AllenCAD

AllenCAD Pro 7

for AutoCAD® Users and AutoLISP®/ADS® Application Developers

Document Revision - May, 2004

For more information on purchasing AllenCAD products and services, locate a Selling Agent or join the Developer Network:

Inside North America Contact:

GiveMePower Corporation
U.S./Canada Toll Free: (800) 258-6360
Tel: (603) 216-6344
Fax: (603) 216-6345
E-Mail: info@adsi-usa.com
Visit: http://www.adsi-usa.com/

Disclaimer

ADSI, GiveMePower Corp and Felix Computer Aided Technologies GmbH make no representations or warranties regarding the contents of this document, and specifically disclaim any implied warranties of merchantability of fitness for any particular purpose. Furthermore, ADSI, GiveMePower Corp and Felix Computer Aided Technologies GmbH reserve the right to revise this document and to make changes from time to time in the contents without an obligation by ADSI, GiveMePower Corp and Felix Computer Aided Technologies GmbH to notify any person of such revision changes.

The Software Product described herein, as with all computer-aided design software and other technical software, is a tool intended to be used by trained professionals only. It is not a substitute for the professional judgment of trained professionals. The Software Product is intended to assist with product design and is not a substitute for independent testing of product stress, safety and utility. Due to the large variety of potential applications for the Software Product, the Software Product has not been tested in all situations under which it may be used. ADSI, GiveMePower Corp and Felix Computer Aided Technologies GmbH shall not be liable in any manner whatsoever for results obtained through the Use of the Software Product. You agree that you are solely responsible for determining whether the Software Product is appropriate in your specific situation in order to achieve your intended results. You are also responsible for establishing the adequacy of independent procedures for testing the reliability and accuracy of any items designed by using the Software Product.

Trademarks

PowerENGINE™, Power Developer Network™, FCAD, FelixCAD®, and FLX™ are either registered trademarks or trademarks of GiveMePower Corp. or Felix Computer Aided Technologies GmbH. MS, Windows®, and WMF are either registered trademarks or trademarks of Microsoft Corporation. AutoCAD®, AutoCAD Development System®, ADS®, AutoCAD LT®, AutoLISP®, DWG®, and DXF® are either registered trademarks or trademarks of Autodesk, Inc. Adobe Acrobat and PostScript are trademarks of Adobe System, Inc. ACIS ® are either registered trademarks or trademarks Spatial Technology, Inc., Three-Space Ltd., and Applied Geometry Corp. All other product names or trademarks mentioned herein are trademarks of their respective owners.

Table of Contents

Introduction

AllenCAD provides a high degree of compatibility with the AutoCAD® computer-aided design software from Autodesk, Inc. AllenCAD is compatible with drawings, commands customization, and programming interfaces from AutoCAD Release 14 and earlier.

As an AutoCAD user, you will feel familiar with AllenCAD right away. Try using AllenCAD for an hour or so, and you will recognize that AllenCAD does work in a way that is similar to AutoCAD.

AllenCAD provides advanced functionality for computer-aided design (CAD) that can be compared to market-leading software — but at a fraction of the price.

AllenCAD is the best choice for integrating CAD workstations in a network, even with mixed AutoCAD-AllenCAD configurations. You can open DWG® drawings and save your work as DWG® files, so that everyone in your company works with the same file format. AllenCAD lets you reference DWG files in your drawings at any time by using the XRef (externally referenced drawings) feature.

AllenCAD is a viable alternative if you have thought about using a less-expensive CAD program, such as AutoCAD LT®, but don't want to miss out on the advantages of customization and programming. AllenCAD has an open architecture, just like many other software programs.

AutoCAD Compatibility

AllenCAD is compatible with AutoCAD in these crucial areas:

- Drawing file format (DWG and DXF®).
- Commands and input modes.
- Customization.
- LISP programming.
- C/C++ programming interface.
- Application-defined dialog boxes.

This manual describes the similarities and differences between AllenCAD and AutoCAD. It is addressed to CAD users and programmers who are already familiar with AutoCAD and want to start using AllenCAD productively in their work — right away!

