$(edtime, $(getvar,date),DDD"," DD MON YYYY - H:MMam/pm)
```

It returns the following:

Sat, 5 Sep 1998 – 4:53am

If *time* is 0, the time and date at the moment that the outermost macro was executed is used. This avoids lengthy and time-consuming multiple calls on `$(getvar,date)` and guarantees that strings composed with multiple `$(edtime)` macros all use the same time.

eq

If the strings *val1* and *val2* are identical, the string returns 1; otherwise, it returns 0.

```
$(eq, val1, val2)
```

The following expression gets the name of the current layer; if the name matches the string value stored in the `USERS1` (`USERS1-5`) system variable, it returns 1. Assume the string "PART12" is stored in `USERS1` and the current layer is the same.

```
$(eq, $(getvar,users1),$(getvar,clayer))      Returns 1
```

eval

Passes the string *str* to the DIESEL evaluator and returns the result of evaluating it.

```
$(eval, str)
```

fix

Truncates the real number *value* to an integer by discarding any fractional part.

```
$(fix, value)
```

getenv

Returns the value of the environment variable *varname*.

```
$(getenv, varname)
```

If no variable with that name is defined, it returns the null string.

getvar

Returns the value of the system variable with the given *varname*.

```
$(getvar, varname)
```

if

Conditionally evaluates expressions.

```
$(if, expr, dotrue [, dofalse])
```

If *expr* is nonzero, it evaluates and returns *dotrue*. Otherwise, it evaluates and returns *dofalse*. Note that the branch not chosen by *expr* is not evaluated.

index

Returns the specified member of a comma-delimited string.

```
$(index, which, string)
```

Assumes that the *string* argument contains one or more values delimited by the macro argument separator character, the comma. The *which* argument selects one of these values to be extracted, with the first item numbered 0.

This function is most frequently used to extract *X*, *Y*, or *Z* coordinate values from point coordinates returned by `$(getvar)`.

Applications can use this function to retrieve values stored as comma-delimited strings from the USERS1-5 system variables.

nth

Evaluates and returns the argument selected by *which*.

`$(nth, which, arg0 [, arg1,..., arg7])`

If *which* is 0, `nth` returns *arg0*, and so on. Note the difference between `$(nth)` and `$(index)`; `$(nth)` returns one of a series of arguments to the function, while `$(index)` extracts a value from a comma-delimited string passed as a single argument. Arguments not selected by *which* are not evaluated.

or

Returns the bitwise logical OR of the integers *val1* through *val9*.

`$(or, val1 [, val2,..., val9])`

rtos

Returns the real value in the format and precision specified.

`$(rtos, value [, mode, precision])`

Edits the given *value* as a real number in the format specified by the *mode* and *precision* as defined by the analogous AutoLISP function. If *mode* and *precision* are omitted, it uses the current values selected with the UNITS command.

strlen

Returns the length of *string* in characters.

`$(strlen, string)`

substr

Returns the substring of *string*, starting at character *start* and extending for *length* characters.

`$(substr, string, start [, length])`

Characters in the string are numbered from 1. If *length* is omitted, it returns the entire remaining length of the string.

upper

Returns the *string* converted to uppercase according to the rules of the current locale.

`$(upper, string)`

xor

Returns the bitwise logical XOR of the integers *val1* through *val9*.

`$(xor, val1 [, val2,..., val9])`

DIESEL Error Messages

Generally, if you make a mistake in a DIESEL expression, what went wrong will be obvious. Depending on the nature of the error, DIESEL embeds an error indication in the output stream.

DIESEL error messages	
Error message	Description
\$?	Syntax error (usually a missing right parenthesis or a runaway string)
\$(func,??)	Incorrect arguments to <i>func</i>
\$(func)??	Unknown function <i>func</i>
\$(++)	Output string too long—evaluation truncated

Slides and Command Scripts

Slides are snapshots of drawing files that can be used for giving presentations, for creating image tile menus, and for viewing another drawing while you work.

A script reads and executes commands from a text file. You can run a script when you start AutoCAD, or you can run a script from within AutoCAD using the `SCRIPT` command. A script provides an easy way to create continuously running displays for product demonstrations and trade shows.

6

In this chapter

- Create Slides
- Create Command Scripts

Create Slides

Slides are snapshots of drawing files. You can use slides for giving presentations, creating custom image tile menus, and viewing an image of another drawing while you work.

Overview of Slides

A slide is a snapshot of a drawing. Although it contains a picture of the drawing at a given instant, it is not a drawing file. You cannot import a slide file into the current drawing, nor can you edit or print a slide. You can only view it.

You can use slide files in the following ways:

- For making presentations within AutoCAD®
- For viewing a snapshot of a drawing while working on a different drawing
- For creating menus of image tiles within a dialog box

You create a slide by saving the current view in slide format. A slide created in model space shows only the current viewport. A slide created in paper space shows all visible viewports and their contents. Slides show only what was visible. They do not show objects on layers that were turned off or frozen or objects in viewports that were turned off.

When you view a slide file, it temporarily replaces objects on the screen. You can draw on top of it, but when you change the view (by redrawing, panning, or zooming), the slide file disappears, and AutoCAD redisplay only what you drew and any preexisting objects.

You can display slides one by one or use a script to display slides in sequence. Slides also can be used in custom menus. For example, if you create scripts that insert blocks containing mechanical parts you use frequently, you can design a custom image tile menu that displays a slide of each part. When you click the slide image on the menu, AutoCAD inserts the block into the drawing.

A slide library is a file containing one or more slides. Slide library files are used for creating custom image tile menus and for combining several slide files for convenient file management.

You cannot edit a slide. You must change the original drawing and remake the slide. If you use a low-resolution graphics monitor when creating a slide file and later upgrade to a high-resolution monitor, you can still view the slide. AutoCAD adjusts the image accordingly; however, the slide does not

take full advantage of the new monitor until you remake the slide file from the original drawing.

To make a slide

- 1 Display the view you want to use for the slide.
- 2 At the Command prompt, enter **mslide**.
- 3 In the Create Slide File dialog box, enter a name and select a location for the slide.

AutoCAD offers the current name of the drawing as a default name for the slide and automatically appends the *.slid* file extension.

- 4 Click Save.

The current drawing remains on the screen, and the slide file is saved in the folder that you specified.

View Slides

You can view slides individually using VSLIDE. To view a series of slides for a presentation, use a script file.

Be careful about using editing commands while you view a slide, which looks like an ordinary drawing. Editing commands affect the current drawing underneath the slide but not the slide itself.

Some commands may force redrawing, which removes the slide from display.

To view a slide

- 1 At the Command prompt, enter **vslide**.
- 2 In the Select Slide File dialog box, select a slide to view and click OK.

The slide image is displayed in the drawing area.

- 3 On the View menu, click Redraw.

The slide image disappears.

Create and View Slide Libraries

A slide library is a file containing one or more slides. Slide library files are used for creating custom image tile menus and for combining several slide files for convenient file management.

You can create slide libraries from slide files using the SLIDELIB utility. After you have set up a slide library, you can view slides by specifying the name of the slide library and the slide.

Do not delete the original slides after creating the slide library. The SLIDELIB utility cannot update a slide library once it is created. If you want to add or delete a slide, update the slide list file and remake the library with SLIDELIB. When you remake the slide library, all the slide files that you intend to include must be available.

To create a slide library

- 1 Use a Windows ASCII text editor to create a list of slide files to include in the library. The file would look similar to this example:

```
entrance.sld  
hall.sld  
stairs.sld  
study.sld  
balcony.sld
```

- 2 Name and save the file as a text file with a *.txt* file extension.
- 3 On the Start menu (Windows), click All Programs (or Programs) ► Accessories ► Command Prompt.
- 4 Change to the AutoCAD directory.
- 5 In the Command Prompt window, use the following syntax to create the slide library:

```
slidelib library name<list.txt
```

For example, if you named your text file *areas.txt*, you could create a library called *house.slb* by entering **slidelib house<areas.txt**. The SLIDELIB utility appends the file extension *.slb* to the slide library file.

To view a slide in a slide library

- 1 At the Command prompt, set the FILEDIA system variable to 0.
- 2 At the Command prompt, enter **vslide**.
- 3 Enter **library (slidenam)** to specify the slide.
For example, enter **house (balcony)** to open the *balcony* slide, which is stored in the *house* slide library file.
- 4 On the View menu, click Redraw to remove the slide from the display.

Create Command Scripts

A script is a text file that contains a series of commands. Common uses for scripts are to customize startup and to run slide shows.

Overview of Command Scripts

A script is a text file with one command on each line.

You can invoke a script at startup, or you can run a script from within AutoCAD using the SCRIPT command. A script provides an easy way to create continuously running displays for product demonstrations and trade shows.

The BACKGROUNDPLOT system variable must be set to 0 before a script can plot multiple jobs.

You create script files outside AutoCAD using a text editor (such as Microsoft® Windows Notepad) or a word processor (such as Microsoft® Word) that can save the file in ASCII format. The file extension must be *.scr*.

Each line of the script file contains a command. Each blank space in a script file is significant: AutoCAD accepts either a space or ENTER as a command or data field terminator. You must be very familiar with the sequence of AutoCAD prompts to provide an appropriate sequence of responses in the script file.

Note Keep in mind that AutoCAD prompts and command names may change in future releases, so you may need to revise your scripts when you upgrade to a later version of AutoCAD. For similar reasons, avoid the use of abbreviations; future command additions might create ambiguities.

A script can execute any command at the Command prompt except a command that displays a dialog box. AutoCAD provides command line versions of the dialog box commands.

Script files can contain comments. Any line that begins with a semicolon (;) is considered a comment, and AutoCAD ignores it while processing the script file. The last line of the file must be blank.

All references to long file names that contain embedded spaces must be enclosed in double quotes. For example, to open the drawing *my house.dwg* from a script, you must use the following syntax:

```
open "my house"
```

The following commands are useful in scripts:

'DELAY	Provides a timed pause within a script (in milliseconds)
'GRAPHSCR	Switches from the text window to the drawing area
RESUME	Continues an interrupted script
RSCRIPT	Repeats a script file
'TEXTSCR	Switches to the text window

When command input comes from a script, AutoCAD assumes that the settings of the PICKADD and PICKAUTO system variables are 1 and 0, respectively; therefore, you do not have to check the settings of these variables.

AutoCAD considers a script to be a group, a unit of commands, reversible by a single U command. However, each command in the script causes an entry in the undo log, which can slow script processing. If you like, you can use UNDO Control None to turn off the undo feature before running the script, or you can write it at the beginning of the script itself. Remember to turn it back on (UNDO Control All) when the script is finished.

The script that is running stops when another script command is invoked.

To create a script that changes settings in a drawing

This script turns on the grid, sets the global linetype scale to 3.0, and sets layer 0 as the current layer with red as the color.

- 1 In a text editor, enter **grid on**.
- 2 On the next line, enter **ltscale 3.0**.
- 3 On the next line, enter **layer set 0 color red**.
- 4 Add a blank line.
- 5 Save the file as ASCII text (TXT file), with a file extension of *.scr*.

The script file may contain comments, as follows:

```
; Turn grid on
grid on
; Set scale for linetypes
ltscale 3.0
; Set current layer and its color
layer set 0 color red

; Blank line above to end LAYER command
```

Run Scripts at Startup

A script that runs at startup can open a drawing and change its settings.

Suppose that every time you begin a new drawing, you turn on the grid, set the global linetype scale to 3.0, and set layer 0 as your current layer, with red as the color. You can do this using a template drawing, but you could do it instead with the following script and store it in a text file called *setup.scr*.

```
grid on
ltscale 3.0
layer set 0 color red 0
```

The first line turns on the grid. The second line sets the global scale for line-types. The third line sets the current layer to layer 0 and sets its default color to red. AutoCAD assumes that in a script you want to use the command line version of LAYER rather than the dialog box version. The result is equivalent to entering **-layer** on the command line. The fourth line is blank, ending LAYER.

Note VBA and AutoLISP scripts that run at startup should check for whether the AutoCAD process is visible or invisible. If the process is invisible, the script should not execute, because the process may be performing background plotting or publishing operations. To check for whether the AutoCAD process is visible or invisible, you can use the Visible property of the Application object in the AutoCAD Object Model.

You could run a script at startup to open a drawing by using the following syntax in the Run dialog box:

```
acad drawing_name /b setup
```

All file names that contain embedded spaces must be enclosed in double quotes, for example, "guest house". You can also specify the view that is displayed when the drawing opens by using the /v switch and the view name. The /b switch and the script file must be the last parameter listed.

Including the file extensions *.exe*, *.dwg*, *.dwt*, and *.scr* is optional. If AutoCAD cannot find the script file, AutoCAD reports that it cannot open the file.

To run the same script at startup but create a new drawing using the *MyTemplate.dwt* file as the template, enter the following in the Run dialog box:

```
acad /t MyTemplate /b setup
```

This command creates a new drawing and issues a sequence of setup commands from the *setup.scr* file. When the script has finished running, the Command prompt is displayed. If you want to use the default template for the new drawing, you can omit the /t switch and the template file name.

Note You can no longer use this method to start a new drawing and give it a name. Name the drawing when you save it.

To run a script at startup

- 1 On the Start menu (Windows), click Run.
- 2 In the Run dialog box, enter **acad *drawing_name* /b *script_name***.
To start a new file, instead of a drawing file name, enter the /t switch and the name of a template file: **/t *template_drawing***.
To open a drawing file to a particular view, follow the drawing name with the /v switch and the name of the view: **/v *view_name***.
The name of the script file must be the last parameter listed. The file extensions are optional.
- 3 Click OK.
AutoCAD opens the drawing and executes the commands in the script file. When the script has been completed, the Command prompt is displayed.

Run Slide Shows from Scripts

Scripts are useful for creating slide shows. Ordinarily, the speed with which you can display slides is limited by the number of times AutoCAD must access the disk to read the slide file. You can, however, preload the next slide from disk into memory while your audience is viewing the current slide and then quickly display the new slide from memory.

To preload a slide, place an asterisk before the file name in VSLIDE. The next VSLIDE command detects that a slide has been preloaded and displays it without asking for a file name.

The disk-access time to load the next slide overlaps with the viewing time for the current slide. You can specify additional delays with the DELAY command. Each delay unit is equal to one millisecond.

To stop a repeating script press ESC. You can resume the script with RESUME.

If the script will run for a long time, it is recommended that you use UNDO Control None to turn off the Undo log file.

To run slide shows from scripts

- 1 Create the slide library file as described in “To create a slide library” on page 122.
- 2 Create a script file using an ASCII text editor, as shown in “To create a script that preloads slides” on page 127.
- 3 On the command line, enter **script**.
- 4 In the Select Script File dialog box, select a script file and choose Open.

To create a script that preloads slides

In this example of a script that displays three slides (files *slide1.sld*, *slide2.sld*, and *slide3.sld*), the time it takes to access the disk drive and load the next slide into memory overlaps with the viewing time for the current slide.

- 1 On the first line of the script, enter **vslide slide1**.
The first line begins the slide show and loads *slide1*.
- 2 On the second line, enter **vslide *slide2**.
The asterisk (*) preceding the slide name on the second line preloads *slide2*.
- 3 On the third line, enter **delay 2000**.
The third line specifies a delay of 2000 milliseconds to allow the audience to view *slide1*.
- 4 On the fourth line, enter **vslide**. On the fifth line, enter **vslide *slide3**. On the sixth line, enter **delay 2000**.
The fourth, fifth, and sixth lines display *slide2*, preload *slide3*, and specify a delay for viewing *slide2*.
- 5 On the seventh line, enter **vslide**. On the eighth line, enter **delay 3000**.
The seventh and eighth lines display *slide3* and specify a delay for viewing *slide3*.

- 6 On the last line, enter **rscript** to repeat the script.
- 7 To stop a repeating script press ESC. To continue the script, enter **resume**.

The script may contain comments, as follows:

