8-1998

# A Braille Translation Solution Using AutoCAD and LISP

Brian E. Bennett
*University of Northern Iowa*

# A Braille Translation Solution Using AutoCAD and LISP


## Abstract

The Americans with Disabilities Act of 1994 brought new requirements for architects, designers, and drafters to apply Braille symbol sets at various locations of public and private buildings. With production of many drawings using computer-aided drafting, the opportunity exists to automate this process, solve the problem of text to Braille conversion, and satisfy the requirements of federal regulation. This paper presents a LISP programing code structure for use within the AutoCAD drafting and design '\ software to solve the Braille symbol set translation problem with both 2D and 3D representations. An explanation of the logic and a detailed listing of the code is presented.

# A BRAILLE TRANSLATION SOLUTION USING AUTOCAD AND LISP

A Paper Submitted in Partial Fulfillment

of the Requirements for the Degree

Master of Arts in

Industrial Technology

Brian E. Bennett

University of Northern Iowa

August, 1998

Approved by:

| | |
|---|---|
| Dr. Charles D. Johnson, Advisor | 7-6-98<br>Date |
| Dr. Ahmed H. ElSway | 7-8-98<br>Date |

# ABSTRACT

The Americans with Disabilities Act of 1994 brought new requirements for architects, designers, and drafters to apply Braille symbol sets at various locations of public and private buildings. With production of many drawings using computer-aided drafting, the opportunity exists to automate this process, solve the problem of text to Braille conversion, and satisfy the requirements of federal regulation. This paper presents a LISP programing code structure for use within the AutoCAD drafting and design software to solve the Braille symbol set translation problem with both 2D and 3D representations. An explanation of the logic and a detailed listing of the code is presented.

# ACKNOWLEDGMENTS

This author would like to acknowledge the contributions of Dr. Charles D. Johnson who acted as advisor to my graduate studies and to this research. This author would like to acknowledge the efforts of Dr. Ahmed H. ElSway who acted as course instructor for this research project. And finally, this author acknowledges the support and direction provided by Dr. Michael R. White.

Table of Contents

# LIST OF FIGURE

# CHAPTER 1

# INTRODUCTION

The passage of the Americans with Disabilities Act (A. D. A.), in 1994, brought new requirements and opportunities for the architect and design drafter. One of the requirements established by the Americans with Disabilities Act is to apply Braille symbol sets at various locations of public and private buildings (American Institute of Architects, 1994). As the production of many drawing is now done with computer-aided drafting packages, the opportunity exists to automate the process of translating the required note or label into Braille.

In the last twenty years, technological advances have prompted a gradual transition, from a reliance on traditional drafting tools to the use of computer-aided design (Kashef, 1993). AutoCAD drafting software, produced by Autodesk, has emerged as the most popular drafting and design software for personal computers (Sanchez, 1998 & Abbott, 1998).

AutoCAD release 13 was used as the basis for this project. There have been fourteen versions of AutoCAD software, and this project will support the latest three versions: Release 12, 13, and 14. LISP is the programming tool integrated in AutoCAD. LISP is used to modify and enhance the AutoCAD software (Autodesk, 1992a & Autodesk, 1994a) by providing the command language and syntax for programming code.

## Statement of Problem

The problem investigated was how to use the programming capabilities of LISP to translate AutoCAD text into Braille symbol sets.

## Statement of the Purpose

The purpose of this project was to develop a LISP programming tool that designers and computer aided draftspersons, using AutoCAD, can use to translate the required AutoCAD text into a Braille symbol set. The resulting graphic Braille symbol set

was designed to satisfy the existing federal regulations for A. D. A. compliance in assisting the visually impaired. The common user of this investigation are the users of AutoCAD software: drafters, designers, and architects. In meeting the regulations of the federal act, the designer of public structures has two general alternatives. The designer can write specifications defining the necessary results, which allows third party manufacturers and producers to accomplish the translation and part production, or design the required Braille symbol set results directly into the drawing documentation. In either case the translation of AutoCAD text into Braille symbol sets must be accomplished.

## Statement of Need

The need for this research was based on several factors. First, there are federal requirements, based on the American's with Disabilities Act of 1994, for designers to apply Braille annotated labels and directions to signs in public areas (American Institute of Architects, 1994). Second, it is uncommon for design and drafting people to have a working knowledge of the Braille symbol set or the translation from English into Braille symbol sets. Third, with all of the features and flexibility of the AutoCAD software, the system does not have the built-in capacity to produce Braille symbol sets.

## Research Questions

1. Can a LISP programming code be developed to automate the translation of AutoCAD text into Braille symbol sets?
2. What is an appropriate graphic user interface for the integration of the Braille LISP routine into the AutoCAD environment?
3. What customization features are required to seamlessly integrate the Braille LISP routine into AutoCAD?

## Delimitations

The project is delimited to working with AutoCAD Release 12, 13, and 14 drafting software. There were major changes in the source code structure of AutoCAD prior to

Release 12. These changes in the source code make translation to versions before release 12 very difficult. Third party software using AutoCAD have been updated to use at least Release 12. Currently, Release 14 is the latest version available from Autodesk.

In order to shorten the length and physical volume of text written in Braille, a system of one hundred ninety contractions and short form words have been standardized (Wise, 1975). This is commonly referred to as Grade 2 Braille. These contractions and short form words are in addition to the direct translation of the individual English characters. The Braille.lsp program is intended to be used by architects and designers who need literal translations for notes, labels, and markers. It is not the intention to translate long prose or extended text. This research involved developing a code to construct literal translations of AutoCAD text. The contractions and short form words can be entered as such, but there is no attempt to directly recognize the longer word or phrase and translate to the contraction or short form word.

There are some AutoCAD text characters for which there are no Grade 1 Braille equivalents. These characters include: @, <, >, {, }, [, ], `, |, ~, \ ,and underscore . The Braille.lsp program is designed to recognize these characters. The program creates a blank space between adjacent characters for these untranslated characters. By recognizing these characters, 'crashes' of the program can be avoided when these characters are encountered by the program, but the constructed space allows the problem to be recognized by the reader.

With the release of AutoCAD R13, Autodesk introduced the MTEXT command. The MTEXT command is used to construct multi-lined text in the AutoCAD environment. The intent of this project is to provide translation for notes and other single line text. In AutoCAD's execution of the MTEXT command "{. . .}//P" is added to the text string. This addition cannot be recognized as different from the original text. This project does not support translations from the AutoCAD command MTEXT. Braille.lsp does

recognize an MTEXT entity. The program does not try and translate the text, instead, the program prompts the user to enter the necessary text. Text generated by either the AutoCAD TEXT or DTEXT command is supported by this project.

As Mackenzie (1953) illustrates, there are standard Braille translations for many of the languages of the world. It is not the intention of this project to resolve the problem of translation from languages, other than English, into Braille. The non-English translation problem can be partially solved through a language translator. Once in English, the Braille translator can be employed to produce the required Braille symbol set.

## Assumptions

Utilizing the LISP program developed through this research assumes the user has a competent knowledge of AutoCAD commands, graphic user interface, and AutoCAD command structure existing in the AutoCAD software (Autodesk , 1994b). Although the user is not modifying the AutoCAD program itself, the user is initiating commands that are designed to imitate a methodology similar to AutoCAD.

Utilizing the developed LISP program, Braille.lsp, assumes a working knowledge of the operating system and file management available on the user's computer. File management involves the ability to create, designate, modify, and reference various subdirectories, folders, and files. The successful use of this programming tool requires the storage of programming code within the computer, while allowing access to the files containing the code and required drawing.

It is assumed that the user is capable of proper storage, loading, and execution of LISP programming code. The Autodesk reference material provides introduction and instruction to the construction and use of LISP programming code.

The standard Braille system has been expanded to include music and mathematical applications. This project does not intended to support tranlations of music or mathmatical notations.

## Definition of Terms

AutoCAD, as used in this study, is a full featured, open architecture, software program used in computer-aided drafting and design (Autodesk, 1994b). The programming code resulting from this project takes advantage of many of the features and commands available within AutoCAD. The open architecture allows the user to access and manipulate the command structure and data bases of both the AutoCAD software and drawing files of the user.

ASCII is the American Standard Code for Information Interchange. This organization establishes a standard for recognizing and manipulating English characters within LISP programs. The standardization of the ASCII code allows many different computer platform configurations to run AutoCAD. This projects resulting code will run on any of these platforms and configurations.

AutoLISP is a programmable interface subset of the XLISP programming language developed by David Betz. AutoLISP is integrated into AutoCAD as a programming tool (Autodesk, 1992b). Autodesk has expanded the LISP command set for use in AutoCAD. For the purpose of this study, AutoLISP and LISP are used interchangeably.

The Graphic User Interface (GUI) is that part of the AutoCAD software that controls how the computer interacts with the user. This occurs in two areas: a. presentation of graphic images, information, and prompts on the monitor, and b. user responses to those prompts in the form of keyboard entry and digitizer or mouse selection.

AutoCAD pull-down menus are part of the computer system graphic user interface. Pull-down menus are represented by headers across the top of the drawing editor area, in the menu area. When the header is selected the remainder of the menu is 'pulled down' into the drawing area for additional selection by the user. Selection of a

tool or command is accomplished by 'left button clicking' on the mouse or digitizer puck. Each of these menu areas will cascade to a variety of submenus.

The AutoCAD command line area is displayed across the bottom of the screen area. Depending on the user configuration, one or more lines may be displayed. In this area the software presents prompts for the user and displays the keyboard entry of the user.

Acad.mnu is the AutoCAD menu source code (Autodesk, 1994b, Autodesk, 1992a, & Autodesk, 1994a). This file contains the command strings and macro syntax language that define and format the AutoCAD commands as they are presented to the user.