AutoCAD Migration Tools

For those areas where AllenCAD is not 100% compatible with AutoCAD, our AutoCAD migration tools help ease the conversion process. These include:

Menu Converter

MnuConv.Exe is a utility program that converts AutoCAD menu files (*.MNU) into AllenCAD menu files. It automatically converts the pulldown and icon menus of the DOS and Windows versions of Release 12 through 2002. If necessary, a file is created that contains all expressions that could not be converted. The tool is integrated into the Dialog and Menu Editor. AutoCAD menus can be converted under the **File** menu and the option **AutoCAD Import**.

An AutoCAD *.MNS menu may be converted easily also, as it is generated from a *.MNU file of the same name. Many cases require only renaming the file from FILENAME.MNS to FILENAME.MNU then using the Menu Converter.

Dialog Box Converter

Dcl2Dlg.Exe allows you to convert AutoCAD *.dcl* (dialog control language) files to AllenCAD *.dlg* files. The tool is included in the Dialog and Menu Editor. It converts dialogs directly before editing.

Notes to Application Developers

To convert AutoCAD .dcl files, the source directory has to include the *acad.dcl* and the *base.dcl*. These files are containing predefined buttons, etc. which are required for converting the dialogs into the AllenCAD .dlg Format.

Dialog and Menu Editor

DME is a visual workbench for creating and modifying dialog boxes. You immediately see the result; no DCL programming skills are necessary to create dialog boxes. DME can also be used for application prototyping. You access DME within AllenCAD by selecting **File > Resource Manager > Run Dialog and Menu Editor** from the menu bar.

Include File ads2fdt.h

AllenCAD provides the *ads2fdt.h* include file, which redefines **ads**_ function names to their **fdt**_ equivalents. You must recompile the ADS® source code using the *ads2fdt.h* header file and link the FDT libraries.

For more information, read the online documentation in file **FDTAPI5.PDF**. Also, have a look at the *ads2fdt.h* header file and libraries that are provided with AllenCAD.

Tools Folder

The **tools** subdirectory contains sample files that may be useful to OEM developers. **Tools\dlgtutor** contains examples for creating AllenCAD dialog boxes. **Tools\dwgcnvrt** contains an FLISP program example to convert drawing files. **Tools\protlisp** contains the utility to protect FLISP files. **Tools\shapes** contains font source files (SHP).

In this manual, chapters 2 through 6 thoroughly discuss customization and programming compatibility issues for AutoCAD and AllenCAD.

Compatible User Interface

AutoCAD Compatible User Interface

Since version 6 AllenCAD ships with a Multiple Document Interface (MDI), a user interface used by most Windows applications. This user interface looks similar to that of AutoCAD Release 13, 14, 2000, and 2002.

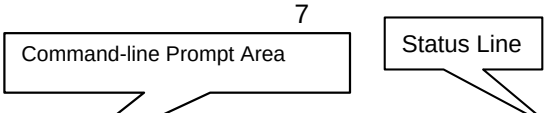
AutoCAD Compatible Commands and Aliases

In the AllenCAD **\cfg** directory is an AutoCAD-compatible alias command file, *facad15e.key* (Note: AutoCAD's equivalent of this aliases file is called *acad.pgp*).

Load this file with the **Config** command. The file contains the redefinitions (where AllenCAD commands are mapped to AutoCAD names) and abbreviations of commands similar to that found in AutoCAD.

AutoCAD Compatible Menus

To make the pulldown menus look like AutoCAD's, load the *facd15e.mnu* menu file (found in the AllenCAD **\support** folder) with the **Menu** command.



AutoCAD Compatible Toolbars

To make the toolbars look like AutoCAD's, use the **PALETTE** command to display the **Open Menu Palette** dialog box. About ten AutoCAD compatible *palettes* (toolbars) are found in the file selection list and are identified by the **ac**_ prefix, for example *ac_draw.mnp*, *ac_edit.mnp*, *ac_view.mnp*, etc. To open a toolbar, select a palette file, then click the Open button.

For more information about the AllenCAD user interface and the desktop layout, see the *AllenCAD User Guide*, Chapter 1 "Getting Started."