```
; Begin slide show, load SLIDE1
VSLIDE SLIDE1
; Preload SLIDE2
VSLIDE *SLIDE2
; Let audience view SLIDE1
DELAY 2000
; Display SLIDE2
VSLIDE
; Preload SLIDE3
VSLIDE *SLIDE3
; Let audience view SLIDE2
DELAY 2000
; Display SLIDE3
VSLIDE
; Let audience view SLIDE3
DELAY 3000
; Cycle
RSCRIPT
```

Introduction to Programming Interfaces

7

In addition to its command line and menu interfaces, AutoCAD provides application programming interfaces (APIs) that you can use to control drawings and databases. This chapter provides an overview of the API features in AutoCAD.

The programming interfaces introduced here are ActiveX Automation, VBA (Visual Basic for Applications), AutoLISP, Visual LISP, and ObjectARX . The type of interface you use depends on your application needs and programming experience.

In this chapter

- ActiveX Automation
- AutoCAD VBA
- AutoLISP and Visual LISP
- Object ARX

ActiveX Automation

ActiveX® Automation is a technology developed by Microsoft® and based on the COM (component object model) architecture. You can use it to customize AutoCAD, share your drawing data with other applications, and automate tasks.

Overview of ActiveX

Through Automation, AutoCAD® exposes programmable objects, described by the AutoCAD Object Model, that can be created, edited, and manipulated by other applications. Any application that can access the AutoCAD Object Model is an Automation controller, and the most common tool used for manipulating another application using Automation is Visual Basic for Applications (VBA). VBA is found as a component in many Microsoft Office applications. You can use these applications, or other Automation controllers, such as Visual Basic 6 and Delphi, to drive AutoCAD.

The advantage of implementing an ActiveX interface for AutoCAD is twofold:

- Programmatic access to AutoCAD drawings is opened up to many more programming environments. Before ActiveX Automation, developers were limited to an AutoLISP or C++ interface.
- Sharing data with other Windows applications, such as Microsoft Excel and Microsoft Word, is made dramatically easier.

For detailed information about using VBA to control AutoCAD ActiveX Automation, see the *ActiveX and VBA Developer's Guide* and *ActiveX and VBA Reference* in the Help system. On the Help menu, click Developer Help.

For example, you might want to prompt for input, set preferences, make a selection set, or retrieve drawing data. You can decide on the controller to use, depending on the type of manipulation.

Using Automation, you can create and manipulate AutoCAD objects from any application that serves as an Automation controller. Thus, Automation enables macro programming across applications, a capability that does not exist in AutoLISP®. With Automation you can combine the features of many applications into a single application.

The displayed objects are called *Automation objects*. Automation objects make methods and properties available. *Methods* are functions that perform an action on an object. *Properties* are functions that set or return information about the state of an object.

Virtually any type of application can access the displayed Automation objects within AutoCAD. These applications can be stand-alone executables, dynamic linked library (DLL) files, and macros within applications such as Microsoft Word or Microsoft Excel. The most common of these is most likely the stand-alone executable. If you are using applications from application developers, follow their instructions for installation and use of their product.

See Also

ActiveX and VBA Developer's Guide
ActiveX and VBA Reference

Define a Command to Start Your Application

You can use the *acad.pgp* file to define a new AutoCAD command that runs an external command to start your application. The following example defines the RUNAPP1 command, which runs the application *app1.exe* in the *c:\vbapps* directory. (Add this code to the external commands section of your *acad.pgp* file.)

```
RUNAPP1, start c:\vbapps\app1, 0
```

If your application requires command line parameters, you can use the following code:

```
RUNAPP2, start c:\vbapps\app2, 0, *Parameters: ,
```

This example defines the RUNAPP2 command, which prompts you for parameters and then passes them to your application.

You can also use the AutoLISP **startapp** function to start an application that makes use of Automation. Once AutoLISP starts the external application, it has no control over its actions. You can, however, use AutoLISP to locate and run different applications based on certain parameters.

Start an Application from a Menu

After defining a new command to start your application, you can make that command available from a menu or toolbar.

The menu macro can be called from a menu item in any section of the menu (MNU) file. If you use only one or two applications, you can add them to one of the standard pull-down menus. If you have a group of applications, you can add your own pull-down menu or toolbar that is specifically dedicated to those applications. For information about creating, editing, and loading menu files, see “The Menu File” on page 45.

The following example is a complete menu file that defines a new pull-down menu called MyApps. The menu file defines the MYAPPS menu group and two groups of menu items. The first three menu items use the AutoLISP **startapp** function to start the associated application, and the last two items assume that the commands APP4 and APP5 have been defined (in the *acad.pgp* file).

This file also includes a Helpstrings section, which provides status line Help when the menu item is selected. The Accelerators section defines accelerator keys that run each of the menu macros. These menu sections use name tags (such as ID_App1) to link their associated actions.

```
// MYAPPS Menu
//
***MENUGROUP=MYAPPS

// This section defines the new pull-down menu
***POP1
ID_MyAppsMnu [My&Apps]
ID_MyApp1    [App1    ] ^C^C^P(startapp "app1")(princ) ^P
ID_MyApp2    [App2    ] ^C^C^P(startapp "app2")(princ) ^P
ID_MyApp3    [App3    ] ^C^C^P(startapp "app3")(princ) ^P
              [--]
ID_MyApp4    [App4    ] ^C^CAPP4
ID_MyApp5    [App5    ] ^C^CAPP5

// This section defines the status line messages that are
// displayed when an associated menu item is highlighted
***HELPSTRINGS
ID_MyApp1    [This is APP1]
ID_MyApp2    [This is APP2]
ID_MyApp3    [This is APP3]
ID_MyApp4    [This is APP4]
ID_MyApp5    [This is APP5]

// This section defines the accelerator keys that run
// the menu macros in the related pull-down menus
***ACCELERATORS
ID_MyApp1    [CONTROL+SHIFT+"1"]
ID_MyApp2    [CONTROL+SHIFT+"2"]
ID_MyApp3    [CONTROL+SHIFT+"3"]
ID_MyApp4    [CONTROL+SHIFT+"4"]
ID_MyApp5    [CONTROL+SHIFT+"5"]
```

After saving this code to a file named *myapps.mnu*, use the `MENULOAD` command to add the menu to the menu bar. When loading the file the first time, you need to specify the MNU file type (*.mnu*).

See Also

“The Menu File” on page 45

AutoCAD VBA

Microsoft Visual Basic for Applications (VBA) is an object-based programming environment designed to provide rich development capabilities. The main difference between VBA and VB (Visual Basic 6) is that VBA runs in the same process space as AutoCAD, providing an AutoCAD-intelligent and very fast programming environment.

Overview of AutoCAD VBA

VBA provides application integration with other VBA-enabled applications. This means that AutoCAD, using other application object libraries, can be an Automation controller for other applications such as Microsoft Word or Excel.

The stand-alone development editions of Visual Basic 6, which must be purchased separately, complement AutoCAD VBA with additional components such as an external database engine and report-writing capabilities.

Develop with AutoCAD VBA

VBA sends messages to AutoCAD by the AutoCAD ActiveX Automation Interface. AutoCAD VBA permits the Visual Basic environment to run simultaneously with AutoCAD and provides programmatic control of AutoCAD through the ActiveX Automation Interface. This linking of AutoCAD, ActiveX Automation, and VBA provides an extremely powerful interface. It not only controls AutoCAD objects, but it also sends data to or retrieves data from other applications.

The integration of VBA into AutoCAD provides an easy-to-use visual tool for customizing AutoCAD. For example, you can create an application that extracts attribute information automatically, inserts the results directly into an Excel spreadsheet, and performs any data transformations you need.

Three fundamental elements define VBA programming in AutoCAD. The first is AutoCAD itself, which has a rich set of objects that include AutoCAD entities, data, and commands. AutoCAD is an open-architecture application with multiple levels of interface. To use VBA effectively, familiarity with AutoCAD programmability is highly desirable. However, you will find that the VBA object-based approach is quite different from that of AutoLISP.

The second element is the AutoCAD ActiveX Automation Interface, which establishes messages (communication) with AutoCAD objects. Programming in VBA requires a fundamental understanding of ActiveX Automation. A description of the AutoCAD ActiveX Automation Interface can be found in the *ActiveX and VBA Developer's Guide* (on the Help menu in AutoCAD, click Developer Help).

The third element that defines VBA programming is VBA itself. It has its own set of objects, keywords, constants, and so forth, that provide program flow, control, debugging, and execution. The Microsoft extensive Help system for VBA is included with AutoCAD VBA.

The AutoCAD ActiveX/VBA interface provides several advantages over other AutoCAD API environments:

- *Speed.* Running in-process with VBA, ActiveX applications are faster than AutoLISP applications.
- *Ease of use.* The programming language and development environment are easy-to-use and come installed with AutoCAD.
- *Windows interoperability.* ActiveX and VBA are designed to be used with other Windows applications and provide an excellent path for communication of information across applications.
- *Rapid prototyping.* The rapid interface development of VBA provides the perfect environment for prototyping applications, even if those applications will be developed eventually in another language.
- *Programmer base.* Programmers already use Visual Basic 6. AutoCAD ActiveX/VBA opens up AutoCAD customization and application development to these programmers as well as those who will learn Visual Basic 6 in the future.

Use AutoCAD VBA Applications

Although Microsoft Office 97 applications store VBA projects, macros, and programs inside a specific document, AutoCAD uses a separate file with the *.dvb* extension. In this way, VBA interfaces with AutoCAD in much the same way that AutoLISP and ObjectARX do. Because VBA projects are stored in a separate file, a VBA project can open and close different AutoCAD drawings during an AutoCAD session.

Note AutoCAD VBA projects are not binary compatible with stand-alone Visual Basic 6 projects (VBP files). However, forms, modules, and classes can be exchanged between dissimilar projects using the IMPORT and EXPORT VBA commands in the VBA integrated development environment (IDE).

You load a VBA project with the VBALOAD command. Once loaded, its modules and macros are available in the Macros dialog box. To run the VBA module you use the VBARUN command. If no VBA project is loaded, the options are unavailable. Procedures listed in the Macro Name box use the following syntax:

module.macro

In the Macros dialog box you choose the Macro Scope and select from the listed modules.

Use the Command Line to Run a VBA Macro

AutoCAD command line prompt equivalents are available using the -VBARUN command (signified by a hyphen in front of the VBARUN command). You can run VBA macros from the command line, scripts, and other AutoCAD programming environments. The only argument for the command is the module name using the *module.macro* syntax. The syntax looks like this:

`-vbarun <module.macro>`

Because macros with the same name can be duplicated in modules, the *module.macro* syntax differentiates the macro and allows for unique selection.

Automatically Load and Execute VBA Projects

As you build up a number of VBA projects, you can load them automatically each time you run AutoCAD. The macros they contain are immediately available. Additionally, the APPLOAD command provides a Startup Suite option that automatically loads the specified applications.

acvba.arx — Automatically Load VBA

You cannot load VBA until an AutoCAD VBA command is issued. If you want to load VBA automatically every time you start AutoCAD include the following line in the *acad.rx* file:

`acvba.arx`

You can automatically run a macro in the *acad.dvb* file by naming the macro AcadStartup. Any macro in your *acad.dvb* file called AcadStartup automatically executes when VBA loads.

acad.dvb — Automatically Load a VBA Project

The *acad.dvb* file is useful if you want to load a specific VBA project that contains macros you want each time you start AutoCAD. Each time you start a new AutoCAD drawing session, AutoCAD searches for the *acad.dvb* file and loads it.

If you want a macro in your *acad.dvb* file to run each time you start a new drawing or open an existing one, add the following code to your *acaddoc.lsp* file:

```
(defun S::STARTUP()  
  (command "_-vbarun" "updatetitleblock")  
)
```

The project name in the example is *updatetitleblock*.

AutoLISP and Visual LISP

AutoLISP is based on the LISP programming language, which is simple to learn and very powerful. Because AutoCAD has a built-in LISP interpreter, you can enter AutoLISP code on the command line or load AutoLISP code from external files. Visual LISP (VLISP) is a software tool designed to expedite AutoLISP program development.

Overview of AutoLISP and Visual LISP

AutoLISP has been enhanced with Visual LISP (VLISP), which offers an integrated development environment (IDE) that includes a compiler, debugger, and other development tools to increase productivity. VLISP adds more capabilities and extends the language to interact with objects using ActiveX. VLISP also enables AutoLISP to respond to events through object reactors.

Unlike in ObjectARX, or VBA, each document opened in the Multiple Design Environment (MDE) has its own Visual LISP *namespace* and environment. A namespace is an insulated environment keeping AutoLISP routines that are specific to one document from having symbol or variable name and value conflicts with those in another document. For example, the following line of code sets a different value to the symbol *a* for different documents.

```
(setq a (getvar "DWGNAME"))
```

Visual LISP provides mechanisms for loading symbols and variables from one namespace to another. More information about namespaces can be found in the *AutoLISP Developer's Guide* (on the Help menu in AutoCAD, click Developer Help).

AutoLISP applications or routines can interact with AutoCAD in many ways. These routines can prompt the user for input, access built-in AutoCAD commands directly, and modify or create objects in the drawing database. By creating AutoLISP routines you can add discipline-specific commands to AutoCAD. Some of the standard AutoCAD commands are actually AutoLISP applications.

Visual LISP provides three file format options for AutoLISP applications:

- Reading an LSP file (*.lsp*)—an ASCII text file that contains AutoLISP program code.
- Reading an FAS file (*.fas*)—a binary, compiled version of a single LSP program file.
- Reading a VLX file (*.vlx*)—a compiled set of one or more LSP and/or dialog control language (DCL) files.

Note Like-named AutoLISP application files are loaded based on their Modified time stamp; the LSP, FAS, or VLX file with the most recent time stamp is loaded unless you specify the full file name (including the file name extension).

Because AutoCAD can read AutoLISP code directly, no compiling is required. While Visual LISP provides an IDE, you may choose to experiment by entering code at the Command prompt, which allows you to see the results immediately. This makes AutoLISP an easy language to experiment with, regardless of your programming experience.

Even if you are not interested in writing AutoLISP applications, your AutoCAD package includes many useful routines. Routines are also available as shareware through third-party developers. Knowing how to load and use these routines can enhance your productivity.

Note When command input comes from the AutoLISP **command** function, the settings of the PICKADD and PICKAUTO system variables are assumed to be 1 and 0, respectively. This preserves compatibility with previous releases of AutoCAD and makes customization easier (because you don't have to check the settings of these variables).

For information about AutoLISP programming, see the *AutoLISP Developer's Guide*, and for information about AutoLISP and Visual LISP functions, see the *AutoLISP Reference* (on the Help menu in AutoCAD, click Developer Help). AutoLISP programs can use dialog boxes with their applications. Programmable dialog boxes are described only in the *AutoLISP Developer's Guide*.

Use AutoLISP Applications

AutoLISP applications are stored in ASCII text files with the *.lsp* extension. These files generally have a header portion that describes a routine, its use, and any specific instructions. This header might also include comments that document the author and the legal information regarding the use of the routine. Comments are preceded by a semicolon (;). You can view and edit these files with a text editor or word processor that can produce an ASCII text file.

Before you can use an AutoLISP application, it must first be loaded. You can use the APPLOAD command or the AutoLISP **load** function to load an application. Loading an AutoLISP application loads the AutoLISP code from the LSP file into your system's memory.

Loading an application with the **load** function involves entering AutoLISP code at the Command prompt. If the **load** function is successful, it displays the value of the last expression in the file on the command line. This is usually the name of the last function defined in the file or instructions on using the newly loaded function. If **load** fails, it returns an AutoLISP error message. A **load** failure can be caused by incorrect coding in the file or by entering the wrong file name on the command line. The syntax for the **load** function is

```
(load filename [onfailure])
```

This syntax shows that the load function has two arguments: *filename*, which is required, and *onfailure*, which is optional. When loading an AutoLISP file on the command line, you typically supply only the *filename* argument. The following example loads the AutoLISP file *newfile.lsp*.

Command: **(load "newfile")**

The *.lsp* extension is not required. This format works for any LSP file in the current library path.

To load an AutoLISP file that is not in the library path, you must provide the full path and file name as the *filename* argument.

Command: **(load "d:/files/morelisp/newfile")**

Note When specifying a directory path, you must use a slash (/) or two backslashes (\\) as the separator, because a single backslash has a special meaning in AutoLISP.

See Also

“Overview of File Organization” on page 4

Automatically Load and Run AutoLISP Routines

You can load AutoLISP routines each time you run AutoCAD. You can also execute certain commands or functions at specific times during a drawing session.

Overview of AutoLISP Automatic Loading

AutoCAD loads the contents of three user-definable files automatically: *acad.lsp*, *acaddoc.lsp*, and the MNL file that accompanies your current menu. By default, the *acad.lsp* file is loaded only once, when AutoCAD starts, whereas *acaddoc.lsp* is loaded with each individual document (or drawing). This lets you associate the loading of the *acad.lsp* file with application startup, and the *acaddoc.lsp* file with document (or drawing) startup. The default method for loading these startup files can be modified by changing the setting of the ACADLSPASDOC system variable.

If one of these files defines a function of the special type **S::STARTUP**, this routine runs immediately after the drawing is fully initialized. The **S::STARTUP** function is described in “S::STARTUP Function: Postinitialization Execution” on page 143. As an alternative, the APPLOAD command provides a Startup Suite option that loads the specified applications without the need to edit any files.

The *acad.lsp* and *acaddoc.lsp* startup files are not provided with AutoCAD. It is up to the user to create and maintain these files.

Command Autoloader

When you automatically load a command using the **load** or **command** functions, the command’s definition takes up memory whether or not you actually use the command. The AutoLISP **autoload** function makes a command available without loading the entire routine into memory. Adding the following code to your *acaddoc.lsp* file automatically loads the commands CMD1, CMD2, and CMD3 from the *cmds.lsp* file and the NEWCMD command from the *newcmd.lsp* file.