Braille is a system of embossed symbolic writing developed by Lois Braille in the early 1800s. The original intent of this language was to assist the blind in obtaining access to written literature. This application has been extended to the areas of mathematics and music. Although not a true translation, the Braille language has been developed for many spoken and written languages of the world (Garland, 1963).

A Braille symbol cell is a six location matrix. The arrangement of dots within this matrix are the representations of individual letters, numbers, punctuation, and contractions of words and phrases.

The Braille symbol set is a series of individual Braille symbol cells representing multiple characters. These multiple characters may be words, phrases, numbers, punctuation, or any combination of the above.

# CHAPTER 2

# REVIEW OF LITERATURE

## Braille

The writing system that bears his name today is the result of Louis Braille. While working and living at the French Royal School for the Blind in 1829, Braille proposed a new system of embossed writing for the blind. The system was composed of six dot locations, arranged in a double column of three. Individual letters, numbers, and punctuation could, in nearly all cases, be represented with different combinations and arrangements of dots at these six locations. The system has been expanded as new elements of punctuation and syntax have entered into common use (Wise, 1965, Harley, Henderson, & Truan, 1979).

The Braille system of embossed writing has been the universal entry, for the blind, to literature for nearly one hundred years (Garland, 1963). After the introduction there were many attempts to alter the system. There have also been numerous independent and competing systems. These alternatives had some short term popularity, but two factors contributed to the continued dominance of the Braille system of embossed writing: a. There was substantial infighting and bickering among those that proposed changes. b. The Braille system was more versatile than any other system. Although written originally for the English language, the Braille system has been adapted to many other languages and scripts of the Americas, European, Asian, and African cultures (Mackenzie, 1953 & Harley, et al., 1979).

While the Braille system is in wide use, there are problems in its application to the spoken and written languages. First, there is a reliance on preexisting alphabets in the determination of the translation. The sequencing or logical progression of the dot patterns do not match the sequencing and order of the alphabet. This problem is particularly true for languages other than English (Mackenzie, 1953). Second, the frequency that the letter

is used does not match the complexity of the dot pattern. Some letters that have a high frequency of use have a complex dot pattern. Some letters that have a low frequency of use have a simple dot pattern. This discrepancy increases the task of constructing the embossed writing as well as increasing the difficulty of reading the result. Third, the Braille system is organized to ease the teaching and construction of the individual characters. This construction is not arranged for the ease and speed of reading the Braille system (Harley, Henderson, & Truan, 1979). There are been attempts to improve this situation with the introduction of short form words and contractions of words (Wise, 1965 & Harley, et al., 1979).

With the development of the computer and software packages, there have been numerous attempts at creating programs for character recognition. Through the middle 1960s these efforts were of limited success, and then only with restricted vocabulary such as the names of cities in the United States (Rabinow, 1963). This situation has improved through the development of more sophisticated computer software and more powerful computer hardware. However, the programs still only convert spoken words into written language or written language into spoken word.

## AutoCAD Graphic User Interface

The control and manipulation of AutoCAD software, and ultimately the drawing database, is accomplished through the AutoCAD user interface menus. Autodesk allows the menus of AutoCAD to be accessed and modified. This feature, referred to as open architecture, was available early in AutoCAD release history. This ability to modify the user interface has been a great advantage in the personalization of AutoCAD and significantly contributes toward the software being the most popular personal computer-based drawing package (Autodesk, 1994a, Head, 1987, & Stellman, 1990). The ability to quickly access and accomplish time consuming and repetitive tasks has been the result of combining customized menus and introduced LISP programming code. The customized

menu consists of a menu item label and the associated command definition (Tickoo, 1996, Abbott, 1998, & Sanchez, 1998).

The syntax for creating customized menu files has been changed from Release 12 to Release 13 and these changes are covered in the AutoCAD customization manuals (Autodesk, 1994a). The differences are reflected in the differences between the changes for integrating the Braille.lsp into the Acad.mnu file for AutoCAD R. 12 and AutoCAD R. 13.

# CHAPTER III

# METHODOLOGY

This chapter describes the tools: computer hardware, AutoCAD software, Braille language, and LISP programming tools that were used in the investigation of this problem. It is the combination and interplay of these tools that resulted in the finished product, Braille.lsp, which can be used by the draftsperson to produce the Braille symbol set.

## Hardware

The hardware used in this research was a Pentium-based 133mHz computer. The system includes 32 Mb of ram and a 1.4 Gb hard drive. Monitor, video card and other peripherals common to the computer industry complete the computer hardware, and matching both the requirements of the previously detailed hardware and the AutoCAD software. The minimum capabilities of the computer are established by Autodesk (1994b) as required hardware parameters for using the AutoCAD software.

## Software

The computer operating system used in this study is the Microsoft Windows 95 operating system using the default installation and operating characteristics. Microsoft Windows 95 operating system is recognized and approved by Autodesk for running the AutoCAD drafting program. Autodesk has produced many different versions of AutoCAD. The different versions allow the user many choices of operating systems for their computer. That user choice of operating system will make no difference in the appearance, use, or results of Braille.lsp.

The drafting software used in this study was AutoCAD R12c3, AutoCAD R13c4, and AutoCAD R14. Integrated within the code of AutoCAD, the LISP programming tools are available in all of the releases of AutoCAD. The Braille.lsp program is designed to perform identically on all releases, with the exception of the AutoCAD command MTEXT, introduced in Release 14.

## Braille Symbol Cell

The Braille symbol cell, shown in FIGURE 1, consists of six dot locations (hereafter referred to as 'dot' or 'dots') arranged in two columns of three dots in a single plane. The dots are numbered one through six, from the upper left dot. Working down the column to dot three and back up to the top of the right column is dot four. Dot six is in the lower right corner. It is the number and arrangement of the dots, within this framework, that determine the letter, word, phrase, number, or punctuation being expressed. The dots are 0.05 inches in diameter, spaced 0.09 inches between centers, and raised 0.025 above the surface in three dimensional applications. The three dimensional Braille dot is based on a hemisphere. The individual cells are horizontally spaced every 0.35 inches (Harley & Henderson, 1979). Braille symbol sets are designed to be read horizontally from left to right.

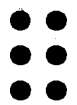


FIGURE 1: Braille Symbol Cell

## Braille Code

The flow and logic of Braille.lsp is laid out in Appendix A. There are three general tasks that must be accomplished to generate the Braille symbol set.

The first task of Braille.lsp is to control the environment that the program runs in. The system variables contained within the AutoCAD software control the appearance and operation of the software. The values contained in these variables can be stored and modified within the Braille.lsp program. Another part of the environmental control is to prompt the user, through the graphic user interface, for the necessary input data, recognize the data type, and store that data in specific variables. After the program has

constructed the Braille symbols, the program must return the system variables to the same values as they were prior to the execution of the program.

The second task of the Braille.lsp program is the recognition of an individual text character and relating that character to the construction of the individual Braille symbol cell. This relationship can be established through a binary list, unique for each text character. The binary list provides an easy yes/no test for constructing the dots of an individual Braille symbol.

The third task of the Braille.lsp is the construction of each individual symbol cell. This construction of the Braille symbol cell is based on the binary list and the calculated location of each dot location within that cell. The user of Braille.lsp should have the option of entering the text to be translated or selecting existing text to be translated. If text is entered from the keyboard, the Braille symbol set should be constructed in the existing user coordinate system. If existing text is selected, the Braille symbol set should be constructed in the coordinate system of the existing text.

The program should then perform a recursive loop for constructing a Braille symbol cell for each character in the text string. The program should recognize the end of the text string to be translated so that the recursive loop can be terminated.

# CHAPTER IV

# RESULTS

## Braille Code

The result of this research is the LISP code listed in Appendix B containing the Braille.lsp code. The controlling program is C:Braille.lsp. Within C:Braille.lsp are two separate, but symbiotic programs: Makecell and Bcode. Makecell and Bcode are recursive or looping programs to do repeated calculations and constructions of the individual Braille symbol cell. The primary functions of the C:Braille.lsp program are to control the 'footprints' of the program and gather the necessary information from the user.

The Bcode program uses a conditional test to match the individual text character with a binary list. This program uses a recursive loop to work through each character, in a text string. A text string is entered by the user from the keyboard or selected by the user from existing text in the drawing.

The Makecell program uses information and data from both Braille.lsp and the Bcode program. The Makecell program calculates the position of all six dots within an individual cell and then uses the binary list from the Bcode program to determine whether to place two and three dimensional dots at each of the six dot locations. Part of the Makecell program searches for a layer named 2DBRAILLE. When the program finds the layer 2DBRAILLE, the program changes the current layer to 2DBRAILLE. The program then loops through creating the dot pattern necessary. The program then repeats the process to determine the existence of a layer named 3DBRAILLE. Again, when the layer is found, the program causes 3DBRAILLE to be the current layer. The Makecell program uses the same binary list to decide whether to insert a three dimensional dot at each of the six locations.

While the C:Braille.lsp program runs, it may change certain system variables from the existing conditions that the user has previously specified or existing default conditions.

These changes mask the operation of the program. Within the Braille.lsp program, the condition of the variables that are going to be altered are first stored. The program alters the variable as it executes the Braille program. At the conclusion of the program execution, the Braille program recalls the stored variables and restores any system variables so there are no 'footprints' left from the execution of the program. This process is accomplished through two LISP commands: getvar and setvar. The (getvar ... command is used to retrieve the value of a system variable. This system value is stored in a specified variable. The (setvar ... command, is used to establish or set the value of a system variable. First the (getvar ... command, then (setvar ... command, and finally (setvar ... command again, to reestablish an environment that the user had before the execution of Braille.lsp. The user of the program can then expect the software will work the same after the execution of the program as it did before the execution of the program. Areas where these changes can occur are: BLIPMODE, SNAPMODE, and ORTHOMODE. Braille.lsp also searches the layer table to determine the existence of two layers: 2DBRAILLE and 3DBRAILLE. If these two layers are not found the program creates them. This process allows the Makecell program to isolate the Braille symbol set entities from other elements of the drawing.