Advanced Features

AllenCAD distinguishes itself from AutoCAD Release 2002 (and earlier) by these advanced features:

- Easy modification of the program's *desktop* (user interface).
- Context menus and dialog box support in many commands.
- Easy to organize *part* (block) libraries.
- Drawing Navigator manages non-graphical objects in drawings, such as layers, linetypes, text styles, user coordinate systems, and named views.
- Visual tools to create and modify menus, palettes, dialog boxes, and icon menus.
- Dialog and Menu Editor, Image Library Tool, drag & drop command buttons from palettes onto the desktop toolbars.
- LISP interpreter for manipulating the drawing database.
- C/C++ API with advanced features for event-driven programming functions via hook, callback, and notification mechanisms.
- Delphi Programming Interface

These features are discussed in greater detail below.

Context Menus and Dialog Box Support

Right from the start, AllenCAD has been designed for Windows integration. Wherever possible, AllenCAD displays a dialog box to allow you to make your choices.

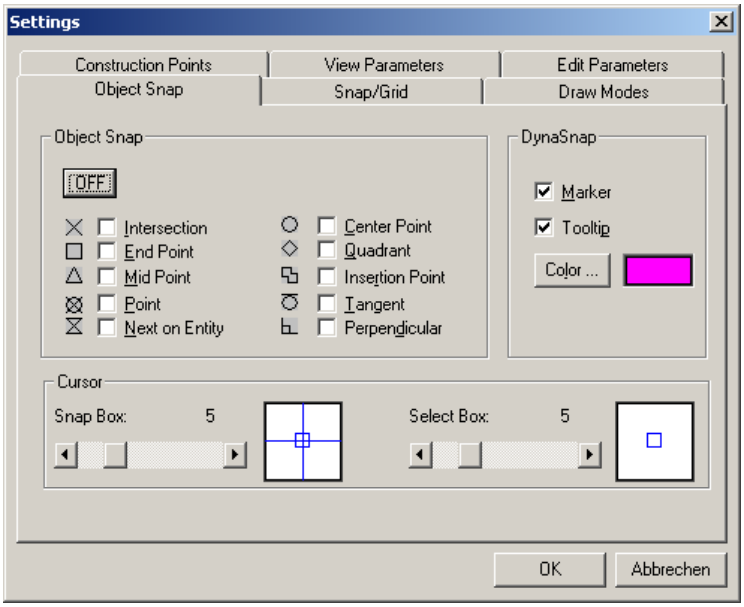


Figure i.4: Many AllenCAD commands display a dialog box. This dialog box is displayed by the **Settings** command.

When a precise specification (like coordinate input) is required, AllenCAD displays the command at the command prompt. In this case, in addition to the Command Prompt area information, AllenCAD displays a context-sensitive options toolbar.

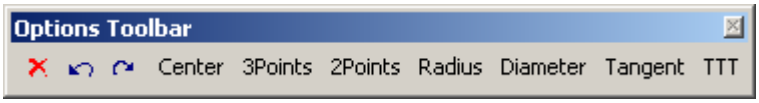


Figure i.5: Some AllenCAD commands display an Option Menu (top) or Option Toolbar, depending which interface you prefer. These options are displayed by the **Circle** command.

At any time when AllenCAD is running, you can change the elements of the user interface (called the *desktop*). All items of the desktop (except pull-down menus) can be turned on and off at run-time.

Drawing Navigator

The Drawing Navigator provides fast access to important non-graphical information of all the drawings that are open. You can view and check the names and status of layers, linetypes, dimension styles, or user coordinate systems defined in the drawings. You can use the Drawing Navigator to organize and manipulate these tables in your drawing.

You access the Drawing Navigator with the **Xplorer** command.