```
(autoload "CMDS" '("CMD1" "CMD2" "CMD3"))  
(autoload "NEWCMD" '("NEWCMD"))
```

The first time you enter an automatically loaded command at the Command prompt, AutoLISP loads the entire command definition from the associated file. AutoLISP also provides the **autoarxload** function for ObjectARX applications. See **autoload** and **autoarxload** in the *AutoLISP Reference* (on the Help menu in AutoCAD, click Developer Help).

Note Like-named AutoLISP startup files are loaded based on their Modified time stamp; the LSP file with the most recent time stamp is loaded unless you specify the full file name (including the file name extension).

See Also

“Load and Unload Menu Files” on page 50

“S::STARTUP Function: Postinitialization Execution” on page 143

The *acad.lsp* File

You can create an *acad.lsp* file if you regularly use specific AutoLISP routines. When you start AutoCAD, it searches the support file search path for an *acad.lsp* file. If an *acad.lsp* file is found, it is loaded into memory.

The *acad.lsp* file is loaded at each drawing session startup when AutoCAD is launched from the Windows desktop. Because the *acad.lsp* file is intended to be used for application-specific startup routines, all functions and variables defined in an *acad.lsp* file are only available in the first drawing. You will probably want to move routines that should be available in all documents from your *acad.lsp* file into the *acaddoc.lsp* file.

The recommended functionality of *acad.lsp* and *acaddoc.lsp* can be overridden with the ACADLSPASDOC system variable. If the ACADLSPASDOC system variable is set to 0 (the default setting), the *acad.lsp* file is loaded just once: upon application startup. If ACADLSPASDOC is set to 1, the *acad.lsp* file is reloaded with each new drawing.

The ACADLSPASDOC system variable is ignored in SDI (single document interface) mode. When the SDI system variable is set to 1, the LISPINIT system variable controls reinitialization of AutoLISP between drawings. When LISPINIT is set to 1, AutoLISP functions and variables are valid in the current drawing only; each time you start a new drawing or open an existing one, all functions and variables are cleared from memory and the *acad.lsp* file is reloaded. Changing the value of LISPINIT when the SDI system variable is set to 0 has no effect.

The *acad.lsp* file can contain AutoLISP code for one or more routines, or just a series of **load** function calls. The latter method is preferable, because modification is easier. If you save the following code as an *acad.lsp* file, the files *mysessionapp1.lsp*, *databasesynch.lsp*, and *drawingmanager.lsp* are loaded every time you start AutoCAD.

```
(load "mysessionapp1")  
(load "databasesynch")  
(load "drawingmanager")
```

Warning! Do not modify the reserved *acad2004.lsp* file. Autodesk provides the *acad2004.lsp* file, which contains AutoLISP defined functions that are required by AutoCAD. This file is loaded into memory immediately before the *acad.lsp* file is loaded.

See Also

“Overview of File Organization” on page 4

“Prevent AutoLISP Errors When Loading Startup Files” on page 143

The *acaddoc.lsp* File

Each time a drawing opens, AutoCAD searches the library path for an *acaddoc.lsp* file. If it finds one, it loads the file into memory. The *acaddoc.lsp* file is always loaded with each drawing regardless of the settings of ACADLSPASDOC and LISPINIT.

Most users will have a single *acaddoc.lsp* file for all document-based AutoLISP routines. AutoCAD searches for an *acaddoc.lsp* file in the order defined by the library path; therefore, with this feature, you can have a different *acaddoc.lsp* file in each drawing directory, which would load specific AutoLISP routines for certain types of drawings or jobs.

The *acaddoc.lsp* file can contain AutoLISP code for one or more routines, or just a series of **load** function calls. The latter method is preferable, because modification is easier. If you save the following code as an *acaddoc.lsp* file, the files *mydocumentapp1.lsp*, *build.lsp*, and *counter.lsp* are loaded every time a new document is opened.

```
(load "mydocumentapp1")  
(load "build")  
(load "counter")
```

AutoCAD searches for an *acaddoc.lsp* file in the order defined by the library path; therefore, you can have a different *acaddoc.lsp* file in each drawing directory. You can then load specific AutoLISP routines for certain types of drawings or jobs.

Warning! Do not modify the reserved *acad2004doc.lsp* file. Autodesk provides the *acad2004doc.lsp* file, which contains AutoLISP-defined functions that are required by AutoCAD. This file is loaded into memory immediately before the *acaddoc.lsp* file is loaded.

See Also

“Overview of File Organization” on page 4

“Prevent AutoLISP Errors When Loading Startup Files” on page 143

The MNL File for an AutoLISP Menu

The other type of file that AutoCAD loads automatically accompanies your current menu file and has the extension *.mnl*. When AutoCAD loads a menu file, it searches for an MNL file with a matching file name. If it finds the file, it loads the file into memory.

This function ensures that AutoCAD loads the AutoLISP functions that are needed for proper operation of a menu. For example, the default AutoCAD menu, *acad.mnu*, relies on the file *acad.mnl*. This file defines numerous AutoLISP functions used by the menu. The MNL file is loaded after the *acad-doc.lsp* file.

Note If a menu file is loaded with the AutoLISP **command** function—with syntax similar to (`command "menu" "newmenu"`)—the associated MNL file is not loaded until the entire AutoLISP routine has run.

For example, if you create a custom menu called *newmenu.mnu* and you need to load three AutoLISP files (*new1.lsp*, *new2.lsp*, and *new3.lsp*) for the menu to work properly, you should create an ASCII text file named *newmenu.mnl* as follows:

```
(load "new1")
(load "new2")
(load "new3")
(princ "\nNewmenu utilities... Loaded.")
(princ)
```

In this example, calls to the **princ** function can be used to display status messages. The first use of **princ** displays the following on the command line:

Newmenu utilities..Loaded.

The second call to **princ** exits the AutoLISP function. Without this second call to **princ**, the message would be displayed twice. As mentioned previously, you can include the *onfailure* argument with calls to the **load** function as an extra precaution.

Prevent AutoLISP Errors When Loading Startup Files

If an AutoLISP error occurs while you are loading a startup file, the remainder of the file is ignored and is not loaded.

Files specified in a startup file that do not exist or that are not in the AutoCAD library path generally cause errors. Therefore, you may want to use the *onfailure* argument with the **load** function. The following example uses the *onfailure* argument:

```
(princ (load "mydocapp1" "\nMYDOCAPP1.LSP file not loaded."))
(princ (load "build" "\nBUILD.LSP file not loaded."))
(princ (load "counter" "\nCOUNTER.LSP file not loaded."))
(princ)
```

If a call to the **load** function is successful, it returns the value of the last expression in the file (usually the name of the last defined function or a message regarding the use of the function). If the call fails, it returns the value of the *onfailure* argument. In the preceding example, the value returned by the **load** function is passed to the **princ** function, causing that value to be displayed on the command line.

For example, if an error occurs while AutoCAD loads the *mydocapp1.lsp* file, the **princ** function displays the following message and AutoCAD continues to load the two remaining files:

MYDOCAPP1.LSP file not loaded.

If you use the **command** function in an *acad.lsp*, *acaddoc.lsp*, or MNL file, it should be called only from within a **defun** statement. Use the **S::STARTUP** function to define commands that need to be issued immediately when you begin a drawing session.

See Also

“S::STARTUP Function: Postinitialization Execution” on page 143

S::STARTUP Function: Postinitialization Execution

The startup LISP files (*acad.lsp*, *acaddoc.lsp*, and MNL) are all loaded into memory before the drawing is completely initialized. Typically, this does not pose a problem, unless you want to use the **command** function, which is not guaranteed to work until after a drawing is initialized.

If the user-defined function **S::STARTUP** is included in an *acad.lsp*, *acad-doc.lsp*, or MNL file, it is called when you enter a new drawing or open an existing drawing. Thus, you can include a definition of **S::STARTUP** in the LISP startup file to perform any setup operations.

For example, if you want to override the standard HATCH command by adding a message and then switching to the BHATCH command, use an *acad-doc.lsp* file that contains the following:

```
(defun C:HATCH ( )
  (alert "Using the BHATCH command!")
  (princ "\nEnter OLDHATCH to get to real HATCH command.\n")
  (command "BHATCH")
  (princ)
)
(defun C:OLDHATCH ( )
  (command ".HATCH")
  (princ)
)
(defun-q S::STARTUP ( )
  (command "undefine" "hatch")
  (princ "\nRedefined HATCH to BHATCH!\n")
)
```

Before the drawing is initialized, new definitions for HATCH and OLDHATCH are defined with the **defun** function. After the drawing is initialized, the **S::STARTUP** function is called and the standard definition of HATCH is undefined.

Note To be appended, the **S::STARTUP** function must have been defined with the **defun-q** function rather than **defun**.

Because an **S::STARTUP** function can be defined in many places (an *acad.lsp*, *acaddoc.lsp*, or MNL file or any other AutoLISP file loaded from any of these), it's possible to overwrite a previously defined **S::STARTUP** function. The following example shows one method of ensuring that your startup function works with other functions.

```
(defun-q MYSTARTUP ( )
  ... your startup function ...
)
(setq S::STARTUP (append S::STARTUP MYSTARTUP))
```

The previous code appends your startup function to that of an existing **S::STARTUP** function and then redefines the **S::STARTUP** function to include your startup code. This works properly regardless of the prior existence of an **S::STARTUP** function.

Object ARX

ObjectARX technology provides the foundation for design software applications to share intelligent object data. You can run third-party ObjectARX application programs or write your own.

Overview of ObjectARX

ObjectARX® (AutoCAD Runtime Extension) is a compiled-language programming environment for developing AutoCAD applications. The ObjectARX programming environment includes a number of dynamic link libraries (DLLs) that run in the same address space as AutoCAD and operate directly with core AutoCAD data structures and code. These libraries take advantage of the AutoCAD open architecture, providing direct access to the AutoCAD database structures, graphics system, and AutoCAD geometry engine to extend AutoCAD classes and capabilities at runtime. Additionally, you can use DLLs to create new commands that operate exactly the same way as native AutoCAD commands.

You can use ObjectARX libraries in conjunction with other AutoCAD programming interfaces, such as AutoLISP or VBA, enabling cross-API integration.

The ObjectARX programming environment is described in the *ObjectARX Developer's Guide*. The documentation is part of the ObjectARX Software Development Kit, which can be downloaded from the Development Tools section of the Autodesk website. For more information, click Developer Help on the Help menu and then click ObjectARX.

Use ObjectARX Applications

To load an ObjectARX application, you use the Load option of the ARX command. After loading, all commands defined by this application are available at the Command prompt.

Some ObjectARX applications use large amounts of system memory. If you are finished using an application and want to remove it from memory, use the Unload option of ARX.

You can also load an ObjectARX application with the **arxload** AutoLISP function. The syntax for the **arxload** function is almost identical to that of the **load** function used with AutoLISP files. If the **arxload** function loads the ObjectARX program successfully, it returns the program name. The syntax for the **arxload** function is as follows:

```
(arxload filename [onfailure])
```

The two arguments for the **arxload** function are *filename* and *onfailure*. As with the **load** function, the *filename* argument is required and must be the complete path name description of the ObjectARX program file to load. The *onfailure* argument is optional and typically not used when you load ObjectARX programs from the command line. The following example loads the ObjectARX application *myapp.arx*.

```
(arxload "myapp")
```

As with AutoLISP files, AutoCAD searches the library path for the specified file. If you need to load a file that is not in the library path, you must provide the full path name description of the file.

Note When specifying a directory path, you must use a slash (/) or two backslashes (\\) as the separator, because a single backslash has a special meaning in AutoLISP.

Attempting to load an application that has previously been loaded results in an error. Before using **arxload** you should use the **arx** function to check the currently loaded applications.

To unload an application with AutoLISP, use the **arxunload** function. The following example unloads the *myapp* application.

```
(arxunload "myapp")
```

Using the **arxunload** function not only removes the application from memory but also removes the command definitions associated with that application.

See Also

“Overview of File Organization” on page 4

Automatically Load ObjectARX Applications

Some ObjectARX samples contain an *acad.rx* file, which lists ObjectARX program files that are loaded automatically when you start AutoCAD.

You can create or edit this file with a text editor or word processor that produces files in ASCII text format, adding to or deleting from its contents to make the appropriate ObjectARX programs available for use. As an alternative, the APPLOAD command provides a Startup Suite option that loads the specified applications without the need to edit any files.

Because AutoCAD searches for the *acad.rx* file in the order specified by the library path, you can have a different *acad.rx* file in each drawing directory. This makes specific ObjectARX programs available for certain types of drawings. For example, you might keep 3D drawings in a directory called *Acad-Jobs/3d_dwgs*. If that directory is set up as the current directory, you could copy the *acad.rx* file into that directory and modify it in the following manner:

```
myapp1  
otherapp
```

If you place this new *acad.rx* file in the *AcadJobs/3d_dwgs* directory and you start AutoCAD with that as the current directory, these new ObjectARX programs are then loaded and are available from the AutoCAD command line. Because the original *acad.rx* file is still in the directory with the AutoCAD program files, the default *acad.rx* file will be loaded if you start AutoCAD from another directory that does not contain an *acad.rx* file.

You can load ObjectARX programs from an MNL file using the **arxload** function. This ensures that an ObjectARX program, required for proper operation of a menu, will be loaded when the menu file is loaded.

You can also autoload many ObjectARX-defined AutoCAD commands. See “Overview of AutoLISP Automatic Loading” on page 139 and **autoarxload** in the *AutoLISP Reference* (on the Help menu in AutoCAD, click Developer Help).

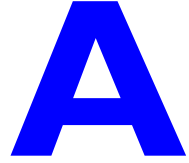
See Also

“Overview of AutoLISP Automatic Loading” on page 139

ASCII Codes

This chart lists standard ASCII codes. The octal value is useful in character or string constants, using the `\nnn` form. Your system might define additional codes in the extended, 256-character set (additional codes are greater than 127). Some systems also redefine some of the little-used ASCII codes such as 1–6 and 14–26.

To see your system's characters with their codes in both decimal and octal form, use the AutoLISP-defined command `ASCII` (described in “ASCII Code Conversion” in the *AutoLISP Developer's Guide*). The characters are displayed in a table on the screen and stored in a file called *ascii.txt*.



In this appendix

- ASCII Code Conversion Chart

ASCII code conversion chart

Dec.	Oct.	Hex.	Character
0	000	00	NUL
1	001	01	SOH
2	002	02	STX
3	003	03	ETX
4	004	04	EOT
5	005	05	ENQ
6	006	06	ACK
7	007	07	BEL (bell)
8	010	08	BS (backspace)
9	011	09	HT
10	012	0A	LF (line-feed)
11	013	0B	VT
12	014	0C	FF
13	015	0D	CR (return)
14	016	0E	SO
15	017	0F	SI
16	020	10	DLE
17	021	11	DC1
18	022	12	DC2
19	023	13	DC3
20	024	14	DC4
21	025	15	NAK
22	026	16	SYN

Dec.	Oct.	Hex.	Character
23	027	17	ETB
24	030	18	CAN
25	031	19	EM
26	032	1A	SUB
27	033	1B	ESC (escape)
28	034	1C	FS
29	035	1D	GS
30	036	1E	RS
31	037	1F	US
32	040	20	(space)
33	041	21	!
34	042	22	"
35	043	23	#
36	044	24	\$
37	045	25	%
38	046	26	&
39	047	27	'
40	050	28	(
41	051	29	)
42	052	2A	*
43	053	2B	+
44	054	2C	,
45	055	2D	-

ASCII code conversion chart (continued)

Dec.	Oct.	Hex.	Character
46	056	2E	.
47	057	2F	/
48	060	30	0
49	061	31	1
50	062	32	2
51	063	33	3
52	064	34	4
53	065	35	5
54	066	36	6
55	067	37	7
56	070	38	8
57	071	39	9
58	072	3A	:
59	073	3B	;
60	074	3C	<
61	075	3D	=
62	076	3E	>
63	077	3F	?
64	100	40	@
65	101	41	A
66	102	42	B
67	103	43	C
68	104	44	D

Dec.	Oct.	Hex.	Character
69	105	45	E
70	106	46	F
71	107	47	G
72	110	48	H
73	111	49	I
74	112	4A	J
75	113	4B	K
76	114	4C	L
77	115	4D	M
78	116	4E	N
79	117	4F	O
80	120	50	P
81	121	51	Q
82	122	52	R
83	123	53	S
84	124	54	T
85	125	55	U
86	126	56	V
87	127	57	W
88	130	58	X
89	131	59	Y
90	132	5A	Z
91	133	5B	[

ASCII code conversion chart (continued)

Dec.	Oct.	Hex.	Character
92	134	5C	\
93	135	5D	]
94	136	5E	^
95	137	5F	_
96	140	60	'
97	141	61	a
98	142	62	b
99	143	63	c
100	144	64	d
101	145	65	e
102	146	66	f
103	147	67	g
104	150	68	h
105	151	69	i
106	152	6A	j
107	153	6B	k
108	154	6C	l
109	155	6D	m
110	156	6E	n
111	157	6F	o
112	160	70	p
113	161	71	q
114	162	72	r

Dec.	Oct.	Hex.	Character
115	163	73	s
116	164	74	t
117	165	75	u
118	166	76	v
119	167	77	w
120	170	78	x
121	171	79	y
122	172	7A	z
123	173	7B	{
124	174	7C	
125	175	7D	}
126	176	7E	~
127	177	7F	DEL (delete)

Shapes and Shape Fonts

With AutoCAD, you can define shapes to use as drawing symbols and text fonts. This appendix explains how to create and compile your own shape and font files.



In this appendix

- Overview of Shape Files
- Create Shape Definition Files

Overview of Shape Files

Shapes are objects that you use like blocks. First you use the LOAD command to load the compiled shape file containing the shape definition. Then you use the SHAPE command to insert shapes from the file into your drawing. You can specify the scale and rotation to use for each shape as you add it. AutoCAD® SHP fonts are a special type of shape file, and are defined in the same way as shape files.

Blocks are more versatile and easier to use and apply than shapes. However, shapes are more efficient for AutoCAD to store and draw. User-defined shapes are helpful when you must insert a simple part many times and when speed is important.

Compile Shape/Font Files

You enter the description of shapes in a specially formatted text file with a file extension of *.shp*. To create the file, use a text editor or word processor that enables you to save in ASCII format, and then compile the ASCII file. Compiling a shape definition file (SHP) generates a compiled shape file (SHX).

The compiled file has the same name as the shape definition file but with a file type of SHX. If the shape definition file defines a font, you use the STYLE command to define a text style. Then, you use one of the text placement commands (TEXT or MTEXT) to place the characters in the drawing. If the shape definition file defines shapes, you use the LOAD command to load the shape file into the drawing. Then, you use the SHAPE command to place the individual shapes in the drawing (similar in concept to the INSERT command).

Compile PostScript Fonts

To use a Type 1 PostScript® font in AutoCAD you must first compile it into an AutoCAD shape file. The COMPILE command accepts both SHP and PFB files as input and generates an SHX file. Compiled versions of PostScript fonts can take a lot of disk space, so compile only those fonts you use frequently.

AutoCAD cannot compile and load every Type 1 font. The PostScript font facilities in AutoCAD are intended to process a subset of Adobe fonts. If you receive an error while compiling a PostScript font, the resulting SHX file (if one is generated) may not load into AutoCAD.

For more information on the Adobe Type 1 font format, refer to *Adobe Type 1 Font Format Version 1.1*. When you've purchased and installed these fonts, you can begin using them with AutoCAD.

Note Make sure you understand any copyright that accompanies the PostScript fonts you use. The same copyright restrictions generally apply to the SHX form of fonts you've compiled.

To compile a shape or font file

- On the Command line, enter **compile**.

In the Select Shape File dialog box, you can select a shape definition file (SHP) or PostScript font file (PFB). After you select the file name, compiling begins. If AutoCAD finds an error in the shape descriptions, a message is displayed telling you the type of error and the line number. When compiling is complete, the following messages are displayed:

Compilation successful.

Output file *name.shx* contains *nnn* bytes.

Create Shape Definition Files

AutoCAD font and shape files (SHX) are compiled from shape definition files (SHP). You can create or modify shape definition files with a text editor or word processor that saves files in ASCII format.

Shape Descriptions

The syntax of the shape description for each shape or character is the same regardless of the final use (shape or font) for that shape description. If a shape definition file is to be used as a font file, the first entry in the file describes the font itself rather than a shape within the file. If this initial entry describes a shape, the file is used as a shape file.

Being able to create your own shape definitions is a valuable skill. Keep in mind, however, that this is a very complex subject to learn and requires patience.

Each line in a shape definition file can contain up to 128 characters. Longer lines cannot be compiled. Because AutoCAD ignores blank lines and text to the right of a semicolon, you can embed comments in shape definition files.

Each shape description has a header line of the following form and is followed by one or more lines containing specification bytes, separated by commas and terminated by a 0.

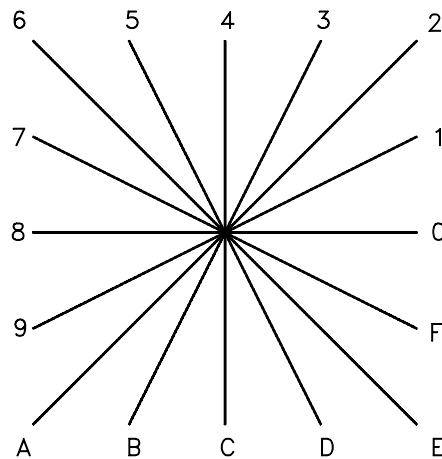
```
*shapenumber,defbytes,shapename  
specbyte1,specbyte2,specbyte3,...,0
```

The following describes the fields of a shape description:

<code>shapenumber</code>	A number, unique to the file, between 1 and 258 (and up to 32768 for Unicode fonts), and preceded by an asterisk (*). Non-Unicode font files use the shape numbers 256, 257, and 258 for the symbolic identifiers Degree_Sign, Plus_Or_Minus_Sign, and Diameter_Symbol. For Unicode fonts these glyphs appear at the U+00B0, U+00B1, and U+2205 shape numbers and are part of the “Latin Extended-A” subset. Text fonts (files containing shape definitions for each character) require specific numbers corresponding to the value of each character in the ASCII code; other shapes can be assigned any numbers.
<code>defbytes</code>	The number of data bytes (<i>specbytes</i>) required to describe the shape, including the terminating 0. The limit is 2,000 bytes per shape.
<code>shapename</code>	The shape name. Shape names must be uppercase to be recognized. Names with lowercase characters are ignored and are usually used to label font shape definitions.
<code>specbyte</code>	A shape specification byte. Each specification byte is a code that defines either a vector length and direction or one of a number of special codes. A specification byte can be expressed in the shape definition file as either a decimal or hexadecimal value. This section uses both decimal and hexadecimal specification byte values for its examples (as do many of the shape definition files). If the first character of a specification byte is a 0 (zero), the two characters that follow are interpreted as hexadecimal values.

Vector Length and Direction Code

A simple shape specification byte contains vector length and direction encoded into one specification byte (one *specbyte* field). Each vector length and direction code is a string of three characters. The first character must be a 0, which indicates to AutoCAD that the next two characters are interpreted as hexadecimal values. The second character specifies the length of the vector in units. Valid hexadecimal values range from 1 (one unit long) through F (15 units long). The third character specifies the direction of the vector. The following figure illustrates the direction codes.



Vector direction codes

All the vectors in the preceding figure were drawn with the same length specification. Diagonal vectors stretch to match the *X* or *Y* displacement of the closest orthogonal vector. This is similar to the action of the snap grid in AutoCAD.

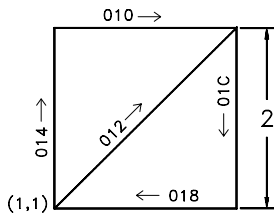
The following example constructs a shape named DBOX with an arbitrarily assigned shape number of 230.

```
*230,6,DBOX  
014,010,01C,018,012,0
```

The preceding sequence of specification bytes defines a box one unit high by one unit wide, with a diagonal line running from the lower left to the upper right. After saving the file as *dbox.shp*, use the COMPILE command to generate the *dbox.shx* file. Use the LOAD command to load the shape file containing this definition, and then use the SHAPE command as follows:

Command: **shape**
 Enter shape name or [?]: **dbox**
 Specify insertion point: **1,1**
 Specify height <current>: **2**
 Specify rotation angle <current>: **0**

The resulting shape is shown in the following illustration.



Special Codes

In addition to defining vectors, a specification byte can use special codes to create additional forms and specify certain actions. To use a special code, the second character of the three-character string (the vector length specification) must be 0, or you can specify only the code number. For example, 008 and 8 are both valid specifications.

Specification byte codes	
Code	Description
000	End of shape definition
001	Activate Draw mode (pen down)
002	Deactivate Draw mode (pen up)
003	Divide vector lengths by next byte
004	Multiply vector lengths by next byte
005	Push current location onto stack

Specification byte codes (*continued*)

Code	Description
006	Pop current location from stack
007	Draw subshape number given by next byte
008	X-Y displacement given by next two bytes
009	Multiple X-Y displacements, terminated (0,0)
00A	Octant arc defined by next two bytes
00B	Fractional arc defined by next five bytes
00C	Arc defined by X-Y displacement and bulge
00D	Multiple bulge-specified arcs
00E	Process next command only if vertical text

Codes 0, 1, and 2: End of Shape and Draw Mode Control

Code 0 marks the end of the shape definition.

Codes 1 and 2 control Draw mode. Draw is activated at the start of each shape. When Draw mode is turned on (code 1), the vectors cause lines to be drawn. When Draw mode is turned off (code 2), the vectors move to a new location without drawing.

Codes 3 and 4: Size Control

Codes 3 and 4 control the relative size of each vector. The height specified with the SHAPE command is initially considered the length of a single orthogonal vector (directions 0, 4, 8, or C). Code 3 divides vector lengths by the next byte. Code 4 multiplies vector lengths by the next byte. Codes 3 and 4 are followed by a specification byte containing an integer scale factor (1 through 255). If you want the shape height to specify the size of the entire shape, and you use 10 vector lengths to draw it, you can use 3,10 to scale the height specification. The scale factor is cumulative within a shape; that is, multiplying by 2 and again by 6 results in a scale factor of 12. Usually, you should reverse the effect of your scale factors at the end of the shape, especially for subshapes and text font shapes. AutoCAD does not reset the scale factor for you.

Codes 5 and 6: Location Save/Restore

Code 5 pushes (saves) and code 6 pops (restores) the current coordinate position while drawing a shape so that you can return to it from a later point in the shape. You must pop everything you push. The position stack is only four locations deep. If the stack overflows because of too many pushes or too many missing pops, the following message is displayed when the shape is drawn.

Position stack overflow in shape *nnn*

Similarly, if you try to pop more locations than have been pushed onto the stack, the following message is displayed when the shape is drawn.

Position stack underflow in shape *nnn*

Code 7: Subshape

Code 7 draws the subshape number given by the next byte. For a non-Unicode font the specification byte following code 7 is a shape number from 1 to 255. For a Unicode font, code 7 is followed by a Unicode shape number from 1 to 65535. Unicode shape numbers should be counted as two bytes (for specific information about the differences between Unicode and non-Unicode fonts, see “Unicode Font Descriptions” on page 175). The shape with that number (in the same shape file) is drawn at this time. Draw mode is not reset for the new shape. When the subshape is complete, drawing the current shape resumes.

Codes 8 and 9: X-Y Displacements

Normal vector specification bytes draw only in the 16 predefined directions, and the longest length is 15. These restrictions help make shape definitions efficient but are sometimes limiting. With codes 8 and 9 you can draw nonstandard vectors using X-Y displacements. Code 8 specifies the X-Y displacement given by the next two bytes. Code 8 must be followed by two specification bytes in the format:

8, X-displacement, Y-displacement

The X-Y displacements can range from -128 to +127. A leading + is optional, and you can use parentheses to improve readability. The following example results in a vector that draws (or moves) 10 units to the left and three units up.

8, (-10, 3)

Following the two displacement specification bytes, the shape returns to Normal Vector mode.

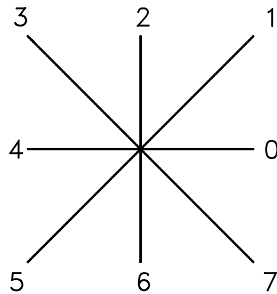
You can use code 9 to draw a sequence of nonstandard vectors. Code 9 specifies any number of *X-Y* displacement pairs. The code sequence is terminated by a (0,0) pair. The following example draws three nonstandard vectors and returns to Normal Vector mode.

```
9,(3,1),(3,2),(2,-3),(0,0)
```

You must terminate the sequence of *X-Y* displacement pairs with a (0,0) pair in order for AutoCAD to recognize any Normal Vectors or special codes that follow.

Code 00A: Octant Arc

Special code 00A (or 10) uses the next two specification bytes to define an arc. This is called an *octant arc* because it spans one or more 45-degree *octants*, starting and ending on an octant boundary. Octants are numbered counterclockwise from the 3 o'clock position, as shown in the following illustration.



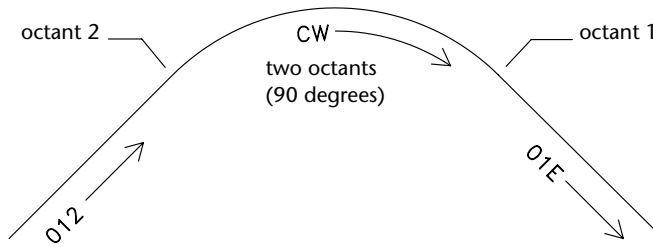
The arc specification is

```
10,radius,(-)OSC
```

The radius can be any value from 1 through 255. The second specification byte indicates the direction of the arc (counterclockwise if positive, and clockwise if negative), its starting octant (*s*, a value from 0 through 7), and the number of octants it spans (*c*, a value from 0 through 7, in which 0 equals eight octants, or a full circle). You can use parentheses to improve readability. For example, consider the following fragment of a shape definition:

```
...012,10,(1,-032),01E,...
```

This code draws a one-unit vector up and to the right, a clockwise arc from octant 3 (with a radius of one unit for two octants), and then a one-unit vector down and to the right, as shown in the following illustration.



Code 00B: Fractional Arc

Special code 00B (11) draws an arc that doesn't necessarily start and end on an octant boundary. The definition uses five specification bytes.