The second function of C:Braille.lsp is used to gather the necessary information and data from the user. The user must decide whether to enter text via the keyboard or select existing text from the drawing. A conditional test for 'E'ntry or 'S'elect controls the resulting data entry.

A (getstring ... command stores keyboard text entry in the variable CHRSTR. The value contained in this variable is passed on to the BCODE program.

The (entsel ... command stores selected text in the same variable CHRSTR. Braille.lsp tests for the delimited AutoCAD text type: MTEXT. A positive test result produces a prompt to enter the text instead of selecting the text. If the selected text is not

an MTEXT entity, the value of the selected text is stored in the variable CHRSTR. The value of CHRSTR will be passed on to the BCODE program.

After the selection of the text, the user coordinate system is adjusted to match the selected text. This adjustment allows the Braille symbol set to be drawn in the same plane as the selected text. After the Braille symbol set is constructed, the user coordinate system is returned to the previous alignment. The starting location for the constructed Braille symbol set is determined by the (getpoint ... command. This command establishes the upper left corner in the construction of the Braille symbol cell. This data is passed to the Makecell routine and used as the starting point to construct the first Braille symbol cell. The Braille.lsp program update the location of this point for the construction of each successive Braille symbol cell.

In the Bcode program, a conditional test, (cond ..., command compares the individual text character with an ASCII code number. When a match is found, the variable replaces the ASCII character code with a binary list (e.g. '1'0'0'1'1'0 for the letter 'd'). This process is repeated for each character in the text string entered though the keyboard or selected from the screen. The Makecell program calculates the position of all six dots within an individual cell. The location of the first cell is based on the user response to the (getpoint ... command. Makecell then uses the results of the Bcode program to determine whether to place a dot at each of the six dot locations. After constructing the cell for the last character in the text string, the Makecell and Bcode programs stop the recursive loop and C:Braille.lsp program exits.

Within any program, the programmer must try to anticipate, possible 'wrong ' or 'unexpected' data entry by the user (Head, 1987 & Head & Head, 1989). The process of error trapping allows the programmer to keep the program working by repeating a request or prompt for necessary input until 'correct' and complete data is given to the computer. The user can feel free that mistakes will not 'crash' the program or harm their

drawing. In this program, error trapping occurs in three areas. The 'UNDO, M' command, near the beginning of the program, establishes a benchmark so that data created before the execution of the C:Braille.lsp program will not be disturbed. Within the program, the (initget ... command restricts the acceptable user input and will repeat prompts until properly satisfied. Any variables that were created within the program are deleted as local variables at the end of the program execution. This elimination of variables allows other routines to use the same variable name without adverse effect.

These error trapping features constrain C:Braille.lsp. Any results of using C:Braille.lsp are isolated from affecting other programs or previous work during the execution of the program and after the program has completed the construction of the Braille symbol set.

## Format and Presentation

The AutoCAD software can prompt the user for input from the command line or dialog boxes. The presentation of this programming tool, through the graphic user interface, is designed to be in a format familiar to the AutoCAD software user. As a result of the design of this programming tool, the user is prompted for data in the familiar AutoCAD style of keyboard text entry and mouse or digitizer point selection , providing a seamless exchange and short learning curve.

Depending on how the Braille programming tool is installed on the AutoCAD system, the user may have to manually load and execute the command, or use customized pull down menus. The Braille program can be installed so that the user recognizes the command as part of the AutoCAD command set.

To use the Braille.lsp program manually, at the command line, the user types (load "drive:/directory location of braille.lsp/Braille"). The computer will search the drive and subdirectory entered, find the Braille.lsp program and load the program into resident memory. The file extension (xxx.lsp) is used to identify the type of file, and does not have

to be entered to load or execute the program. After loading the program, the user types the name of the routine (Braille) to execute the program. The program only needs to be loaded once during each drawing session. Once loaded, the program can be used at any time by entering the name of the program: Braille.

Release 14 of AutoCAD does allow LISP programs to be used in different drawings within the same drawing session without loading the program for the new drawing data base. This eliminates the need to reload the program when a new drawing file is opened. The user will still be required to call the routine for execution each time. As the Braille program runs, the user will be prompted for three pieces of information: the method of text entry, the entry or selection of that text, and the starting location of the Braille symbol set.

Users of the AutoCAD software communicate with the software through the Graphic User Interface (GUI). Prompts are displayed on the monitor for user response. Mouse selection is recorded and used by the software to execute commands. Keyboard entry is displayed on the monitor and used by the computer to execute commands of the software. All of these things occur through the Graphic User Interface. In AutoCAD, this GUI is controlled and defined by the Acad.mnu file. Acad.mnu is an ASCII text file that forms the basis of open architecture in AutoCAD. Autodesk encourages the users of AutoCAD to investigate and modify this file (Autodesk , 1992a & Autodesk , 1994a). By modifying this file, the user can customize AutoCAD to personal or corporate needs (Head, 1987, Raker & Rice, 1989, & Jansen, 1997).

The process of loading and executing the Braille.lsp program can be integrated into the Acad.mnu file. By integrating the Braille.lsp program into the Acad.mnu, the appearance is seamless to the user, and similar in presentation and format to existing AutoCAD commands.

The modification of the Acad.mnu menu file occurs in: line 265 of the AutoCAD Rel. 12 Acad.mnu file, line 385 of the AutoCAD Rel. 13 Acad.mnu file, and line 456 of the AutoCAD Rel. 14 Acad.mnu file. Changes to each of the files are the similar, and are detailed in Appendix C.

The text inside the square brackets (i.e. [ ]) is what will appear in the pull down menu. This is the label of the command that will be selected by the user to load and execute an AutoCAD command. In the case of this project, the command label is: Braille Text. The menu selection first cancels out all other commands. The programming code then tests to determine if the Braille program is already in resident memory. If the Braille program is already loaded in resident memory, the program is executed. If the program is not in resident memory the Braille program is loaded into resident memory and then executed. By first checking to see if the program is in resident memory, the Braille program is loaded only on the initial execution of the program.

Shown in Appendix D, is the Braille Text label as it would be presented to the user on the screen for AutoCAD Release 12, Release 13, and Release 14. Access to the Braille.lsp is contained in a text submenu within the Draw pull-down menu. The Draw pull down menu is selected from the AutoCAD menu bar above the drawing area. The selection of the Text command displays a cascading second menu containing the Braille Text label. The Braille.lsp program was placed in this location because of the similarity and relationship to other existing AutoCAD text commands. Selection of the Braille Text label will execute the Braille.lsp routine.

## Results Summary

The result of this research project is the Braille.lsp program, shown in Appendix B. The utilization of this LISP program does successfully translate AutoCAD text into a Braille symbol set. The construction of Braille symbol set is in the current user coordinate system when the text is entered from the keyboard. The construction of the Braille

symbol set is in the user coordinate system relevant to the AutoCAD text when existing text is selected.

The Braille.lsp LISP Code, in Appendix B, is written so that the program and necessary drawings and menu files are contained on a disk in drive A. Users of the program can move the necessary files to various locations on the user's computer. When these files are moved, changes in the drive and directory listings in the programming code and menu files will have to be made to accommodate the placement of the program and menu files.

The graphic user interface prompts the AutoCAD user for input from either a dialog box or command line entry. There are three areas for this entry: (a) the type of entry, (b) the text to be translated, and (c) the location of the constructed Braille symbol set. The familiarity and simplicity of using the command line for text entry were used in the results of this research. The use of a mouse or digitizer for text selection and Braille symbol set location are consistent with solutions to similar user requirements in AutoCAD, and used in this research project.

The open architecture feature of AutoCAD allows the user to customize the software. This research, detailed in Appendix C, results in the modifications to the Acad.mnu file for Rel. 12, Rel. 13, and Rel. 14. These modifications successfully integrate the Braille.lsp program into the AutoCAD software. When access to the Braille.lsp program is integrated into the Acad.mnu file the user has direct access to the program under the Draw pull-down menu and Text submenu. The results of this integration are fewer mistakes to load and execute the Braille.lsp program, and quicker access to the Braille.lsp program.

# CHAPTER V

# CONCLUSIONS and RECOMMENDATIONS

## Conclusions

The user of AutoCAD is already familiar with the graphic user interface provided by AutoCAD. The Braille.lsp uses the existing graphic user interface tools: a monitor for displaying text and prompts, keyboard to enter text, and mouse or digitizer to make selections. The syntax and format of Braille.lsp are similar to the format and syntax of existing AutoCAD commands. The open architecture of AutoCAD allows that integration to appear seamless. The appropriate integration of Braille.lsp into the existing graphic user interface provides the least disruption in the efficient use of AutoCAD. That familiarity with the AutoCAD graphic user interface will shorten the learning curve for new users of the program.

Rather than require the AutoCAD user to learn Braille or attempt an infrequent manual translation, the combination of computer and software provide an opportunity to automate the process of translating AutoCAD text into the Braille symbol set. The use of LISP within AutoCAD provides a solution to one of the new requirements involved in current design and drafting practice. The Braille programming tool provides many advantages. Errors that could be introduced during a manual translation can be reduced. The design portion of the manufacturing time frame necessary for producing the tag or label can be reduced. Incomplete or incorrect translations that could cause an increase in the insurance liability exposure, can be reduced through an automated process. By automating the design of the dot pattern, it is easier for the designer to meet the Federal A. D. A.. requirements with an inexpensive solution.