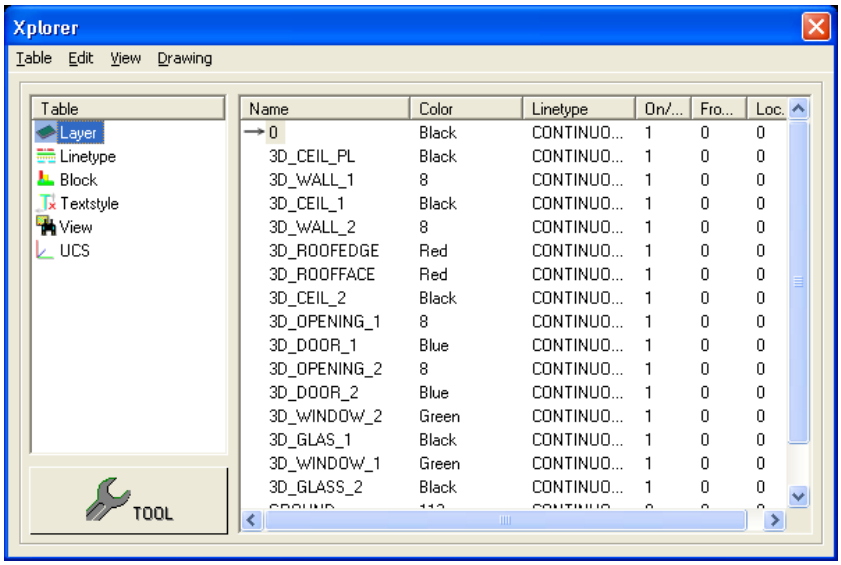


Figure i.6: The Drawing Navigator lists all data known about layers, linetypes, dimension styles, and user coordinate systems for all open drawings.

Part Library Manager

The Part Library Manager allows you to easily manage and organize parts, blocks, and symbols, as well as project drawing files and your company's template drawings.

A drawing library is set up quickly with the Part Library Manager with visual control. This tool collects all or selected files that are located in a folder (or subdirectory). Descriptive information on the parts or drawings is stored within the drawing files.

You access the Part Library Manager with the **Library** command.

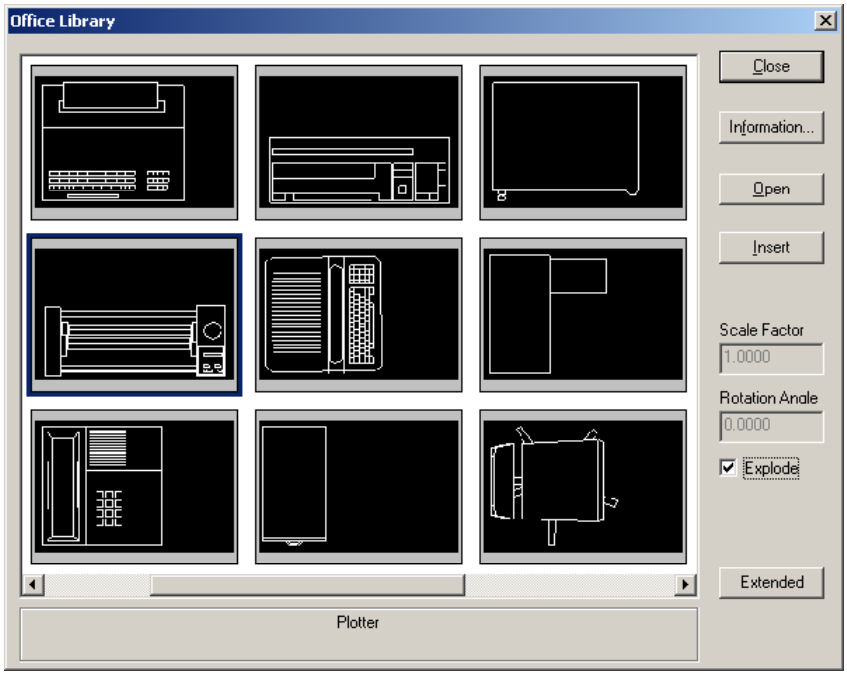


Figure i.7: The Parts Library Manager organize parts, blocks, and symbols, as well as project and template drawings.

Additional User Interface Tools

AllenCAD is delivered with a number of tools and features that let you efficiently customize the user interface without wasting time in studying the syntax of support files:

- The Dialog and Menu Editor (DME) allows you to design and lay out your menus, palettes, and dialog boxes in an visual environment; knowledge of programming is not required. You access the DME via the **DlgEdit** command. See Section 6 for more details.
- The Image Library Tool organizes images (raster type: i.e. *.BMP files, WMF files, and SLD slide files) in libraries. The images can be used in icon menus, dialog boxes, palettes, and toolbars. You access the Image Library Tool via the **ImageLib** command.