```
11,start_offset,end_offset,high_radius,radius,(-)0SC
```

The *start_offset* and *end_offset* represent how far from an octant boundary the arc begins or ends. The *high_radius* represents the most significant eight bits of the radius; the high radius will be 0 unless the *radius* is greater than 255 units. Multiply the *high_radius* value by 256 and add that value to the *radius* value to generate an arc radius greater than 255. The *radius* and ending specification byte are the same as for the octant arc specification (code 00A, described previously).

You determine the *start_offset* by calculating the difference in degrees between the starting octant's boundary (a multiple of 45 degrees) and the start of the arc. Then, you multiply this difference by 256 and divide by 45. If the arc starts on an octant boundary, its *start_offset* is 0.

The *end_offset* is calculated in a similar fashion, but you use the number of degrees from the last octant boundary crossed to the end of the arc. If the arc ends on an octant boundary, its *end_offset* is 0.

For example, a fractional arc from 55 degrees to 95 degrees with a 3 unit radius would be coded as follows:

```
11,(56,28,0,3,012)
```

Here is the explanation:

<i>start_offset</i>	= 56	<i>because</i> $((55 - 45) * 256 / 45) = 56$
<i>end_offset</i>	= 28	<i>because</i> $((95 - 90) * 256 / 45) = 28$
<i>high_radius</i>	= 0	<i>because</i> $(radius < 255)$
<i>radius</i>	= 3	
<i>starting octant</i>	= 1	<i>because arc starts in the 45 degree octant</i>
<i>ending octant</i>	= 2	<i>because arc ends in the 90 degree octant</i>

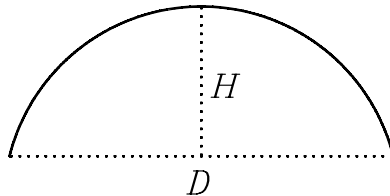
Codes 00C and 00D: Bulge-Specified Arcs

Special codes 00C and 00D (12 and 13) provide another mechanism for including arc segments in shape descriptions. They are similar to codes 8 and 9 in that you can use them to specify X - Y displacements. However, codes 00C and 00D draw arcs by applying a *bulge factor* to the displacement vector. Code 00C draws one arc segment, while code 00D draws multiple arc segments (*polyarcs*) until it is terminated by a (0,0) displacement.

Code 00C must be followed by three bytes describing the arc:

$00C, X\text{-displacement}, Y\text{-displacement}, \text{Bulge}$

Both the X and Y displacement and the bulge, which specifies the curvature of the arc, can range from -127 to $+127$. If the line segment specified by the displacement has length D , and the perpendicular distance from the midpoint of that segment has height H , the magnitude of the bulge is $((2 * H / D) * 127)$. The sign is negative if the arc from the current location to the new location is clockwise.



A semicircle has bulge 127 (or -127) and is the greatest arc that can be represented as a single-arc segment using these codes (use two consecutive arc segments for larger arcs). A bulge specification of 0 is valid and represents a straight-line segment. Note, however, that using code 8 for a straight-line segment saves a byte in the shape description.

The polyarc code (00D, or 13) is followed by 0 or by more arc segment triples, and is terminated by a (0,0) displacement. Note that no bulge is specified after the final displacement. For example, the letter *S* might be defined by the following sequence:

```
13, (0,5,127), (0,5,-127), (0,0)
```

Zero bulge segments are useful within polyarcs to represent straight segments; they are more efficient than terminating the polyarc, inserting one straight segment, and then starting another polyarc.

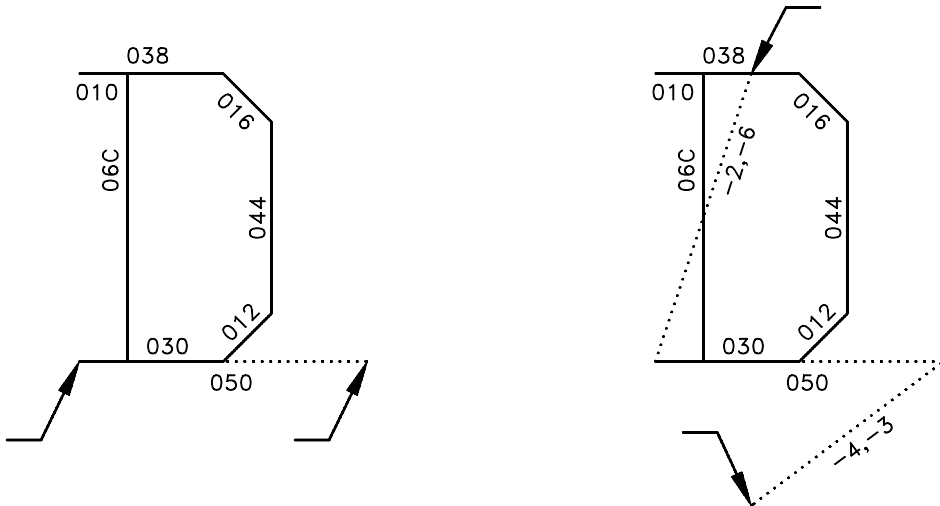
The number -128 cannot be used in arc segment and polyarc definitions.

Code 00E: Flag Vertical Text Command

Special code 00E (14) is used only in dual-orientation text font descriptions, where the font is used in both horizontal and vertical orientations. When this special code is encountered in a character definition, the next code is either processed or skipped, depending on orientation. If the orientation is vertical, the next code is processed; if it is horizontal, the next code is skipped.

In horizontal text, the start point for each character is the left end of the baseline. In vertical text, the start point is assumed to be the top center of the character. At the end of each character, a pen-up segment is normally drawn to position to the next character's start point. For horizontal text, it is to the right; for vertical text, it is downward. The special 00E (14) code is used primarily to adjust for differences in start points and endpoints, permitting the same character shape definition to be used both horizontally and vertically. For instance, the following definition of an uppercase D could be used in either horizontal or vertical text.

```
*68,22,ucd  
2,14,8,(-2, 6),1,030,012,044,016,038,2,010,1,06C,2,050,  
14,8,(-4,-3),0
```



Text Font Descriptions

AutoCAD is packaged with numerous text fonts. You can use the `STYLE` command to apply expansion, compression, or obliquing to any of these fonts, thereby tailoring the characters to your needs. You can draw text of any height, at any baseline angle, and with either horizontal or vertical orientation using these fonts.

If you intend to create your own text fonts, study the *txt.shp* file and the other fonts that are packaged with AutoCAD. They provide working examples of the topics discussed here.

A display of the contents of the *txt.shp* file is available in “Sample Files” in the *Customization Guide* in the Help system.

AutoCAD text fonts are files of shape definitions with shape numbers corresponding to the ASCII code for each character. For a list of the ASCII codes, see “ASCII Codes” on page 149.

Codes 1 through 31 are for control characters, only one of which is used in AutoCAD text fonts:

10 (LF)

The line feed (LF) must drop down one line without drawing. This is used for repeated `TEXT` commands, to place succeeding lines below the first one.

```
*10,5,1f
2,8,(0,-10),0
```

You can modify the spacing of lines by adjusting the downward movement specified by the LF shape definition.

Text fonts must include a special shape number 0 that conveys information about the font itself. The format has the following syntax:

```
*0,4,font-name
above,below,modes,0
```

The *above* value specifies the number of vector lengths above the baseline that the uppercase letters extend, and *below* indicates how far the lowercase letters descend below the baseline. The baseline is similar in concept to the lines on writing paper. These values define the basic character size and are used as scale factors for the height specified in the `TEXT` command.

The *modes* byte should be 0 for a horizontally oriented font and 2 for a dual-orientation (horizontal or vertical) font. The special 00E (14) command code is honored only when *modes* is set to 2.

The standard fonts supplied with AutoCAD include a few additional characters required for the AutoCAD dimensioning feature.

%%d Degree symbol (°)
%%p Plus/minus tolerance symbol (±)
%%c Circle diameter dimensioning symbol

You can use these and the %%*nnn* control sequences, as described under TEXT in the *Command Reference*.

Note AutoCAD draws text characters by their ASCII codes (shape numbers) and not by name. To save memory, specify the shape name portion of each text shape definition in lowercase as shown in the following example. (Lowercase names are not saved in memory.)

```
*65,11,uca  
024,043,04d,02c,2,047,1,040,2,02e,0
```

Because the shape name *uca* contains lowercase letters, AutoCAD doesn't save the name in memory. However, you can use the name for reference when editing the font definition file. In this example, *uca* stands for uppercase A.

Big Font Descriptions

Some languages, such as Japanese, use text fonts with thousands of non-ASCII characters. In order for drawings to contain such text, AutoCAD supports a special form of shape definition file called a *Big Font* file.

Define a Big Font

Special codes in the first line of a Big Font file specify how to read two-byte hexadecimal codes.

A font with hundreds or thousands of characters must be handled differently from a font containing the ASCII set of up to 256 characters. In addition to using more complicated techniques for searching the file, AutoCAD needs a way to represent characters with two-byte codes as well as one-byte codes. Both situations are addressed by the use of special codes at the beginning of a Big Font file.

The first line of a Big Font shape definition file must be as follows:

```
*BIGFONT nchars, nranges, b1, e1, b2, e2, . . .
```

where *nchars* is the approximate number of character definitions in this set; if it is off by more than about 10 percent, either speed or file size suffers. You can use the rest of the line to name special character codes (escape codes) that signify the start of a two-byte code. For example, on Japanese computers, Kanji characters start with hexadecimal codes in the range 90-AF or E0-FF. When the operating system sees one of these codes, it reads the next byte and combines the two bytes into a code for one Kanji character. In the *BIGFONT line, *nranges* tells how many contiguous ranges of numbers are used as escape codes; *b1, e1, b2, e2*, and so on, define the beginning and ending codes in each range. Therefore, the header for a Japanese Big Font file might look like this:

```
*BIGFONT 4000,2,090,0AF,0E0,0FF
```

After the *BIGFONT line, the font definition is just like a regular AutoCAD text font, except that character codes (shape numbers) can have values up to 65535.

Define an Extended Big Font File

To reduce the size of composite Kanji characters, you can define an extended Big Font file. Extended big fonts use the subshape code, followed immediately by a 0.

The first line of an extended Big Font file is the same as the regular Big Font file. This is the format for the remaining lines of the file:

```
*0,5,font-name
character-height, 0, modes, character-width,0
.
.
.
*shape-number,defbytes,shape-name
.
code,0,primitive#,basepoint-x,basepoint-y,width,height,
.
.
code,0,primitive#,basepoint-x,basepoint-y,width,height,
.
terminator
```

The following list describes the fields of a Big Font definition file:

character height	Used along with character width to indicate the number of units that define the font characters.
------------------	--

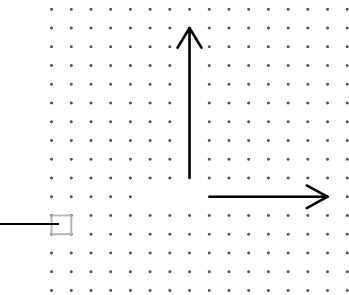
<code>character width</code>	Used along with character height to indicate the number of units that define the font characters. The <code>character-height</code> and <code>character-width</code> values are used to scale the primitives of the font. In this context, primitives are the points, lines, polygons, or character strings of the font geometrically oriented in two-dimensional space. A Kanji character consists of several primitives used repeatedly in different scales and combinations.
<code>modes</code>	The <i>modes</i> byte should be 0 for a horizontally oriented font and 2 for a dual-orientation (horizontal or vertical) font. The special 00E (14) command code is honored only when <i>modes</i> is set to 2.
<code>shape-number</code>	Character code.
<code>defbytes</code>	Byte size. It is always 2 bytes, consisting of a hexadecimal or a combination of decimal and hexadecimal codes.
<code>shape-name</code>	Character name.
<code>code</code>	Shape description special code. It is always 7 so that it can use the subshape feature.
<code>primitive#</code>	Reference to the subshape number. It is always 2.
<code>basepoint-x</code>	<i>X</i> origin of the primitive.
<code>basepoint-y</code>	<i>Y</i> origin of the primitive.
<code>width</code>	Scale of the width of the primitive.
<code>height</code>	Scale of the height of the primitive.
<code>terminator</code>	End-of-file indicator for the shape definition. It is always 0.

To arrive at the scale factor, AutoCAD scales down the primitive to a square unit and then multiplies it by the height and width to get the shape of the character. Character codes (shape numbers) in the Big Font shape definition

file can have values up to 65535. The following table describes the fields of the extended Big Font file.

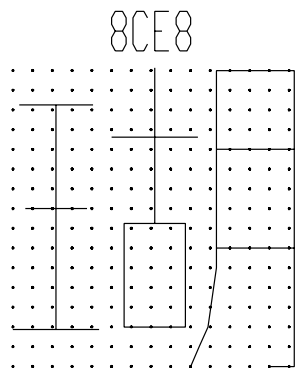
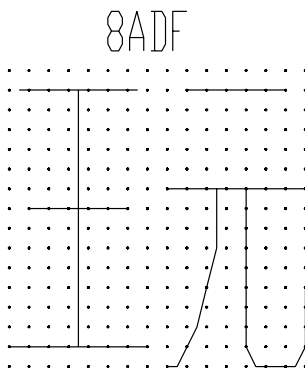
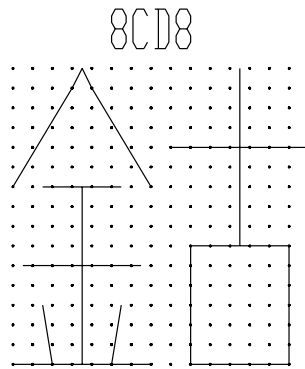
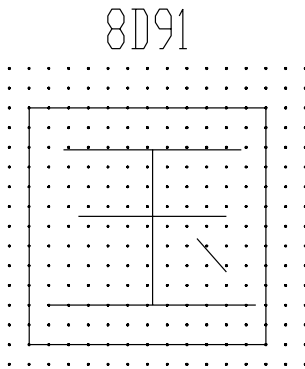
Fields of the extended Big Font file			
Variable	Value	Byte size	Description
<i>shape-number</i>	xxxx	2 bytes	Character code
<i>code</i>	7,0	2 bytes	Extended font definition
<i>primitive#</i>	xxxx	2 bytes	Refer to subshape number
<i>basepoint-x</i>		1 byte	Primitive X origin
<i>basepoint-y</i>		1 byte	Primitive Y origin
<i>width</i>		1 byte	Scale of primitive width
<i>height</i>		1 byte	Scale of primitive height
<i>terminator</i>	0	1 byte	End of shape definition

The following figure is an example of a 16×16 dot matrix that you could use to design an extended Big Font, such as a Kanji character. In the example, the distance between each dot is one unit. The callout points to a square unit.



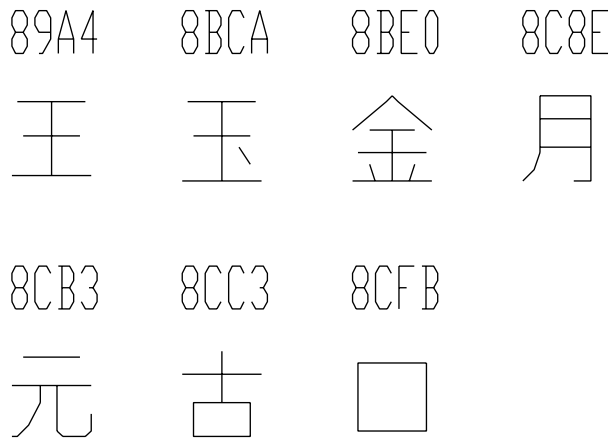
A square matrix for a Kanji character

The following figure shows examples of Kanji characters. Each character occupies an $M \times N$ matrix, (matrices don't have to be square) similar to the one shown in the previous figure. The numbers above each figure are the associated shape numbers.



Examples of Kanji characters

The following figure shows Kanji primitives.



Examples of Kanji primitives

Note Not all fonts are defined in a square matrix; some are defined in rectangular matrices.

Example: Shape Definition File for an Extended Big Font

```
*BIGFONT 50,1,080,09e
*0,5,Extended Font
15,0,2,15,0
*08D91,31,unspecified
2,0e,8,-7,-15,
7,0,08cfb,0,0,16,16,7,0,08bca,2,3,12,9,
2,8,18,0,2,0e,8,-11,-3,0
*08CD8,31,unspecified
2,0e,8,-7,-15,
7,0,08be0,0,0,8,16,7,0,08cc3,8,0,8,16,
2,8,18,0,2,0e,8,-11,-3,0
*08ADF,31,unspecified
2,0e,8,-7,-15,
7,0,089a4,0,0,8,16,7,0,08cb3,8,0,8,16,
2,8,18,0,2,0e,8,-11,-3,0
*08CE8,39,unspecified
2,0e,8,-7,-15,
7,0,089a4,0,1,5,14,7,0,08cc3,5,2,5,14,7,0,08c8e,9,0,7,
16,2,8,18,0,2,0e,8,-11,-3,0
*089A4,39,primitive
2,0e,8,-7,-15,2,8,1,14,1,0c0,
2,8,-11,-6,1,0a0,2,8,-12,-7,1,
0e0,2,8,-7,13,1,0dc,2,8,11,-1,
2,0e,8,-11,-3,0
*08BCA,41,primitive
```

```

2,0e,8,-7,-15,2,8,1,14,1,0c0,
2,8,-11,-6,1,0a0,2,8,-12,-8,1,
0e0,2,0e5,1,0ec,2,063,1,8,
2,-3,2,06f,2,0e,8,-11,-3,0
*08BE0,81,primitive
2,0e,8,-7,-15,2,8,3,9,1,080,
2,8,-10,-4,1,0c0,2,8,-13,-5,1,
0e0,2,8,-7,9,1,09c,2,8,-1,14,
1,8,-6,-5,2,8,8,5,1,8,6,-5,
2,8,-11,-6,1,8,1,-3,2,8,7,3,
1,8,-1,-3,2,8,-3,15,1,01a,2,
012,1,01e,2,8,10,-14,2,0e,8,
-11,-3,0
*08C8E,44,primitive
2,0e,8,-7,-15,2,8,3,15,1,090,0fc,038,
2,8,-6,11,1,090,2,8,-9,-5,1,
090,2,096,1,0ac,8,-1,-3,01a,01a,2,8,
18,0,2,0e,8,-11,-3,0
*08CB3,61,primitive
2,0e,8,-7,-15,2,042,1,02b,02a,018,2,
0d0,1,012,034,2,069,1,01e,040,2,8,
-8,6,1,02b,2,8,4,5,1,08c,2,8,
-3,8,1,03c,2,8,-5,3,1,0e0,2,8,
-12,5,1,0a0,2,8,6,-14,2,0e,8,
-11,-3,0

```

```
*08CC3,34,primitive
2,0e,8,-7,-15,2,0c1,1,06c,0a8,064,0a0,2,8,
-5,9,1,09c,2,8,-7,5,1,0e0,2,8,
4,-11,2,0e,8,-11,-3,0
*08CFB,22,primitive
2,0e,8,-7,-15,2,0d2,1,0cc,0c8,0c4,0c0,2,8,
5,-13,2,0e,8,-11,-3,0
```

Use Big Font Text in a Drawing

To use a Big Font for drawing text, you must set up a text style by using the STYLE command and then specify the name of the Big Font file. The same text style can use a normal ASCII font as well; enter only the two file names, separated by a comma. The following example uses the command line version of the STYLE command. To enable Big Fonts from the Text Style dialog box, choose the Use Big Font option.