The results of the Braille.lsp program produce both two-dimensional and three-dimensional figures. Each of these figures is constructed on separate layers and can be isolated for different uses. Printers and others who need two dimensional shapes for

masks have access to direct, full size images. Machinists and rapid prototype service companies have access to full size three-dimensional representations of the same Braille symbol set text.

The visually challenged also benefit from this programming tool. If support, in the form of Braille notes, labels, and tags, is easier to design, easier to produce and place, and less expensive to produce, there should be more of that support. The federal regulations will be easier to satisfy, both in spirit and in fact. In addition the Braille.lsp can be integrated with virtual reality applications for the teaching and training of the visually challenged without the need to produce physical text copies.

In conclusion, each of the research questions was successfully resolved. A LISP programming code was developed to translate AutoCAD text into Braille symbol sets. Prompts in the command line, presented by the Braille programming code, provide an appropriate graphic user interface between the user and the Braille programming code. The customization of the Acad.mnu file allows the Braille.lsp program to be integrated into the AutoCAD software.

## Recommendations

The purpose of this study was to investigate an approach for the conversion of AutoCAD text entities into Braille symbol sets. This program has been designed to perform a literal translation of the text into Braille. Further research should be done to recognize the contractions and short forms of words and phrases that are incorporated into Grade 2 Braille. If these contractions and short forms of words can be recognized, the use of this program could be extended into Grade 2 Braille.

The AutoCAD drafting software package has had fourteen major versions released to the public. Minor changes that have been incorporated into the program, as shown by designations Rel. 14-c4, have not necessitated a major new version or release. The Braille programming code generated by this study is designed to be seamlessly

incorporated into the standard body of AutoCAD code. The bundling of this programming code with AutoCAD software would facilitate the use of the code by any user of the AutoCAD software and increase the compliance with Federal A. D. A. requirements.

There are recognized standard translations from many languages into Braille symbol sets (Mackenzie, 1953). The AutoCAD software is available and used worldwide. The logic and structure of the Braille.lsp programming code from this study should allow programmers to translate languages other than English. The same structure and format of the Braille.lsp code can be used to form the code basis for this new translation.

# REFERENCES

Abbott, D. (1998). Customizing AutoCAD part 2. In CADENCE, 13 (2), 42-48.

American Institute of Architects (1994). Americans with disabilities act. Washington, DC: Author.

Autodesk (1992a). AutoCAD customization guide (Rel. 12). Sausolito, CA: Author.

Autodesk (1992b). AutoLISP programmers reference manual (Rel. 12). Sausolito, CA: Author.

Autodesk (1994a). AutoCAD customization guide (Rel. 13). Sausolito, CA: Author.

Autodesk (1994b). AutoCAD users guide (Rel. 13). Sausolito, CA: Author.

Garland, C. W. (1963). Solid dot printing. In Clark, L. L. (Ed.), Proceedings of the International Congress on Technology and Blindness, Vol. III (pp. 129-138). New York: The American Foundation for the Blind.

Harley, R., Henderson, F., & Truan, M. (1979). The teaching of braille reading. Springfield, IL: Charles Thomas.

Head, G. (1987). AutoLISP in plain English. Chapel Hill, NC: Ventana Press.

Head, G., & Head, J. (1989). 1000 AutoCAD tips And tricks (fifth ed.). Chapel Hill, NC: Ventana Press.

Jansen, E. (1997). Customizing the acad.lsp file. In CADENCE 12 (10), 48-56.

Kashef, A. E. (1993) A comparison of the effectiveness between computer aided drafting and the traditional drafting techniques as methods of teaching pictorial and multi view drawing. A paper presented at the American Vocational Association Convention, Nashville, TN.

Mackenzie, C. (1953). World Braille usage. Paris, France: UNESCO.

Rabinow, J. (1963). Some logic presently used in character recognition machines. In Clark, L. L. (Ed.), Proceedings of the International Congress on Technology and Blindness, Vol. 1, (pp. 245-259). New York: The American Foundation for the Blind.

Raker, D., & Rice, H. (1989). Inside AutoCAD: The complete guide to AutoCAD. Thousand Oaks, CA: New Riders Publishing.

Sanchez, D. (1998). CADENCE tutorial: Customizing AutoCAD part 1 (menus). In CADENCE, 13 (1), 40-46.

Stellman, T. A. (1990). Practical AutoLISP. Albany, NY: Delmar Press.

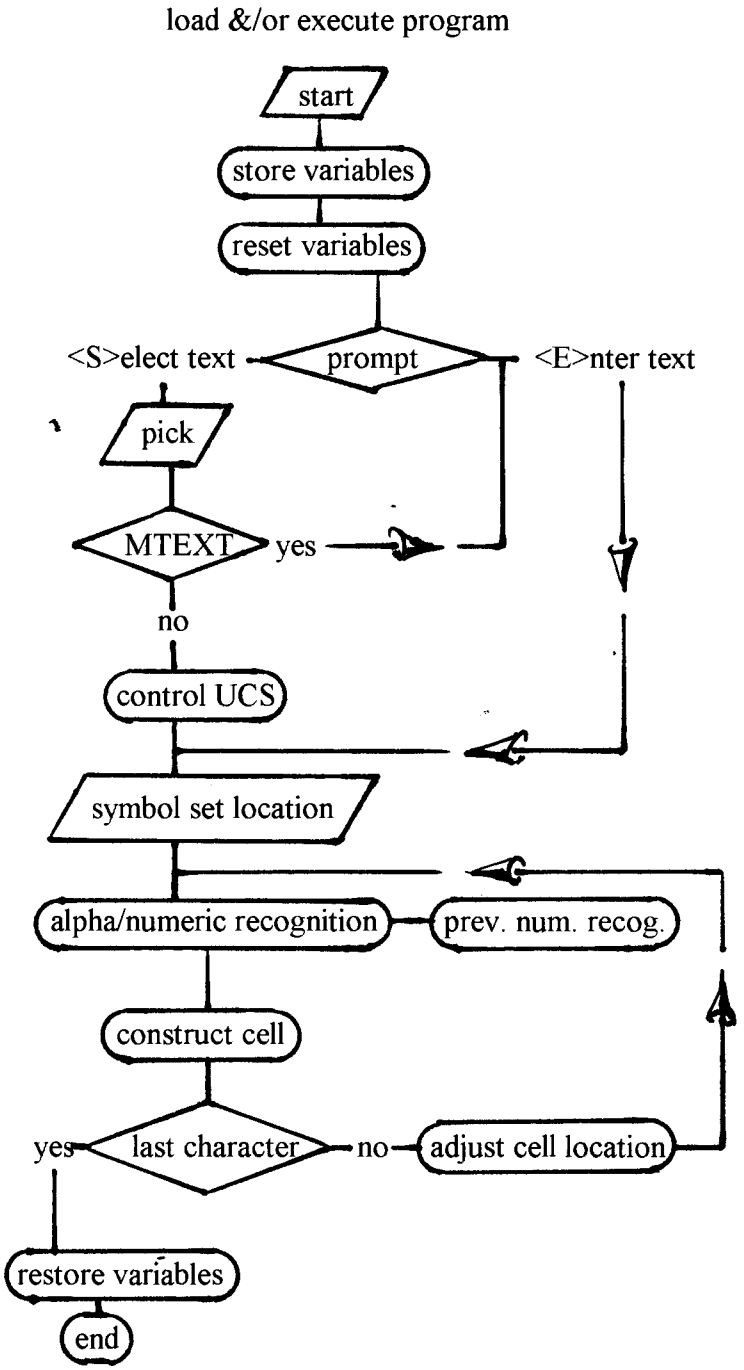
Tickoo, S. (1996). Customizing AutoCAD for dos and windows. Albany, NY: Delmar Publishers.

Wise, J. (1965). Dot writing (fourth ed.). New York, NY: Janet Wise Publishing.

# APPENDIX A

# PROGRAMMING LOGIC

Program Logic




LISP code location

(if (not C:BRAILLE . . .

(Braille) or [Braille Text]

(setq . . . (getvar . . .

(setvar. . .

(getkword . . .

(entsel . . .

(getpoint . . .

(BCODE . . .

(MAKECELL . . .

(repeat (strlen . . .

(setvar . . .