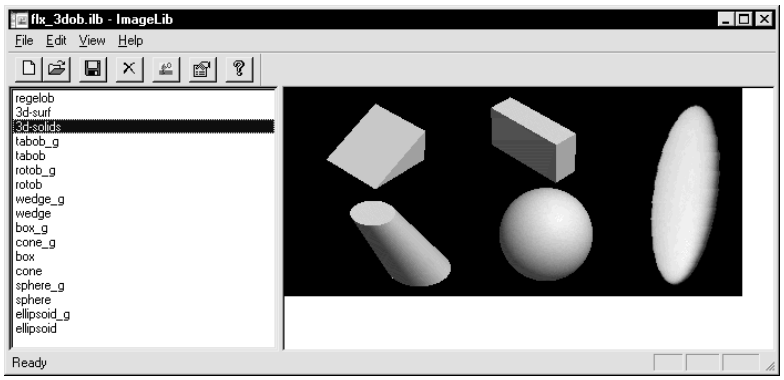


Figure i.8: The Image Library Tool displays raster images stored in a library file.

1. Drawing Database Compatibility

AllenCAD provides full support for reading and writing drawing files produced by AutoCAD:

- **Read DWG files** from Release 2.5 up to Release 16 (AutoCAD 2002), and now with AllenCAD Pro 6 up to 2004/2005, as well as all versions of AutoCAD LT.
- **Write DWG files** in formats that are read by AutoCAD Release 12 up to Release 16 (AutoCAD 2002), and now with AllenCAD Pro 6 up to 2004/2005, as well as by all versions of AutoCAD LT.

In addition, AllenCAD allows you to import and export files in DXF format.

The proprietary drawing file format of AllenCAD is *.flx*. From the beginning, the *.flx* file format was designed to be extremely similar to the *.dwg* format. This ensures that there is nearly no loss of data when exchanging drawings between the two CAD systems.

In drawing database compatibility, AllenCAD provides these features:

- **DWG Roundtrip.** If you open an AutoCAD drawing, you now have the possibility to "Roundtrip" all unsupported AutoCAD entities. That means, that while being edited in AllenCAD, unsupported AutoCAD entities remain unchanged and invisible. If the drawing is saved in AutoCAD drawing format again, all entities including these unsupported AutoCAD entities will be written back to the file.

  "Roundtripped" AutoCAD Entities include:

  ARCALIGNEDTEXT
  BODY
  IMAGE
  MLINE
  PROXY
  RAY
  REGION
  RTEXT
  SOLID3D
  WIPEOUT
  XLINE
  XRECORD-OBJECTS

  "Roundtripped" AutoCAD Table Information include:

  DICTIONARYWDFLTOBJTYPE
  DICTVAR
  IMAGEDEF
  MLINESTYLE
  OBJECTPOINTER
  PLACEHOLDEROBJECT
  VBAPROJECTOBJECT
  WIPEOUTVAROBJECT

- Blocks and WBlocks, including those with attributes, are handled just like in AutoCAD. You can insert *.flx*, *.dwg*, and *.dxf* files into drawings.

- You can attach externally referenced drawings (xrefs). Again, you have the choice of inserting *.flx*, *.dwg*, and *.dxf* files into drawings.
- Supports user coordinate systems (UCS).
- Supports model space and paper space.
- Supports grouping of drawing objects.
- You can use the same font files as in AutoCAD (*.shx*), including BigFonts.
- You can use the same linetype definition files as in AutoCAD (*.lin*).
- You can use the same hatch pattern definition files as in AutoCAD (*.pat*).

AllenCAD provides a high degree of DWG and DXF compatibility. It means you have a secure investment when you want to set up a new computer-aided design workstation, or who want to add CAD workstations in a mixed configuration.

OpenDWG Alliance

AllenCAD employs the DWG read-write technology provided by the OpenDWG Alliance.

Prior to the founding of the OpenDWG Alliance, AllenCAD used the AutoDirect DWG/DXF read-write technology from MarComp. The AutoDirect technology has since become the basis of the OpenDWG Alliance's OpenDWG Toolkit.

Drawing Objects

AllenCAD supports many AutoCAD objects, converts some AutoCAD objects to another format, and does not display a small number of objects. The following sections describe how AllenCAD deals with all objects found in AutoCAD Release 2002.