```
Command: -style
Enter name of text style or [?] <current>: style_name
Specify full font name or font file name (TTF or SHX): txt,greek
```

AutoCAD assumes that the first name is the normal font and that the second is the big font.

If you enter only one name, AutoCAD assumes it is the normal font and removes any associated Big Font.

By using leading or trailing commas when specifying the font file names, you can change one font without affecting the other, as shown in the following table.

Input for changing fonts	
Input	Result
normal, big	Both normal and Big Font specified
normal,	Normal font only (Big Font unchanged)
,big	Big Font only (normal font unchanged)
normal	Normal font only (if necessary, Big Font removed)
ENTER (null response)	No change

When you use the STYLE command to list styles or to revise an existing style, AutoCAD displays the normal font file, a comma, and the Big Font file. If the style has only a Big Font file, it is displayed with a leading comma: ,greek.

For each character in a text string, AutoCAD searches the Big Font file first. If the character is not found there, the normal font file is searched.

Use a Big Font to Extend a Font

In some drafting disciplines, many special symbols can appear in text strings. The AutoCAD standard text fonts can be extended to include special symbols. However, extending standard text fonts has several limitations:

- The number of shapes is 255 per font file.
- Standard character set uses almost half the available shape numbers. Only codes 1 through 9, 11 through 31, and 130 through 255 are available.
- Multiple text fonts require duplication of the symbol definitions in each font.
- Special symbols require that you enter `%%nnn`, where *nnn* is the symbol's shape number.

The Big Font mechanism avoids these problems. You can select one or more seldom-used characters, such as the tilde (~) or the vertical bar (|), as escape codes, and use the next character to select the appropriate special symbol. For instance, you can use the following Big Font file to draw Greek letters by entering a vertical bar (|, ASCII code 124) followed by the equivalent Roman letter. Because the first byte of each character is 124, the character codes are biased by 124×256 , or 31744.

```
*BIGFONT 60,1,124,124
*0,4,Greek
above, below, modes, 0
*31809,n,uca
. . . uppercase Alpha definition, invoked by "|A"
*31810,n,ucb
. . . uppercase Beta definition, invoked by "|B"
*31841,n,lca
. . . lowercase Alpha definition, invoked by "|a"
*31842,n,lcb
. . . lowercase Beta definition, invoked by "|b"
*31868,n,vbar
. . . vertical bar definition, invoked by "||"
. . .
```

Unicode Font Descriptions

The standard AutoCAD fonts correspond to the character mapping used by the host operating system. This is because characters are stored directly in the database in the format in which they are obtained from the keyboard. The same character codes are used to generate fonts. This becomes a problem when using accented (8-bit) characters for which many character encoding standards exist.

Due to character mapping limitations, AutoCAD provides a set of Unicode fonts for the various code pages that AutoCAD uses. These fonts, while essentially the same, have some characters located in different places, depending on the code page they are defined for. If the font encoding used does not match that of the text in the drawing, the wrong characters may be drawn.

With Unicode fonts, text strings are converted to Unicode *before* being drawn, so it is no longer necessary to provide additional fonts for other languages or platforms. A single Unicode font, due to its large character set, is capable of supporting all languages and platforms. This feature is transparent to the user because the drawings are, if needed (due to differing code pages), converted to the AutoCAD system code page at load time. The drawings are always saved in the AutoCAD system code page.

Note Unicode does not provide adequate support for all Asian languages, so Big Fonts are still used by some or all Asian versions.

Unicode shape definition files are virtually identical in format and syntax to regular AutoCAD shape definition files. The main difference is in the syntax of the font header as shown in the following code:

```
*UNIFONT,6,font-name  
above,below,modes,encoding,type,0
```

The *font-name*, *above*, *below*, and *modes* parameters are the same as in regular fonts. The remaining two parameters are defined as follows:

encoding Font encoding. Uses one of the following integer values.

- 0** Unicode
- 1** Packed multibyte 1
- 2** Shape file

type	Font embedding information. Specifies whether the font is licensed. Licensed fonts must not be modified or exchanged. Bitcoded values can be added.
0	Font can be embedded
1	Font cannot be embedded
2	Embedding is read-only

Another important, and sometimes confusing, difference is the handling of the code 7 subshape reference. If a shape description includes a code 7 subshape reference, the data following the code 7 is interpreted as a two-byte value. This affects the total number of data bytes (*defbytes*) in the shape description header. For example, the following shape description is found in the *romans.shp* file:

```
*00080,4,keuroRef
7,020AC,0
```

The second field in the header represents the total number of bytes in the shape description. If you are not used to working with Unicode font descriptions, you may be inclined to use three bytes rather than four, but this would cause an error during the compiling of the SHP file. This is true even if the shape number you are referencing is not in the two-byte range (below 255); the compiler always uses two bytes for this value, so you must account for that in the header.

The only other difference between Unifont shape definitions and regular shape definitions is the shape numbers. The Unifont shape definitions that AutoCAD provides use hexadecimal shape numbers as opposed to decimal values. Although hexadecimal numbers are not required their use makes it easier to cross-reference the shape numbers with the \u+ control character values.

Superscripts and Subscripts in SHX Files

The AutoCAD SHX fonts have limited superscript and subscript capabilities. However, it is relatively easy to modify shape definition files to improve superscript and subscript capability.

Creating superscripts and subscripts requires two steps. First, the “imaginary pen” that is creating the text, vector by vector, on your screen needs to be shifted up or down. Then, the font “scale” needs to be reduced. In addition, the reverse process has to take place to return to the normal font. The font needs to recognize four new keys: two for superscripts and two for subscripts. To avoid altering the existing font definitions, you can access these with the numeric keypad on your keyboard.

To add superscript and subscript definitions to a font

This example procedure is based on the AutoCAD Romans font file, although a similar method applies to any AutoCAD font. This procedure adds four new shape definitions to a font: `super_on`, `super_off`, `sub_on`, and `sub_off`, which control the position and size of the characters that follow. For simplicity, this example replaces the left- and right-bracket characters ([and]) and the left and right curly brace characters ({ and }) with the new characters. You may choose to replace other characters or use a shape number in the extended range (ASCII codes 128 through 256). If you use an extended shape number, you need to use the `%%nnn` method (where *nnn* is the ASCII value of the character) for placing the new characters. (See “ASCII Codes” on page 149.)

- 1 Edit the SHP (in this case, *romans.shp*) file with an ASCII text editor. You can create a new file called *romanss.shp* rather than modify the original file. The *romans.shp* file is available on the AutoCAD CD.

The *romans.shp* file is also available at “Sample Files” in the *Customization Guide* in the Help system.

- 2 Search for the shape definitions of the characters you are replacing. To comment out those definitions so the new definitions can take their place, insert a semicolon in front of each line of the shape definition. The shape definition may continue for a number of lines.

The left- and right-bracket characters have ASCII values of 91 and 93 (05B and 05D hex values, if the font is Unicode). The left and right curly brace characters have ASCII values of 123 and 125 (07B and 07D hex).

- 3 Add the first and second values on the second line of the definition, and divide the total by 2 as shown in the following example:

```
*UNIFONT,6,Extended Simplex Roman for UNICODE
21,7,2,0 21 + 7 = 28, then 28 / 2 = 14. This number is used
later.
```

- 4 Add the following lines to the end of the SHP file:

```
*91,8,super_on
2,8,(0,14),003,2,1,0
*93,8,super_off
2,004,2,8,(0,-14),1,0
*123,8,sub_on
2,8,(0,-14),003,2,1,0
*125,8,sub_off
2,004,2,8,(0,14),1,0
```

Notice the 14 and -14 values in the preceding lines. They are *Y* axis offsets for the imaginary pen. The value 14 is half the maximum height of a character in this font, which is the correct approximation for superscripts and subscripts. This value needs to be calculated for each font file, but you can modify it any way you want.

5 Save the file.

6 Use the `COMPILE` command to compile the SHP file.

Once the shape is compiled and an appropriate style is defined, you can access the new pen-up and pen-down commands by entering the `[`, `]`, `{`, and `}` characters. The `[` character initiates superscript and the `]` character returns from superscript to normal. The `{` character initiates subscript and the `}` character returns from subscript to normal.

Index

Symbols

-) (close parenthesis), in menu item labels, 76, 77
- _ (underscore character)
 - in commands, 12, 14
 - in hatch pattern definitions, 38
 - in menu item name tags, 48
 - in menu macros, 54, 58
 - in screen menus, 92
- > and <-, in menu item labels, 71
- \$ commands
 - in DIESEL expressions, 106
 - in hatch pattern definitions, 38
 - in menu macros, 80
- /b switch, running scripts and, 125
- ^c^c, in menu macros, 54, 61
- 'GRAPHSCR command, using in command scripts, 124
- \$I= command, loading image tile menus and, 89
- !., in menu item labels, 76, 77
- \$M= command, in menu macros, 63
- ^P, in menu macros, 56
- \$Pn=* command, for menu display, 67, 80, 81
- SHADE command, 91
- /t switch, running scripts and, 126
- 'TEXTSCR command, in command scripts, 124
- TOOLBAR command, 81, 84
- /v switch, running scripts and, 125
- ^z, in menu macros, 63

A

- abbreviations for commands. *See* aliases
- absolute referencing of menus, 78
- ACAD menu group, 8
- acad.bat* file, 8

- acad.lin* (linetype library) file, 28
- acad.lsp* file, 139, 140
- acad.mnl* (menu LISP) file, 52
- acad.mnu* (template menu) file, 46, 50, 52
- acad.pat* (hatch pattern library) file, 39, 41, 43
- acad.pgp* (program parameters) file, 21, 25, 131
- acad.rx* file, 146
- acad2000.lsp* (reserved AutoLISP file required by AutoCAD) file, 141
- acad2000doc.lsp* (reserved AutoLISP file required by AutoCAD) file, 142
- acad2004.cfg* file, 6
- acadapps* folder, 5
- acaddoc.lsp* file, 139, 141
- acadiso.lin* (metric linetype definition) file, 28
- acadiso.pat* (metric hatch pattern definition) file, 39, 41, 43
- ACADLSPASDOC system variable, 139, 140
- accelerator keys. *See* shortcut keys
- Accelerators menu file section
 - about, 48, 49
 - examples, 98, 101, 132
- ActiveX Automation
 - about, 130
 - controllers, 130, 133
 - defining commands to start applications, 131
 - objects, 131
 - starting applications from a menu, 131
 - Visual Basic for Applications and, 130, 133
- addition (+) DIESEL function, 111
- aliases
 - in Aux and Buttons menus, 68
 - for commands, 2, 25
 - in Pop menus, 73
 - for toolbar flyouts, 85

- ALT key combinations, 74
- and DIESEL function, 112
- angtos DIESEL function, 112
- apostrophe ('), transparent command character, 12, 14
- applications
 - AutoLISP, 137, 138
 - ObjectARX, 145
 - starting
 - from a command, 131
 - from a menu, 131
 - VBA, 134
- APPLOAD command, 135, 138, 139, 146
- arcs
 - bulge-specified, 163
 - fractional, 161
 - octant, 161
- Array of Thumbnails template (Publish to Web wizard), 19, 20
- Array Plus Summary template (Publish to Web wizard), 19, 20
- arrow keys, using with shortcut key modifiers, 100
- ARX command, 145
- arxload AutoLISP function, 145, 147
- arxunload AutoLISP function, 146
- ASCII codes
 - table of, 149
 - in text font shape numbers, 165, 166
- A-type alignment of linetypes, 29, 30
- autoarxload AutoLISP function, 140, 147
- AutoCAD
 - classic shortcut key assignments, 16, 133
- AutoCAD Runtime Extension. *See* ObjectARX
- AutoCAD VBA. *See* Visual Basic for Applications
- AutoLISP (LSP) applications
 - about, 137
 - acad.lsp* file, 139, 140
 - acaddoc.lsp* file, 139, 141
 - DIESEL returned values in, 108
 - errors when loading, 143
 - loading, 137, 138
 - loading/running automatically, 139
 - MNL file, 139, 142
 - specifying a directory path for, 139, 146
 - storage of, 138
- AutoLISP code
 - about, 136
 - for customizing the status line, 106
 - for defining OTYPE, 75
 - examples, 64, 106
 - in menu files, 52
 - in menu macros, 64, 108
 - reading of, by AutoCAD, 137
 - setting MODEMACRO values with, 106
 - for toolbar display, 84

- AutoLISP functions
 - arxload, 145, 147
 - arxunload, 146
 - autoarxload, 140, 147
 - autoload, 139
 - command, 137, 139, 143
 - defun, 143, 144
 - defun-q, 144
 - load, 138, 139, 141, 143
 - menucmd, 69, 76, 77, 79, 80, 89
 - princ, 142, 143
 - S::STARTUP, 107
 - startapp, 131, 132
 - strcat, 106
- AutoLISP routines. *See* AutoLISP (LSP)
 - applications
- autoload AutoLISP function, 139
- Automation. *See* ActiveX Automation
- Aux menu file sections
 - aliases in, 68
 - vs. Buttons menu sections, 65
 - coordinate entry in, 69
 - examples, 66, 67, 68, 69
 - labels in, 47, 67, 68, 69
 - swapping contents of, 67, 69

B

- backslash (\)
 - in AutoLISP, 139, 146
 - in menu macros, 54, 55, 56, 57, 59
- base menu
 - Aux and Buttons menu sections and, 65
 - loading, 50, 53
 - relative referencing and, 77
- Big Font file
 - byte size, 168
 - character height/width, 167
 - defining, 166
 - drawing text with, 173
 - end-of-file indicator, 168
 - examples, 167
 - extended, 167
 - extending text fonts with, 174
 - fields described, 167
 - height/width of the primitive, 168
 - modes byte, 168
 - one-byte vs. two-byte codes in, 166
 - shape description special code, 168
 - shape names, 168
 - shape numbers, 168
 - specifying names of, 173
 - subshape number, 168
 - X and Y origins of the primitive, 168
- bitmaps (BMP images)
 - for toolbar button image resources, 84, 87
 - for toolbar flyout image resources, 85, 87

- bitmaps (BMP images) (*continued*)
 - user-defined, 87
- blocks, 154
- bulge factor, in arc specifications, 163
- bulge-specified arc codes in shape specification
 - bytes, 163
- button assignments, Aux and Buttons menu
 - sections and, 66
- Button Editor, 13
- button images, customizing, 12, 13
- button macros, 12, 14
- Button Properties tab (Customize dialog box), 14
- Buttons menu file sections
 - aliases in, 68
 - vs. Aux menus, 65
 - coordinate entry in, 69
 - labels in, 47, 67, 68, 69
 - swapping contents of, 67, 69

C

- cascading menus. *See* pull-down menus
- cascading menus. *See* shortcut menus
- character height/width in Big Font file, 167
- check marks, on menu item labels, 76
- Circle button, in Button Editor, 14
- CMCOMMAND menu alias, 75
- cmd (Windows system command), 23
- CMDEFAULT menu alias, 74
- CMEDIT menu alias, 74, 75
- color control element of toolbars, 86
- command aliases, 2, 25
 - in Aux and Buttons menus, 68
 - in Pop menus, 73
 - for toolbar flyouts, 85
- command AutoLISP function, 137, 139, 143
- command autoloader, 139
- command field, in the external commands
 - section, 22
- command line switches
 - running scripts and, 125
- Command mode shortcut menu, alias for, 75
- command name validity, search procedure for, 5
- command scripts. *See* scripts
- commands
 - associated with buttons, changing, 12
 - disabling, 3
 - redefining, 3
 - repeating
 - in menu macros, 61
 - to start applications, 131
- Commands tab (Customize dialog box), 13
- comment lines, in scripts, 123, 124
- COMPILE command, 154, 155, 158
- compiling shape or font files, 154, 155
- complex linetypes, including shapes in, 34

- component objects
 - model architecture, 130
- conditional expressions, in menu macros, 63
- configuration files
 - about, 6
 - multiple, 7
 - for toolbars, 8
- context menus, 70
 - See also* shortcut menus
- control elements of toolbars, 86
- coordinate entry
 - in button menus, 69
 - with shortcut keys, 101
- coordinate position in shape specification bytes, 160
- coordinates
 - of toolbars, 83
- custom linetypes. *See* linetypes
- custom menu
 - AutoLISP and, 64
 - using slides with, 120
- custom templates. *See* templates
- custom-defined commands, 24
 - See also* external commands
- CUSTOMIZE command, 2, 9, 82
- Customize dialog box, 8
 - toolbar button modifications, 10
 - Toolbars tab, 8

D

- dashes
 - in hatch pattern definitions, 39, 40
 - in linetype definitions, 28, 30, 31
- data bytes, required to describe shapes, 156
- date and time formats, DIESEL function for, 113
- DCL (dialog control language) files, 137
- Default mode shortcut menu, alias for, 74
- defbytes
 - in Big Font files, 168
 - in shape descriptions, 156
 - in Unicode font descriptions, 176
- defun AutoLISP function, 143, 144
- defun-q AutoLISP function, 144
- DEL key, using with shortcut key modifiers, 100
- DELAY command, 124, 127
- deleting
 - error messages, 10
 - toolbar buttons, 15
- DIESEL (Direct Interpretively Evaluated String Expression Language)
 - about, 103
- DIESEL expressions
 - for changing menu width, 110
 - for customizing the status line, 2, 105
 - error messages, 117
 - examples, 105, 108, 109

- DIESEL expressions (*continued*)
 - in menu item labels, 76
 - in menu macros, 63, 108
 - nesting, 105
 - in pull-down menu labels, 109, 110
 - using getvar in, 105
 - using quoted strings in, 106
 - using returned values in AutoLISP routines, 108
 - using the dollar sign (\$) in, 106
- DIESEL functions
 - addition (+), 111
 - and, 112
 - angtos, 112
 - division (/), 111
 - edtime, 113
 - eq, 114
 - equal to (=), 111
 - eval, 115
 - getenv, 115
 - getvar, 105, 115
 - greater than (>), 112
 - greater than or equal to (>=), 112
 - if, 115
 - index, 115, 116
 - less than (<), 111
 - less than or equal to (<=), 112
 - multiplication (*), 111
 - not equal to (!=), 112
 - nth, 116
 - or, 116
 - parameter limits, 111
 - rtos, 116
 - strlen, 116
 - substr, 116
 - subtraction (-), 111
 - upper, 117
 - xor, 117
- dimension style control element of toolbars, 86
- dimensioning
 - text font characters required for, 166
- directories
 - structure for program and support files
 - changing, 4
 - recommendations for, 5
- directory path
 - for AutoLISP files, 139, 146
 - for program and support files, 4
- disabling
 - commands, 3
 - menu item labels, 76, 77
- display characteristics, of toolbars, 83
- division (/) DIESEL function, 111
- dollar signs (\$)
 - in DIESEL expressions, 106
 - in hatch pattern definitions, 38
 - in menu macros, 80

- dollar signs (\$) (*continued*)
 - in screen menus, 92
- dots in linetype definitions, 28, 30, 31
- Draw mode codes in shape specification bytes, 159
- dual-orientation text font descriptions, 164, 165
- DXF names of objects, 75

E

- echoes of characters in command window,
 - suppressing, 56
- Edit mode shortcut menu, alias for, 74
- editing commands
 - slide files and, 121
- edtime DIESEL function, 113
- enabling
 - menu item labels, 76
- eq DIESEL function, 114
- equal to (=) DIESEL function, 111
- Erase button, in Button Editor, 14
- error messages
 - AutoLISP, 143
 - DIESEL, 117
- es.shx* file, 34
- ESC key
 - using with shortcut key modifiers, 100
- eval DIESEL function, 115
- examples
 - \$M= command, 63
 - absolute referencing of menus, 78
 - Accelerators menu file section, 98, 101
 - alias labels in button menus, 68
 - arc definitions, 161, 162, 163
 - arxload AutoLISP function, 146
 - AutoLISP
 - access to label status, 79
 - expressions in menu items, 108
 - routines, 109
 - autoload AutoLISP function, 139
 - Aux menu file section, 66
 - Big Font file header, 167
 - cascading submenu indicator characters, 71
 - commands that start applications, 131
 - coordinate entry
 - in button menus, 69
 - with shortcut keys, 101
 - custom-defined commands, 24
 - DIESEL expressions
 - in menu files, 109
 - in menu item labels, 76
 - in menu items, 108
 - in pull-down menu labels, 109, 110
 - to change menu width, 110
 - DIESEL language macro expressions, 105
 - dual-orientation text font descriptions, 164
 - edtime DIESEL function, 114

examples (*continued*)

- extended Big Fonts, 169, 172
 - extending text fonts using Big Fonts, 174
 - hatch pattern definitions, 38, 40, 42
 - Helpstrings menu file section, 97
 - image tile menus, 89
 - Kanji characters, 170
 - Kanji primitives, 171
 - linetype definitions, 28, 34
 - linetypes, text characters in, 31
 - load AutoLISP function, 141
 - loading VBA projects, 136
 - menu bar title, 73
 - menu files, 46
 - menu item labels
 - disabling, 78
 - marking, 78
 - menu item numbering, 79
 - menu items, 49
 - menu macros, 54
 - international language support in, 58
 - longer than one line, 56
 - pausing for user input, 56
 - repeating commands in, 61
 - special characters in, 59
 - terminating, 55, 63
 - using AutoLISP in, 64
 - menucmd load function, 77, 78, 80
 - MNL file for an AutoLISP menu, 142
 - MODEMACRO system variable, 104, 106
 - nonstandard vectors, 160
 - object-specific shortcut menu items, 75
 - pull-down menus, 72, 81, 132
 - relative referencing of menus, 77
 - S::STARTUP AutoLISP function, 144
 - screen menus, 91, 93, 94
 - scripts, 124, 125, 127
 - shape files, 157
 - Single Object Selection mode, 61
 - slide library, 122
 - status line customization, 104
 - STYLE command, 173
 - submenu referencing, 63
 - superscript/subscript definitions, 177
 - swapping menus, 80
 - text shape definitions, 166
 - toolbar definition, 83
 - Toolbars menu file section, 82
- executable field, in the external commands section, 22
- extended Big Font files
- defining, 167
 - end-of-file indicator, 169
 - examples, 169, 172
 - extended font definition, 169
 - fields described, 168

extended Big Font files (*continued*)

- height/width of primitives, 169
 - shape and subshape numbers, 169
 - X and Y origins of the primitive, 169
- external applications
- running from within AutoCAD, 2
- external commands
- about, 21
 - command field, 22
 - defining, 22, 23
 - executable field, 22
 - flags field, 22
 - prompt field, 23
 - return_code field, 23

F

- FAS files, 137
- file types
- for menu files, 51
- FILEDIA system variable
- viewing slides and, 122
- flag vertical text code in shape specification
- bytes, 164
- flags field, in the external commands section, 22
- floating toolbars, changing the shape of, 11
- flyouts (in toolbar buttons)
- alias option, 85
 - creating, 12, 13, 85
 - icon option, 85
 - image resource bitmap options, 85
 - menu item macro option, 85
 - naming, 85
- font files
- adding superscript/subscript definitions to, 176
 - compiling, 154, 155
 - dual-orientation font descriptions, codes for, 164
 - specifying names of, 173
 - See also* fonts
- fonts
- character requirements for dimensioning, 166
 - creating, 2, 165
 - extending to include special symbols, 174
 - shape numbers in, 165
 - Unicode fonts, 175
 - See also* font files
- forward slash (/)
- using in menu macros, 57
- fractional arc code in shape specification bytes, 162
- function keys (F1-F12), using with shortcut key modifiers, 99

G

- getenv DIESEL function, 115
- getvar DIESEL function, 105, 115
- global referencing. *See* relative referencing
- greater than (>) DIESEL function, 112
- greater than or equal to (>=) DIESEL function, 112
- grips
 - resizing, 65
- GRIPS menu alias, 74

H

- hatch patterns
 - about, 38
 - creating, 39, 41, 43
 - with dashes, 39, 40
 - descriptor line, 39, 41, 44
 - examples, 38, 40, 42
 - format for, 38
 - header line, 39, 41, 43
 - line families, 39
 - with multiple lines, 42
 - rejection of, 39
 - rules for, 38
- height/width of primitives, in Big Font files, 168
- Help
 - messages, on the status line, 97
- Helpstrings menu file section, 48, 49, 97, 132
- hierarchical menus. *See* pull-down menus; shortcut menus
- horizontal toolbars, 11
- Hot Grip shortcut menu, alias for, 74
- hyphen (-)
 - in hatch pattern definitions, 38
 - as prefix for command aliases, 26, 75
 - in screen menus, 92

I

- icons
 - for toolbar flyouts, 85
- ID strings
 - of toolbar button bitmaps, 84
 - of toolbar button flyouts, 85
- IDE. *See* integrated development environment
- if DIESEL function, 115
- Image menu file section, 48, 49, 87
- image tile menus, 121
 - aspect ratio for display of, 91
 - defining, 87
 - displaying labels in, 88
 - icon menu-labeling options for, 88
 - loading, 89
 - menu macros for, 89
 - preparing slides for, 90

- image tile menus (*continued*)
 - slide files and, 87, 120
- index DIESEL function, 115, 116
- INS key, using with shortcut key modifiers, 99
- inserting
 - pull-down menus, 80
- integrated development environment
 - VBA, 135
 - Visual LISP, 136
- international language support
 - Japanese/Kanji, 167, 168, 170
 - in menu macros, 58

K

- Kanji primitives, examples, 171
- Kanji text
 - examples, 170
 - handling, 167, 168
- keys (for shortcut key modifiers), 99

L

- labels. *See* menu item labels; section labels; labels (data labels)
- large-image resource bitmap
 - for toolbar buttons, 84
 - for toolbar flyouts, 85
- layer control element of toolbars, 86
- less than (<) DIESEL function, 111
- less than or equal to (<=) DIESEL function, 112
- libraries
 - See also* standard libraries
 - slide libraries, 88
- library search path, for program and support files, 4
- LIN (linetype library) files, 28
- Line button, in Button Editor, 14
- line feed shape definition, 165
- linetype control element of toolbars, 86
- linetypes
 - about, 28
 - alignment field (A), 29
 - complex, 34
 - creating, 30
 - dashes/dots in, 28, 30, 31
 - description field, 29
 - examples, 28, 34
 - format for, 28, 31, 34
 - including shapes in, 34
 - including text characters in, 31, 33
 - linetype name field, 29, 30
 - loading, 31
 - pattern descriptor fields, 30
 - simple, 28
 - transform field, 34
- lineweight control element of toolbars, 86

- LISPINIT system variable, 140
- List of Drawings template (Publish to Web wizard), 19, 20
- List Plus Summary template (Publish to Web wizard), 19, 20
- load AutoLISP function, 138, 139, 141, 143