# APPENDIX B

## BRAILLE.LSP LISP CODE

```
;;****************** BRAILLE.LSP **********************
;;Convert text to Braille symbols. by Brian Bennett

;;construct each braille symbol cell
(defun MAKECELL (TXT / DT1 DT2 DT3 DT4 DT5 DT6
                       DT7 DT8 DT9 DT10 DT11 DT12)
 (setq TXT (BCODE TXT))          ;;obtain binary list for character
 (cond ((/= (nth 11 TXT) nil)
   (progn
     (setq DT1 SP              ;;establish dot locations 11 digit
         DT2 (polar DT1 4.7123889 0.1)
         DT3 (polar DT2 4.7123889 0.1)
         DT4 (polar DT1 0.0 0.1)
         DT5 (polar DT2 0.0 0.1)
         DT6 (polar DT3 0.0 0.1)
         DT7 (polar DT1 0.0 0.35)
         DT8 (polar DT2 0.0 0.35)
         DT9 (polar DT3 0.0 0.35)
         DT10 (polar DT4 0.0 0.35)
         DT11 (polar DT5 0.0 0.35)
         DT12 (polar DT6 0.0 0.35))
     (command ".LAYER" "S" "2DBRAILLE" "")
     (if (= (nth 0 TXT) 1)(command ".DONUT" 0.0 0.05 DT1 ""))
     (if (= (nth 1 TXT) 1)(command ".DONUT" 0.0 0.05 DT2 ""))
     (if (= (nth 2 TXT) 1)(command ".DONUT" 0.0 0.05 DT3 ""))
     (if (= (nth 3 TXT) 1)(command ".DONUT" 0.0 0.05 DT4 ""))
     (if (= (nth 4 TXT) 1)(command ".DONUT" 0.0 0.05 DT5 ""))
     (if (= (nth 5 TXT) 1)(command ".DONUT" 0.0 0.05 DT6 ""))
     (if (= (nth 6 TXT) 1)(command ".DONUT" 0.0 0.05 DT7 ""))
     (if (= (nth 7 TXT) 1)(command ".DONUT" 0.0 0.05 DT8 ""))
     (if (= (nth 8 TXT) 1)(command ".DONUT" 0.0 0.05 DT9 ""))
     (if (= (nth 9 TXT) 1)(command ".DONUT" 0.0 0.05 DT10 ""))
     (if (= (nth 10 TXT) 1)(command ".DONUT" 0.0 0.05 DT11 ""))
     (if (= (nth 11 TXT) 1)(command ".DONUT" 0.0 0.05 DT12 ""))
     (command ".LAYER" "S" "3DBRAILLE" "")
     (if (= (nth 0 TXT) 1)
       (command ".INSERT" "A:/DOT" DT1 1 "" ""))
     (if (= (nth 1 TXT) 1)
       (command ".INSERT" "A:/DOT" DT2 1 "" ""))
     (if (= (nth 2 TXT) 1)
       (command ".INSERT" "A:/DOT" DT3 1 "" ""))
     (if (= (nth 3 TXT) 1)
       (command ".INSERT" "A:/DOT" DT4 1 "" ""))
     (if (= (nth 4 TXT) 1)
```

```
    (command ".INSERT" "A:/DOT" DT5 1 "" ""))
  (if (= (nth 5 TXT) 1)
    (command ".INSERT" "A:/DOT" DT6 1 "" ""))
  (if (= (nth 6 TXT) 1)
    (command ".INSERT" "A:/DOT" DT7 1 "" ""))
  (if (= (nth 7 TXT) 1)
    (command ".INSERT" "A:/DOT" DT8 1 "" ""))
  (if (= (nth 8 TXT) 1)
    (command ".INSERT" "A:/DOT" DT9 1 "" ""))
  (if (= (nth 9 TXT) 1)
    (command ".INSERT" "A:/DOT" DT10 1 "" ""))
  (if (= (nth 10 TXT) 1)
    (command ".INSERT" "A:/DOT" DT11 1 "" ""))
  (if (= (nth 11 TXT) 1)
    (command ".INSERT" "A:/DOT" DT12 1 "" ""))
  (setq SP DT7)
 ))
 ((= (nth 6 TXT) nil)
 (progn
  (setq DT1 SP          ;;establish dot locations 6 digit
      DT2 (polar DT1 4.7123889 0.1)
      DT3 (polar DT2 4.7123889 0.1)
      DT4 (polar DT1 0.0 0.1)
      DT5 (polar DT2 0.0 0.1)
      DT6 (polar DT3 0.0 0.1))
  (command ".LAYER" "S" "2DBRAILLE" "")  ;;2D construction
  (if (= (nth 0 TXT) 1)(command ".DONUT" 0.0 0.05 DT1 ""))
  (if (= (nth 1 TXT) 1)(command ".DONUT" 0.0 0.05 DT2 ""))
  (if (= (nth 2 TXT) 1)(command ".DONUT" 0.0 0.05 DT3 ""))
  (if (= (nth 3 TXT) 1)(command ".DONUT" 0.0 0.05 DT4 ""))
  (if (= (nth 4 TXT) 1)(command ".DONUT" 0.0 0.05 DT5 ""))
  (if (= (nth 5 TXT) 1)(command ".DONUT" 0.0 0.05 DT6 ""))
  (command ".LAYER" "S" "3DBRAILLE" "")  ;;3D construction
  (if (= (nth 0 TXT) 1)
    (command ".INSERT" "A:/DOT" DT1 1 "" ""))
  (if (= (nth 1 TXT) 1)
    (command ".INSERT" "A:/DOT" DT2 1 "" ""))
  (if (= (nth 2 TXT) 1)
    (command ".INSERT" "A:/DOT" DT3 1 "" ""))
  (if (= (nth 3 TXT) 1)
    (command ".INSERT" "A:/DOT" DT4 1 "" ""))
  (if (= (nth 4 TXT) 1)
    (command ".INSERT" "A:/DOT" DT5 1 "" ""))
  (if (= (nth 5 TXT) 1)
```

```
        (command ".INSERT" "A:/DOT" DT6 1 "" ""))
    ))
   ((= (nth 0 TXT) nil) nil)
 )
 (if (or
      (and (= (nth 0 TXT) 1)(= (nth 1 TXT) 0)(= (nth 2 TXT) 0)
           (= (nth 3 TXT) 0)(= (nth 4 TXT) 0)(= (nth 5 TXT) 0));;1
      (and (= (nth 0 TXT) 1)(= (nth 1 TXT) 1)(= (nth 2 TXT) 0)
           (= (nth 3 TXT) 0)(= (nth 4 TXT) 0)(= (nth 5 TXT) 0));;2
      (and (= (nth 0 TXT) 1)(= (nth 1 TXT) 0)(= (nth 2 TXT) 0)
           (= (nth 3 TXT) 1)(= (nth 4 TXT) 0)(= (nth 5 TXT) 0));;3
      (and (= (nth 0 TXT) 1)(= (nth 1 TXT) 0)(= (nth 2 TXT) 0)
           (= (nth 3 TXT) 1)(= (nth 4 TXT) 1)(= (nth 5 TXT) 0));;4
      (and (= (nth 0 TXT) 1)(= (nth 1 TXT) 0)(= (nth 2 TXT) 0)
           (= (nth 3 TXT) 0)(= (nth 4 TXT) 1)(= (nth 5 TXT) 0));;5
      (and (= (nth 0 TXT) 1)(= (nth 1 TXT) 1)(= (nth 2 TXT) 0)
           (= (nth 3 TXT) 1)(= (nth 4 TXT) 0)(= (nth 5 TXT) 0));;6
      (and (= (nth 0 TXT) 1)(= (nth 1 TXT) 1)(= (nth 2 TXT) 0)
           (= (nth 3 TXT) 1)(= (nth 4 TXT) 1)(= (nth 5 TXT) 0));;7
      (and (= (nth 0 TXT) 1)(= (nth 1 TXT) 1)(= (nth 2 TXT) 0)
           (= (nth 3 TXT) 0)(= (nth 4 TXT) 1)(= (nth 5 TXT) 0));;8
      (and (= (nth 0 TXT) 0)(= (nth 1 TXT) 1)(= (nth 2 TXT) 0)
           (= (nth 3 TXT) 1)(= (nth 4 TXT) 0)(= (nth 5 TXT) 0));;9
      (and (= (nth 0 TXT) 0)(= (nth 1 TXT) 0)(= (nth 2 TXT) 1)
           (= (nth 3 TXT) 0)(= (nth 4 TXT) 1)(= (nth 5 TXT) 1));;0
      (and (= (nth 0 TXT) 0)(= (nth 1 TXT) 1)(= (nth 2 TXT) 0)
           (= (nth 3 TXT) 0)(= (nth 4 TXT) 1)(= (nth 5 TXT) 1));;$
      (and (= (nth 0 TXT) 0)(= (nth 1 TXT) 0)(= (nth 2 TXT) 1)
           (= (nth 3 TXT) 1)(= (nth 4 TXT) 0)(= (nth 5 TXT) 0));;/
      (and (= (nth 0 TXT) 0)
           (= (nth 1 TXT) 0)
           (= (nth 2 TXT) 1)
           (= (nth 3 TXT) 1)
           (= (nth 4 TXT) 1)
           (= (nth 5 TXT) 1)))
     (setq NUMCHK 1)
     (setq NUMCHK 0))
  (princ)
);end of MAKECELL.LSP

;;construct binary code to make each symbol
(defun BCODE (TXT)
 (if (= NUMCHK 1)              ;; previous character was a number test
  (cond
```

```
((= TXT (chr 47))(setq TXT (list '0'0'1'1'0'0)))          ;;fraction
((= TXT (chr 95))(setq TXT (list '0'0'1'0'0'1)))          ;;hyphen
((= TXT (chr 59))(setq TXT (list '0'1'1'0'0'0)))          ;;semicolon
((= TXT (chr 58))(setq TXT (list '0'1'0'0'1'0)))          ;;colon
((= TXT (chr 33))(setq TXT (list '0'1'1'0'1'0)))          ;;exclamation
((= TXT (chr 34))(setq TXT (list '0'0'1'0'1'1)))          ;;dbl quote
((= TXT (chr 40))(setq TXT (list '0'1'1'0'1'1)))          ;;open paren
((= TXT (chr 41))(setq TXT (list '0'1'1'0'1'1)))          ;;close paren
((= TXT (chr 39))(setq TXT (list '0'0'0'1'0'0)))          ;;apostrophy
((= TXT (chr 32))(setq TXT (list '0'0'0'0'0'0)))       ;;space
((= TXT (chr 44))(setq TXT (list '0'1'0'0'0'0)))       ;;comma
((= TXT (chr 46))(setq TXT (list '0'1'0'0'1'1)))       ;;period
((= TXT (chr 35))(setq TXT (list '0'0'1'1'1'1)))       ;;#
((= TXT (chr 36))(setq TXT (list '0'1'0'0'1'1)))       ;;$
((= TXT (chr 63))(setq TXT (list '0'1'1'0'0'1)))       ;;?
((= TXT (chr 37))(setq TXT (list '0'0'0'1'0'1)))       ;;%
((= TXT (chr 94))(setq TXT (list '0'0'0'0'0'1)))       ;;^
((= TXT (chr 38))(setq TXT (list '0'0'0'0'1'1)))       ;;&
((= TXT (chr 60))(setq TXT (list '0'0'0'0'0'0)))       ;;<
((= TXT (chr 62))(setq TXT (list '0'0'0'0'0'0)))       ;;>
((= TXT (chr 64))(setq TXT (list '0'0'0'0'0'0)))       ;;@
((= TXT (chr 95))(setq TXT (list '0'0'0'0'0'0)))       ;;underscore
((= TXT (chr 96))(setq TXT (list '0'0'0'0'0'0)))       ;;`
((= TXT (chr 92))(setq TXT (list '0'0'0'0'0'0)))       ;;\
((= TXT (chr 91))(setq TXT (list '0'0'0'0'0'0)))       ;;[
((= TXT (chr 93))(setq TXT (list '0'0'0'0'0'0)))       ;;]
((= TXT (chr 123))(setq TXT (list '0'0'0'0'0'0)))      ;;{
((= TXT (chr 124))(setq TXT (list '0'0'0'0'0'0)))      ;;|
((= TXT (chr 125))(setq TXT (list '0'0'0'0'0'0)))      ;;}
((= TXT (chr 126))(setq TXT (list '0'0'0'0'0'0)))      ;;~
((= TXT (chr 42))(setq TXT (list '0'0'1'0'1'0'0'0'1'0'1'0))) ;;*
((= TXT (chr 48))(setq TXT (list '1'0'0'0'0'0))) ;;1
((= TXT (chr 49))(setq TXT (list '1'1'0'0'0'0))) ;;2
((= TXT (chr 50))(setq TXT (list '1'0'0'1'0'0))) ;;3
((= TXT (chr 51))(setq TXT (list '1'0'0'1'1'0)))
((= TXT (chr 52))(setq TXT (list '1'0'0'0'1'0)))
((= TXT (chr 53))(setq TXT (list '1'1'0'1'0'0)))
((= TXT (chr 54))(setq TXT (list '1'1'0'1'1'0)))
((= TXT (chr 55))(setq TXT (list '1'1'0'0'1'0))) ;;8
((= TXT (chr 56))(setq TXT (list '0'1'0'1'0'0))) ;;9
((= TXT (chr 57))(setq TXT (list '0'0'1'0'1'1))) ;;0
((= TXT (chr 65))(setq TXT (list '0'0'0'0'0'1'1'0'0'0'0'0))) ;;A
((= TXT (chr 66))(setq TXT (list '0'0'0'0'0'1'1'1'0'0'0'0))) ;;B
((= TXT (chr 67))(setq TXT (list '0'0'0'0'0'1'1'0'0'1'0'0))) ;;C
```

```
((= TXT (chr 68))(setq TXT (list '0'0'0'0'0'1'1'0'0'1'1'0)))
((= TXT (chr 69))(setq TXT (list '0'0'0'0'0'1'1'0'0'0'1'0)))
((= TXT (chr 70))(setq TXT (list '0'0'0'0'0'1'1'1'0'1'0'0)))
((= TXT (chr 71))(setq TXT (list '0'0'0'0'0'1'1'1'0'1'1'0)))
((= TXT (chr 72))(setq TXT (list '0'0'0'0'0'1'1'1'0'0'1'0)))
((= TXT (chr 73))(setq TXT (list '0'0'0'0'0'1'0'1'0'1'0'0)))
((= TXT (chr 74))(setq TXT (list '0'0'0'0'0'1'0'1'0'1'1'0)))
((= TXT (chr 75))(setq TXT (list '0'0'0'0'0'1'1'0'1'0'0'0)))
((= TXT (chr 76))(setq TXT (list '0'0'0'0'0'1'1'1'1'0'0'0)))
((= TXT (chr 77))(setq TXT (list '0'0'0'0'0'1'1'0'1'1'0'0)))
((= TXT (chr 78))(setq TXT (list '0'0'0'0'0'1'1'0'1'1'1'0)))
((= TXT (chr 79))(setq TXT (list '0'0'0'0'0'1'1'0'1'0'1'0)))
((= TXT (chr 80))(setq TXT (list '0'0'0'0'0'1'1'1'1'1'0'0)))
((= TXT (chr 81))(setq TXT (list '0'0'0'0'0'1'1'1'1'1'1'0)))
((= TXT (chr 82))(setq TXT (list '0'0'0'0'0'1'1'1'1'0'1'0)))
((= TXT (chr 83))(setq TXT (list '0'0'0'0'0'1'0'1'1'1'0'0)))
((= TXT (chr 84))(setq TXT (list '0'0'0'0'0'1'0'1'1'1'1'0)))
((= TXT (chr 85))(setq TXT (list '0'0'0'0'0'1'1'0'1'0'0'1)))
((= TXT (chr 86))(setq TXT (list '0'0'0'0'0'1'1'1'1'0'0'1)))
((= TXT (chr 87))(setq TXT (list '0'0'0'0'0'1'0'1'0'1'1'1))) ;;W
((= TXT (chr 88))(setq TXT (list '0'0'0'0'0'1'1'0'1'1'0'1))) ;;X
((= TXT (chr 89))(setq TXT (list '0'0'0'0'0'1'1'0'1'1'1'1))) ;;Y
((= TXT (chr 90))(setq TXT (list '0'0'0'0'0'1'1'0'1'0'1'1))) ;;Z
((= TXT (chr 97))(setq TXT (list '1'0'0'0'0'0)))          ;;a
((= TXT (chr 98))(setq TXT (list '1'1'0'0'0'0)))          ;;b
((= TXT (chr 99))(setq TXT (list '1'0'0'1'0'0)))          ;;c
((= TXT (chr 100))(setq TXT (list '1'0'0'1'1'0)))
((= TXT (chr 101))(setq TXT (list '1'0'0'0'1'0)))
((= TXT (chr 102))(setq TXT (list '1'1'0'1'0'0)))
((= TXT (chr 103))(setq TXT (list '1'1'0'1'1'0)))
((= TXT (chr 104))(setq TXT (list '1'1'0'0'1'0)))
((= TXT (chr 105))(setq TXT (list '0'1'0'1'0'0)))
((= TXT (chr 106))(setq TXT (list '0'1'0'1'1'0)))
((= TXT (chr 107))(setq TXT (list '1'0'1'0'0'0)))
((= TXT (chr 108))(setq TXT (list '1'1'1'0'0'0)))
((= TXT (chr 109))(setq TXT (list '1'0'1'1'0'0)))
((= TXT (chr 110))(setq TXT (list '1'0'1'1'1'0)))
((= TXT (chr 111))(setq TXT (list '1'0'1'0'1'0)))
((= TXT (chr 112))(setq TXT (list '1'1'1'1'0'0)))
((= TXT (chr 113))(setq TXT (list '1'1'1'1'1'0)))
((= TXT (chr 114))(setq TXT (list '1'1'1'0'1'0)))
((= TXT (chr 115))(setq TXT (list '0'1'1'1'0'0)))
((= TXT (chr 116))(setq TXT (list '0'1'1'1'1'0)))
((= TXT (chr 117))(setq TXT (list '1'0'1'0'0'1)))
```

```
 ((= TXT (chr 118))(setq TXT (list '1'1'1'0'0'1)))
 ((= TXT (chr 119))(setq TXT (list '0'1'0'1'1'1)))
 ((= TXT (chr 120))(setq TXT (list '1'0'1'1'0'1)))      ;;x
 ((= TXT (chr 121))(setq TXT (list '1'0'1'1'1'1)))      ;;y
 ((= TXT (chr 122))(setq TXT (list '1'0'1'0'1'1)))      ;;z
 (t nil));end of cond
                        ;; previous sharacter not a number
(cond
 ((= TXT (chr 47))(setq TXT (list '0'0'1'1'0'0)))       ;;fraction
 ((= TXT (chr 95))(setq TXT (list '0'0'1'0'0'1)))       ;;hyphen
 ((= TXT (chr 59))(setq TXT (list '0'1'1'0'0'0)))       ;;semicolon
 ((= TXT (chr 58))(setq TXT (list '0'1'0'0'1'0)))       ;;colon
 ((= TXT (chr 33))(setq TXT (list '0'1'1'0'1'0)))       ;;exclamation
 ((= TXT (chr 34))(setq TXT (list '0'0'1'0'1'1)))       ;;dbl quote
 ((= TXT (chr 40))(setq TXT (list '0'1'1'0'1'1)))       ;;open paren
 ((= TXT (chr 41))(setq TXT (list '0'1'1'0'1'1)))       ;;close paren
 ((= TXT (chr 39))(setq TXT (list '0'0'0'1'0'0)))       ;;apostrophy
 ((= TXT (chr 32))(setq TXT (list '0'0'0'0'0'0)))     ;;space
 ((= TXT (chr 44))(setq TXT (list '0'1'0'0'0'0)))     ;;comma
 ((= TXT (chr 46))(setq TXT (list '0'1'0'0'1'1)))     ;;period
 ((= TXT (chr 35))(setq TXT (list '0'0'1'1'1'1)))     ;;#
 ((= TXT (chr 36))(setq TXT (list '0'1'0'0'1'1)))     ;;$
 ((= TXT (chr 63))(setq TXT (list '0'1'1'0'0'1)))     ;;?
 ((= TXT (chr 37))(setq TXT (list '0'0'0'1'0'1)))     ;;%
 ((= TXT (chr 94))(setq TXT (list '0'0'0'0'0'1)))     ;;^
 ((= TXT (chr 38))(setq TXT (list '0'0'0'0'1'1)))     ;;&
 ((= TXT (chr 60))(setq TXT (list '0'0'0'0'0'0)))     ;;<
 ((= TXT (chr 62))(setq TXT (list '0'0'0'0'0'0)))     ;;>
 ((= TXT (chr 64))(setq TXT (list '0'0'0'0'0'0)))     ;;@
 ((= TXT (chr 95))(setq TXT (list '0'0'0'0'0'0)))     ;;underscore
 ((= TXT (chr 96))(setq TXT (list '0'0'0'0'0'0)))     ;;`
 ((= TXT (chr 92))(setq TXT (list '0'0'0'0'0'0)))     ;;\
 ((= TXT (chr 91))(setq TXT (list '0'0'0'0'0'0)))     ;;[
 ((= TXT (chr 93))(setq TXT (list '0'0'0'0'0'0)))     ;;]
 ((= TXT (chr 123))(setq TXT (list '0'0'0'0'0'0)))     ;;{
 ((= TXT (chr 124))(setq TXT (list '0'0'0'0'0'0)))     ;;|
 ((= TXT (chr 125))(setq TXT (list '0'0'0'0'0'0)))     ;;}
 ((= TXT (chr 126))(setq TXT (list '0'0'0'0'0'0)))     ;;~
 ((= TXT (chr 42))(setq TXT (list '0'0'1'0'1'0'0'0'1'0'1'0))) ;;*