- LOAD command, 154, 158
- Load option (ARX command), 145
- loading
 - AutoLISP applications, 138, 139
 - base menu, 53
 - image tile menus, 89
 - menu files, 47, 50
 - ObjectARX applications, 145, 146
 - partial menu files, 50, 53
 - VBA applications, 135
- location save/restore codes in shape specification
 - bytes, 160
- LSP (AutoLISP) files, 137, 138

M

- Macros dialog box, 135
- macros. *See also* button macros; menu macros; VBA macros
- MaxHatch system registry variable, 39
- MBUTTONPAN system variable, 66
- MDE. *See* Multiple Design Environment
- menu bar
 - changing, 50
 - constructing for pull-down menus, 73
 - defining the title of, 73
 - restoring, 51
- MENU command, 50, 53
- menu files
 - about, 46
 - for AutoLISP menus, 139
 - comment lines, 47
 - compiling, 51, 123
 - DIESEL expressions in, 108
 - editing, 2
 - examples, 46, 109
 - file types, 51
 - generating, 51, 52
 - Helpstrings section, 97
 - item definition lines, 47
 - item numbering and, 79
 - loading, 47, 50, 53
 - Menugroup section, 46, 47
 - naming, 46, 47
 - pick button assignments and, 67
 - section labels, 47, 49
 - structure of, 47
 - unloading, 50, 53
- menu groups
 - relative referencing and, 78
 - for toolbars, 8

- menu item labels
 - about, 48
 - AutoLISP access to, 79
 - in button menus, 67, 68, 69
 - for cascading submenus, 71
 - for creating separator lines on menus, 71
 - DIESEL expressions in, 76, 109
 - disabling, 76, 77
 - enabling, 76
 - examples, 109
 - graying out, 76
 - Helpstrings and, 98
 - marking, 76, 77
 - for screen menus, 91
 - for shortcut keys, 98
 - special characters allowed in, 72
 - for tablet menus, 96
 - tracking changes to, 76
 - usage of, 49
- menu items
 - accessing from the keyboard, 74
 - defining, 47
 - defining the action of. *See* menu macros
 - examples, 49, 108
 - format for, 48
 - name tags for, 48
 - numbering, 79
 - using AutoLISP for, 64, 108
 - using DIESEL for, 108
- menu LISP files. *See* MNL files
- menu macros
 - *^c^c in, 54, 61
 - about, 48, 49
 - AutoLISP expressions in, 52, 64, 108
 - backslash (\) character in, 54, 55, 56, 57, 59
 - brackets ([]) and, 59
 - canceling commands before executing, 54
 - conditional expressions in, 63
 - DIESEL expressions in, 108
 - examples, 54, 55, 56, 58, 61, 63, 80
 - forward slash (/) in, 57
 - for image tile menus, 89
 - international language support in, 58
 - longer than one line, 56
 - pausing for user input, 56, 99
 - plus sign (+) in, 55, 56
 - repeating commands in, 61
 - resuming after a pause, 57
 - semicolon (;) in, 55, 57, 59
 - separator line labels and, 71
 - for shortcut keys, 98
 - Single Object Selection Mode and, 61
 - spaces in, 55
 - special characters in, 58, 60
 - starting applications from a menu, 131
 - submenu referencing, 63

- menu macros (*continued*)
 - suppressing echoes and prompts in, 56
 - swapping menus and, 62, 80
 - for tablet menus, 96, 97
 - terminating, 55, 63
 - for toolbar buttons, 85
 - for toolbar flyouts, 85
 - underscore (_) character in, 54, 58
- menu resource files. *See* MNR files
- menu sections
 - labels and, 47, 49
 - valid names of, 62
- menu swapping
 - Aux and Buttons menus, 67, 69
 - screen submenus, 96
 - toolbars and, 82
 - using menu macros for, 62, 80
- menucmd AutoLISP function
 - accessing label status, 79
 - disabling menu item labels, 76
 - inserting/removing menus, 80
 - loading image tile menus, 89
 - marking menu item labels, 76
 - referencing menu items, 77, 78
 - swapping menus, 69, 80
- MENUCTL system variable, 96
- MENUECHO system variable, 56
- Menugroup section of menu files, 46, 47
- MENULOAD command
 - altering menus and, 51
 - inserting sections into the menu bar, 73, 133
 - partial menus and, 50, 53
 - small menu files and, 48
- menus
 - accessing from the keyboard, 74
 - changing, 50
 - custom menus, 51
 - inserting, 80
 - reinitializing, 51
 - removing, 51, 80, 81
 - restoring, 51
 - shortcut keys for items on, 18
 - starting applications from, 131
 - swapping. *See* menu swapping
 - See also* image tile menus; partial menu files; pull-down menus; shortcut menus
- MENUUNLOAD command
 - altering menus and, 51
 - partial menus and, 50, 53
 - small menu files and, 48
- MNC files, 51, 52
- MNL files, 52, 64, 139, 142
- MNR (menu resource) files, 51, 52
- MNS files, 51, 52
- MNT files, 51
- MNU (template menu) files, 51, 52
 - See also* menu files
- model space
 - slides created in, 120
- MODEMACRO system variable
 - about, 2, 104
 - examples, 104
 - setting values for, 104, 106
 - string length and character limitations, 104
 - using getvar, 105
- modes byte in Big Font files, 168
- modifiers for shortcut keys
 - concatenating, 99
 - virtual keys to use with, 99
- MSLIDE command, 91, 121
- MTEXT command, 154
- multiple configurations, 7
- Multiple Design Environment, 136
- multiple-line hatch patterns, 42
- multiplication (*) DIESEL function, 111

N

- namespaces, in Visual LISP, 136
- naming
 - toolbar buttons, 84
 - toolbar flyouts, 85
 - toolbars, 83
- New Toolbar dialog box, 9
- nonstandard vectors, 160
- not equal to (!=) DIESEL function, 112
- nth DIESEL function, 116
- numeric keypad, modifying for coordinate entry, 101
- numeric keys, using with shortcut key modifiers, 100

O

- object menus, 75
- object reactors, AutoLISP and, 136
- object snap menu, 70
- ObjectARX applications
 - loading, 145
 - loading automatically, 146
 - unloading, 145, 146
- ObjectARX environment, 145
- octant arc code in shape specification bytes, 161
- octant boundaries, 162
- offset
 - in arc specifications, 162
 - of shapes in linetypes, 36
 - of text characters in linetypes, 33
- OPTIONS command, 52
- or DIESEL function, 116
- orientation
 - text fonts, 164, 165

orientation (*continued*)
 toolbars, 83
OTYPE command, 75

P

PAN command, 91
paper space
 slides created in, 120
parameter limits DIESEL function, 111
partial menu files
 Aux and Buttons sections and, 65
 controlling toolbars across, 81, 84
 loading, 50, 53
 relative referencing and, 77
 removing, 51
 unloading, 50, 53
path name. *See* directory path
pauses within scripts, 56, 99
pc.shx file, 35
Pencil button, in Button Editor, 14
pen-down lengths (dashes) in linetype
 definitions, 28, 30
pen-up lengths (spaces) in linetype definitions,
 28, 30
PFB (printer font binary) files, 154, 155
pick button (mouse)
 assignments, menu files and, 67
PICKADD system variable
 command AutoLISP function and, 137
 menu items and, 58
 scripts and, 124
PICKAUTO system variable
 command AutoLISP function and, 137
 menu items and, 58
 scripts and, 124
plot style control element of toolbars, 86
plus sign (+)
 in menu macros, 55, 56
Pn= syntax
 inserting/removing pull-down menus and,
 81
pointing devices
 Aux and Buttons menu sections and, 65
 buttons
 default pick button on, 67
polyarcs, 163
Pop menu file sections
 about, 47, 49
 alias labels for, 73
 calling macros and, 64
 menu bar for, 73
 for pull-down menus, 70, 72
 referencing and, 78
 for shortcut menus, 70, 74
 special characters for labels in, 72
 swapping menus and, 63, 80, 82

PostScript fonts
 compiling, 154, 155
 copyright restrictions, 155
presets (block insertion), menu macros and, 64
princ AutoLISP function, 142, 143
program files
 changing the directory structure for, 4
 library search path for, 4
 organizing, 2, 4
programming interfaces
 ActiveX Automation, 130
 AutoCAD VBA, 133
 AutoLISP, 136
 ObjectARX, 145
 Visual LISP, 136
prompt field, in the external commands section,
 23
prompts
 suppressing, 56
PTWTemplates folder, 20
Publish to Web wizard
 customizing a template for, 2, 19
pull-down menus
 about, 70
 absolute referencing of, 77, 78
 cascading submenu indicators, 71
 creating, 72
 DIESEL expressions in, 109
 examples, 72, 132
 inserting, 80
 menu bar for, 73
 menu item limits, 70
 relative referencing of, 77, 80
 removing, 80, 81
 separator lines on, 71
 starting applications from, 131
 swapping, 80, 82
 width of, 110
PWT (Publish to Web template) files, 19

Q

quoted strings, in DIESEL expressions, 106

R

radius
 in arc specifications, 161, 162
Redraw (View menu), 121, 122
Redraw command (View menu), viewing slides,
 122
referencing
 of pull-down or shortcut menus, 77
 swapping menus and, 80
relative referencing
 of pull-down and shortcut menus, 77
 swapping menus and, 80

- removing
 - partial menus, 51
 - pull-down menus, 80, 81
 - shortcut keys, 18
- renaming
 - toolbars, 9
- repeating commands
 - in menu macros, 61
- repositioning toolbar buttons, 10
- restoring
 - deleted toolbar buttons, 15
 - screen submenus, 93
- RESUME command, 124
- return_code field, in the external commands section, 23
- right-click menus. *See* shortcut menus
- rotating
 - shapes in linetypes, 34, 35
 - text characters in linetypes, 33
- rows specification for toolbars, 83
- RSCRIPT command, 124
- rtos DIESEL function, 116

S

- S::STARTUP AutoLISP function
 - including in startup LISP files, 139, 143
 - overwriting, 144
 - setting the MODEMACRO variable with, 107
- scale factors
 - linetypes, 35
 - in a shape specification byte, 159
 - for text characters in linetypes, 32
 - text objects, 165, 168
- SCR (script) files, 123, 124
- Screen menu file section, 48, 49, 91
- screen menus
 - displaying, 96
 - enabling, 91
 - examples, 91, 93, 94
 - menu item labels for, 95
 - overwriting the previous menu, 94
 - section labels for, 92, 95
 - size limitations, 92, 96
 - submenu label format, 92
 - submenu replacement, 93
 - swapping screen submenus, 96
 - ZOOM command submenu, 94
- SCREENBOXES system variable, 92, 96
- SCRIPT command, 123
- scripts
 - 'GRAPHSCR command and, 124
 - 'TEXTSCR command and, 124
 - about, 3, 123
 - blank spaces in, 123
 - comment lines in, 123, 124

- scripts (*continued*)
 - continuously repeating, 128
 - creating, 123, 124
 - DELAY command and, 124, 127
 - displaying slides with, 120
 - double quotes in, 124
 - embedded spaces in file names and, 124, 125
 - examples, 124, 125, 127
 - preloading slides and, 126, 127
 - RESUME command and, 124
 - RSCRIPT command and, 124
 - running, 123, 125
 - running slide shows from, 126
 - undo feature and, 127
 - using double quotes in, 125
 - VSLIDE command and, 126
- SDI system variable, 140
- search paths (for program files), 4
- search paths (for support files), 4
- section labels
 - for screen menus, 91, 92, 95
 - for tablet menus, 97
- Select Objects prompt, canceling repetition of, 61
- semicolon (;)
 - in AutoLISP application files, 138
 - in command scripts, 123
 - in menu macros, 55, 57, 59
- separator lines on menus, 71
- SHAPE command, 154, 158
- shape definition files
 - about, 154
 - Big Font files, 166
 - compiling, 154, 155
 - creating, 154, 155
 - examples, 157
 - text fonts, 165
 - Unicode fonts and, 175
- shape descriptions
 - about, 155
 - fields described, 156
 - shape specification byte, 156, 157, 158
- shape names
 - in Big Font files, 168
 - in shape descriptions, 156
- shape numbers
 - in Big Font files, 168
 - in extended Big Font files, 169
 - in shape descriptions, 156, 160
 - in text fonts, 165
 - in Unicode fonts, 176
- shape specification bytes
 - about, 156
 - bulge-specified codes for, 163
 - Draw mode codes for, 159
 - flag vertical text code for, 164

- shape specification bytes (*continued*)
 - fractional arc code for, 162
 - location save/restore codes for, 160
 - octant arc code for, 161
 - size control codes for, 159
 - special codes for, 158
 - subshape code for, 160
 - vector length and direction codes in, 157
 - X-Y displacement codes for, 160
- shapes
 - creating, 2
 - data bytes required for description of, 156
 - including in linetypes, 34
- shortcut keys
 - about, 16
 - coordinate entry with, 101
 - creating, 18, 98
 - displaying on toolbar buttons, 11
 - removing, 18
 - table of, 16
- shortcut menus
 - about, 70
 - absolute referencing of, 77, 78
 - accessing, 70
 - cascading submenu indicators, 71
 - context-sensitive, 75
 - creating, 74, 75
 - inserting object menus into, 75
 - menu item limits, 70
 - relative referencing of, 77
 - separator lines on, 71
 - types of, 70
 - width of, 110
- SHORTCUTMENU system variable, 66
- SHP (shape definition) files, 154, 155, 177
- SHP (shape definition) fonts, 154
- SHX (compiled shape) files, 154, 155
- Single Object Selection mode, 61
- size control codes in shape specification bytes, 159
- slide files
 - image tile menus and, 87, 88, 90
- slide libraries
 - about, 120
 - creating, 88, 121
 - displaying a slide in, 122
 - examples, 89, 122
- slide shows, running from scripts, 126
- SLIDELIB (slide library creation) utility, 88, 121
- slides
 - about, 120
 - creating, 120, 121
 - editing commands and, 121
 - image tile menus and, 120, 121
- slides (*continued*)
 - preloading, 126, 127
 - remaking, 120
 - using command scripts with, 120
 - using in custom menus, 120
 - viewing, 120, 121
- small-image resource bitmap
 - for toolbar buttons, 84
 - for toolbar flyouts, 85
- spaces
 - in linetype definitions, 28, 30, 31
 - in menu macros, 55
- spacing between toolbar buttons, 11
- specbyte. *See* shape specification bytes
- special characters
 - allowed in menu macros, 60
 - for cascading submenu, 71
 - for labels in Pop menu sections, 72
- special codes for shape specification bytes, 158
- square brackets ([])
 - menu macros and, 59
- square toolbars, 11
- Start (Windows system command), 23
- startapp AutoLISP function, 131, 132
- status line
 - customizing, 2, 104
 - Help messages on, 97
 - MODEMACRO system variable and, 104
- strcat AutoLISP function, 106
- strlen DIESEL function, 116
- STYLE command, 154, 165, 173
- submenus
 - image tile submenus, 87
 - referencing, 63
 - screen submenus, 91, 92
 - special characters as indicators for, 71
- subscripts
 - adding font definitions for, 176
 - example, 177
- subshape code in shape specification bytes, 160
- subshape number
 - in Big Font files, 168
 - in extended Big Font files, 169
- substr DIESEL function, 116
- subtraction (-) DIESEL function, 111
- superscripts
 - adding font definitions for, 176
 - example, 177
- support files
 - changing the directory structure for, 4
 - library search path for, 4
 - organizing, 2, 4
- support folder, 4, 21, 25, 46, 50
- swapping menus. *See* menu swapping
- switches. *See* command-line switches

T

- tablet
 - menus
 - creating, 96
 - size limitations, 97
- TABLET command, 97
- Tablet menu file sections, 48
- templates
 - for the Publish to Web wizard, 2, 19
- terminators
 - for menu macros, 55, 63
 - for shape definitions, 168, 169
- text characters in linetypes
 - examples, 31
 - format for, 31
 - including, 31, 33
- TEXT command, 154, 165
- text fonts. *See* fonts
- text styles
 - defining, 154, 173
- time/date
 - formats, DIESEL function for, 113
- toolbar buttons
 - adding/deleting, 15
 - associating with different commands, 12
 - changing size of, 11
 - copying to other toolbars, 15
 - custom button images, 12, 13
 - defining, 84
 - flyouts. *See* flyouts (in toolbar buttons)
 - image resource bitmap options, 84
 - menu item macro option, 84
 - moving to other toolbars, 15
 - naming, 84
 - new buttons, 14, 84
 - repositioning, 10
 - restoring, 15
 - space between, 11
- toolbar flyouts. *See* flyouts (in toolbar buttons)
- toolbars
 - adding buttons to, 15
 - changing appearance of, 10, 11
 - characteristics of, 83
 - control elements for, 86
 - controlling across partial menus, 81, 84
 - coordinates specifications, 83
 - creating, 8, 9, 82
 - deleting, 10
 - floating
 - shape of, 11
 - large buttons, 11
 - naming, 83
 - new buttons for, 14
 - orientation, 83
 - recording changes to, 52
 - renaming, 9

- toolbars (*continued*)
 - rows specification, 83
 - shortcut keys for, 11, 18
 - tooltips on, 11
 - visibility, 83
 - See also* flyouts (in toolbar buttons)
 - See also* toolbar buttons
- Toolbars section of the MNU file
 - about, 47, 49, 82
 - menu item syntax for, 82
- tooltips, 11
- txt.shp* file, 165
- Type 1 PostScript fonts
 - compiling, 154, 155

U

- UCS control element of toolbars, 86
- undoing
 - command scripts and, 127
- Unicode fonts, 175
- Unicode shape definition files
 - font encoding, 175
 - font header syntax, 175
 - licensing information, 176
 - shape numbers, 176
 - subshape references, 176
- Unload option of ARX command, 145
- unloading
 - partial menus, 50, 53
- upper DIESEL function, 117
- user input
 - menu macros and, 56, 65
 - pausing for, 56, 99
- user-defined area on status line, 104
- user-defined bitmaps, 87
- user-defined toolbar buttons, 14
- utilities
 - running from within AutoCAD, 2

V

- VB. *See* Visual Basic
- VBA macros
 - running
 - from the command line, 135
- VBA projects
 - example, 136
 - loading, 135
 - loading/running automatically, 135
 - storing, 134
- VBA. *See* Visual Basic for Applications
- VBALOAD command, 135
- VBARUN command, 135
- vector length/direction in a shape specification
 - byte, 157
- vectors, specification codes for, 160

- vertical toolbars, 11
- view control element of toolbars, 86
- viewing
 - slides, 121, 122
- viewport scale control element of toolbars, 86
- visibility
 - of toolbars, 83
- Visual Basic for Applications, 130, 133
 - about, 133
 - ActiveX Automation and, 130, 133
 - advantages of, 134
 - AutoCAD VBA and, 133
 - developing with, 133
 - DVB files, 134, 135
 - loading projects, 135
 - loading/running projects automatically, 135
 - project compatibility issues, 135
 - project file storage, 134
- Visual LISP interactive development environment
 - about, 136
 - file format options, 137
 - namespaces in, 136
- VLISP. *See* Visual LISP
- VLX (Visual LISP executable) files, 137
- VSLIDE command, 121, 122, 126, 127

W

- walking menus. *See* pull-down menus; shortcut menus
- width
 - of menus, changing, 110
- Windows (operating system)
 - shortcut key assignments, 16
 - system commands, 23
- wizards
 - Publish to Web wizard, 2, 19

X

- X and Y origins of primitives
 - in Big Font files, 168, 169
- xor DIESEL function, 117
- xref control element of toolbars, 86
- X-Y displacement codes in shape specification
 - bytes, 160

Z

- ZOOM command
 - screen submenu, 94