 ((= TXT (chr 48))(setq TXT (list '0'0'1'1'1'1'1'0'0'0'0'0))) ;;1
 ((= TXT (chr 49))(setq TXT (list '0'0'1'1'1'1'1'1'0'0'0'0))) ;;2
 ((= TXT (chr 50))(setq TXT (list '0'0'1'1'1'1'1'0'0'1'0'0))) ;;3
 ((= TXT (chr 51))(setq TXT (list '0'0'1'1'1'1'1'0'0'1'1'0)))
 ((= TXT (chr 52))(setq TXT (list '0'0'1'1'1'1'1'0'0'0'1'0)))
```

```
((= TXT (chr 53))(setq TXT (list '0'0'1'1'1'1'1'1'0'1'0'0)))
((= TXT (chr 54))(setq TXT (list '0'0'1'1'1'1'1'1'0'1'1'0)))
((= TXT (chr 55))(setq TXT (list '0'0'1'1'1'1'1'1'0'0'1'0))) ;;8
((= TXT (chr 56))(setq TXT (list '0'0'1'1'1'1'0'1'0'1'0'0))) ;;9
((= TXT (chr 57))(setq TXT (list '0'0'1'1'1'1'0'0'1'0'1'1))) ;;0
((= TXT (chr 65))(setq TXT (list '0'0'0'0'0'1'1'0'0'0'0'0))) ;;A
((= TXT (chr 66))(setq TXT (list '0'0'0'0'0'1'1'1'0'0'0'0))) ;;B
((= TXT (chr 67))(setq TXT (list '0'0'0'0'0'1'1'0'0'1'0'0))) ;;C
((= TXT (chr 68))(setq TXT (list '0'0'0'0'0'1'1'0'0'1'1'0)))
((= TXT (chr 69))(setq TXT (list '0'0'0'0'0'1'1'0'0'0'1'0)))
((= TXT (chr 70))(setq TXT (list '0'0'0'0'0'1'1'1'0'1'0'0)))
((= TXT (chr 71))(setq TXT (list '0'0'0'0'0'1'1'1'0'1'1'0)))
((= TXT (chr 72))(setq TXT (list '0'0'0'0'0'1'1'1'0'0'1'0)))
((= TXT (chr 73))(setq TXT (list '0'0'0'0'0'1'0'1'0'1'0'0)))
((= TXT (chr 74))(setq TXT (list '0'0'0'0'0'1'0'1'0'1'1'0)))
((= TXT (chr 75))(setq TXT (list '0'0'0'0'0'1'1'0'1'0'0'0)))
((= TXT (chr 76))(setq TXT (list '0'0'0'0'0'1'1'1'1'0'0'0)))
((= TXT (chr 77))(setq TXT (list '0'0'0'0'0'1'1'0'1'1'0'0)))
((= TXT (chr 78))(setq TXT (list '0'0'0'0'0'1'1'0'1'1'1'0)))
((= TXT (chr 79))(setq TXT (list '0'0'0'0'0'1'1'0'1'0'1'0)))
((= TXT (chr 80))(setq TXT (list '0'0'0'0'0'1'1'1'1'1'0'0)))
((= TXT (chr 81))(setq TXT (list '0'0'0'0'0'1'1'1'1'1'1'0)))
((= TXT (chr 82))(setq TXT (list '0'0'0'0'0'1'1'1'1'0'1'0)))
((= TXT (chr 83))(setq TXT (list '0'0'0'0'0'1'0'1'1'1'0'0)))
((= TXT (chr 84))(setq TXT (list '0'0'0'0'0'1'0'1'1'1'1'0)))
((= TXT (chr 85))(setq TXT (list '0'0'0'0'0'1'1'0'1'0'0'1)))
((= TXT (chr 86))(setq TXT (list '0'0'0'0'0'1'1'1'1'0'0'1)))
((= TXT (chr 87))(setq TXT (list '0'0'0'0'0'1'0'1'0'1'1'1))) ;;W
((= TXT (chr 88))(setq TXT (list '0'0'0'0'0'1'1'0'1'1'0'1))) ;;X
((= TXT (chr 89))(setq TXT (list '0'0'0'0'0'1'1'0'1'1'1'1))) ;;Y
((= TXT (chr 90))(setq TXT (list '0'0'0'0'0'1'1'0'1'0'1'1))) ;;Z
((= TXT (chr 97))(setq TXT (list '1'0'0'0'0'0)))          ;;a
((= TXT (chr 98))(setq TXT (list '1'1'0'0'0'0)))          ;;b
((= TXT (chr 99))(setq TXT (list '1'0'0'1'0'0)))          ;;c
((= TXT (chr 100))(setq TXT (list '1'0'0'1'1'0)))
((= TXT (chr 101))(setq TXT (list '1'0'0'0'1'0)))
((= TXT (chr 102))(setq TXT (list '1'1'0'1'0'0)))
((= TXT (chr 103))(setq TXT (list '1'1'0'1'1'0)))
((= TXT (chr 104))(setq TXT (list '1'1'0'0'1'0)))
((= TXT (chr 105))(setq TXT (list '0'1'0'1'0'0)))
((= TXT (chr 106))(setq TXT (list '0'1'0'1'1'0)))
((= TXT (chr 107))(setq TXT (list '1'0'1'0'0'0)))
((= TXT (chr 108))(setq TXT (list '1'1'1'0'0'0)))
((= TXT (chr 109))(setq TXT (list '1'0'1'1'0'0)))
```

```
    ((= TXT (chr 110))(setq TXT (list '1'0'1'1'1'0)))
    ((= TXT (chr 111))(setq TXT (list '1'0'1'0'1'0)))
    ((= TXT (chr 112))(setq TXT (list '1'1'1'1'0'0)))
    ((= TXT (chr 113))(setq TXT (list '1'1'1'1'1'0)))
    ((= TXT (chr 114))(setq TXT (list '1'1'1'0'1'0)))
    ((= TXT (chr 115))(setq TXT (list '0'1'1'1'0'0)))
    ((= TXT (chr 116))(setq TXT (list '0'1'1'1'1'0)))
    ((= TXT (chr 117))(setq TXT (list '1'0'1'0'0'1)))
    ((= TXT (chr 118))(setq TXT (list '1'1'1'0'0'1)))
    ((= TXT (chr 119))(setq TXT (list '0'1'0'1'1'1)))
    ((= TXT (chr 120))(setq TXT (list '1'0'1'1'0'1)))          ;;x
    ((= TXT (chr 121))(setq TXT (list '1'0'1'1'1'1)))          ;;y
    ((= TXT (chr 122))(setq TXT (list '1'0'1'0'1'1)))          ;;z
    (t nil));end of cond
  );; end of if...
);end of BCODE.LSP

(defun C:BRAILLE (/ BM SM OM SP CHRCNT TXT)
  (setvar "CMDECHO" 0)
  (command ".UNDO" "M")
  (setq BM (getvar "BLIPMODE")          ;;store system variables
        SM (getvar "SNAPMODE")
        OM (getvar "ORTHOMODE")
        LAY (getvar "CLAYER")           ;;current layer
        UCSCHK 0)                       ;;UCS check marker
  (setvar "BLIPMODE" 0)
  (setvar "SNAPMODE" 0)
  (setvar "ORTHOMODE" 0)
  (if (equal (tblsearch "LAYER" "2DBRAILLE") nil);;create 2D layer
    (command ".LAYER" "N" "2DBRAILLE" "C" "BLUE" "2DBRAILLE" ""))
  (if (equal (tblsearch "LAYER" "3DBRAILLE") nil);;create 3D layer
    (command ".LAYER" "N" "3DBRAILLE" "C" "RED" "3DBRAILLE" ""))
  (initget 1 "E e S s")
  (setq CHRSTR (getkword "\nEnter or Select text to convert:<E or S>"))
  (cond ((= CHRSTR "E")                 ;;entered text
         (setq CHRSTR (getstring T "\nEnter text to convert: ")))
        ((= CHRSTR "e")                 ;;entered text
         (setq CHRSTR (getstring T "\nEnter text to convert: ")))
        ((= CHRSTR "S")                 ;;selected text
         (progn
          (setq CHRSTR (entget (car (entsel "\nPick text to convert:"))))
          (if (= (cdr (assoc 0 CHRSTR)) "TEXT");;MTEXT test
            (progn                      ;;user coordinate modification
              (command "UCS" "OB" (cdr (assoc -1 CHRSTR)))
```

```
          (setq UCSCHK 1)
          (setq CHRSTR (cdr (assoc 1 CHRSTR))))
         (progn
          (prompt "\nSorry, MTEXT is not supported.")
          (setq CHRSTR (getstring T "\nEnter text to convert:"))))))
    ((= CHRSTR "s")                ;;selected text to convert
      (progn
       (setq CHRSTR (entget (car (entsel "\nPick text to convert:"))))
       (if (= (cdr (assoc 0 CHRSTR)) "TEXT");;MTEXT test
        (progn                     ;;user coordinate modification
         (command "UCS" "OB" (cdr (assoc -1 CHRSTR)))
         (setq UCSCHK 1)
         (setq CHRSTR (cdr (assoc 1 CHRSTR))))
        (progn
         (prompt "\nSorry, MTEXT is not supported.")
         (setq CHRSTR (getstring T "\nEnter text to convert:"))))))
);;end of (cond....
(initget 1)
(setq SP (getpoint "\nStarting point of Braille Symbol set: "))
(setq CHRCNT 1)                    ;;initiate counter for each character
(setq NUMCHK 0)                    ;;previous number marker
(repeat (strlen CHRSTR)
     (setq TXT (substr CHRSTR CHRCNT 1)) ;;individual character
     (MAKECELL TXT)                ;;construct individual cell
     (setq SP (polar SP 0.0 0.35)) ;;adjust cell location
     (setq CHRCNT (1+ CHRCNT)))    ;;end of repeat
(if (= UCSCHK 1)(command "UCS" "P")) ;;restore previous UCS for selected text
(setvar "BLIPMODE" BM)             ;;restore system variables
(setvar "SNAPMODE" SM)
(setvar "ORTHOMODE" OM)
(setvar "CLAYER" LAY)
(princ)
) ;;end of BRAILLE.LSP
```

# APPENDIX C

## ACAD.MNU CUSTOMIZATION

Release 12 Acad.mnu Customization for Braille.lsp

```
[--]
[->Text]
  [Dynamic]^C^C_dtext
  [Braille Text]^C^C(if (not C:BRAILLE)(load "A:/BRAILLE"))
     BRAILLE
  [Import Text]^C^Casctext
  [Set Style...]$I=icon_fonts1 $I=*
  [--]
  [->Attributes]
    [Define...]^C^Cddattdef
    [Edit...]^C^C_ddatte
    [<-<-Extract...]^C^Cddattext
[--]
```

Release 13 Acad.mnu Customization for Braille.lsp

```
[--]
[->&Text]
 [&Text]^C^C_mtext
 [&Dynamic Text]^C^C_dtext
 [&Single-Line Text]^C^C_text
 [<-&Braille Text]^C^C(if (not C:BRAILLE)
     (load "A:/BRAILLE")) BRAILLE
[->&Dimensioning]
```

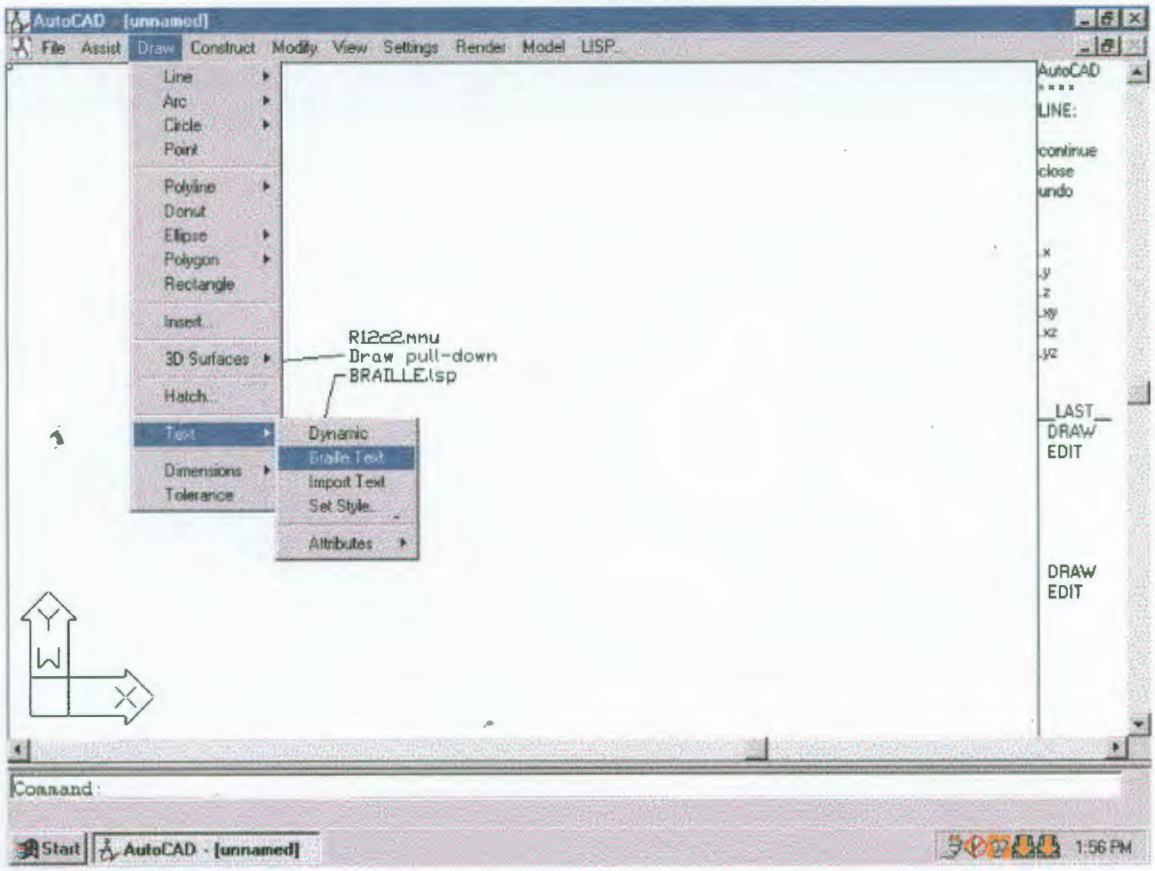
Release 14 Acad.mnu Customization for Braille.lsp

```
[--]
[->Te&xt]
[&Multiline Text...]^C^C_mtext
[<-&Single Line Text]^C^C_dtext
[<-&Braille Text]^C^C(if(not C:BRAILLE)(load "A:/BRAILLE"))
    BRAILLE
[--]
```

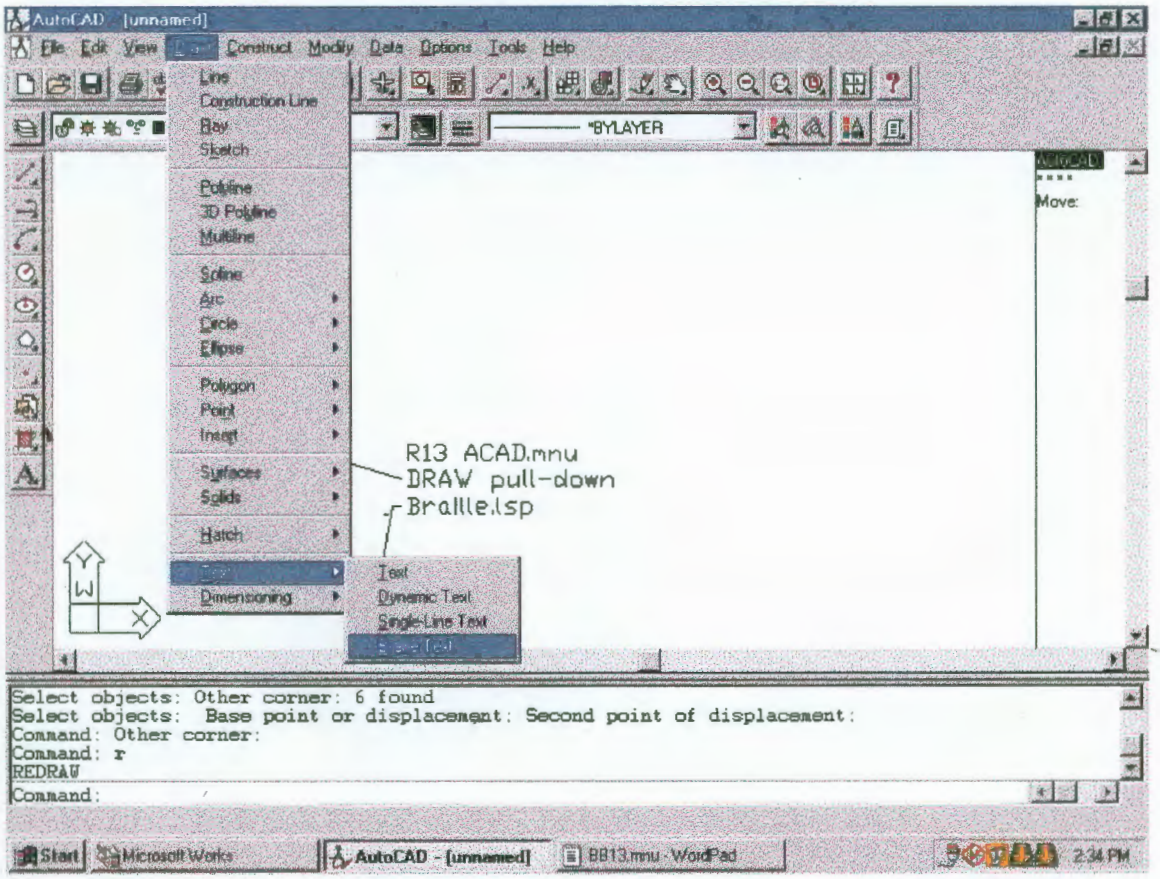
# APPENDIX D

## AUTOCAD CUSTOMIZED SCREENS

# AutoCAD Release 12 Customized Screen



# AutoCAD Release 13 Customized Screen



# AutoCAD Release 14 Customized Screen