Supported AutoCAD Objects

The following table lists the geometric entities that are identical in AutoCAD and AllenCAD:

Table 1.1: AutoCAD Objects Supported by AllenCAD

AutoCAD-AllenCAD Object	Comments
3DFACE	3D face entity
ARC	Arc entity
ATTDEF	Attribute definition
ATTRIB	Attribute of a block insertion
BLOCKS & WBLOCKS	Block insertion (block reference, internal & external)
CIRCLE	Circle entity
DIMENSION (*R12 and earlier*)	Dimension (complex entity)
LINE	Line entity
LEADER	Leader line
LWPOLYLINE (*R14 and later*)	Lightweight Polyline (complex entity)
MTEXT (*Release 13*)	Multi-line Text entity
OLE2FRAME *(Release 13)*	OLE (object linking and embedding) object
POLYLINE	Polyline (complex entity)

POINT	Point entity
SOLID	Filled 2D face entity
TEXT	Text entity
TOLERANCE	Tolerance dimensioning
VIEWPORT	Paper space viewport

Note: Dimensioning and hatching is associative in AllenCAD, as in AutoCAD.

Converted AutoCAD Objects

The following table lists DWG geometric entities that are converted by AllenCAD:

Table 1.2: AutoCAD Objects Converted by AllenCAD

AutoCAD Object	AllenCAD Entity	Comments
TRACE (*Introduced to AutoCAD with version 1*)	3DFACE	In AutoCAD's drawing database definition, the TRACE object is identical to the 3DFACE object. When reading an AutoCAD drawing, AllenCAD converts the TRACE object to a 3DFACE entity.
ELLIPSE (*Release 13*)	POLYLINE	In AutoCAD, the ELLIPSE object draws true ellipses and elliptical arcs as a planar object with no thickness. When reading an AutoCAD drawing, AllenCAD converts the ELLIPSE object to POLYLINE entities with an elliptical shape, as in AutoCAD Release 12.
SPLINE (*Release 13*)	POLYLINE	In AutoCAD, the SPLINE object draws a true non-uniform rational B-spline (NURBS). When reading an AutoCAD drawing, AllenCAD converts the SPLINE object to POLYLINE entities with a spline shape.
DIMENSION (*Release 14*)	BLOCK	In AutoCAD, the DIMENSION object is an optimized dimension. When reading an AutoCAD drawing, AllenCAD converts the DIMENSION object to a BLOCK entity containing the dimension elements (lines, arrowheads, and text), as in AutoCAD Release 12.
HATCH (*Release 14*)	BLOCK	In AutoCAD, the HATCH object is an optimized hatch pattern. When reading an AutoCAD drawing, AllenCAD converts the HATCH object to a BLOCK entity containing the hatch element (lines and points), as in AutoCAD Release 12.

Ignored Objects

The following table lists AutoCAD geometric objects that are not displayed by AllenCAD. When objects are not displayed in AllenCAD, they are maintained in the *.dwg* file.

This ensures that the objects remain intact when displayed by AutoCAD.

Table 1.3: AutoCAD Objects Not Supported by AllenCAD

AutoCAD Object	Comments
SHAPE *(AutoCAD version 1)*	Shape entity used in shapes, complex linetypes, and tolerances
3DSOLID *(Release 13)*	ACIS object
BODY *(Release 13)*	ACIS object
MLINE *(Release 13)*	Multiline
ZOMBIE *(Release 13)* PROXY*(Release 14)*	ObjectARX-defined object
RAY *(Release 13)*	Semi-infinite construction line
REGION *(Release 13)*	ACIS object
XLINE *(Release 13)*	Infinite construction line
IMAGE *(Release 14)*	Raster image object

Entity Properties

AutoCAD objects have the following properties: Layer, Color, Linetype, Visibility flag, Linetype scale factor, Space assignment (Model or Paper space), and Thickness (entity extrusion). All of these properties are supported by AllenCAD and they have the same characteristics as in AutoCAD.

For example, AllenCAD applies the same color palette to objects as does AutoCAD. There are 255 RGB colors and two logical colors, BYLAYER and BYBLOCK.

Note for Applications Developers

When an entity extrusion (thickness) is applied *TEXT* and *POINT* entities, the thickness is not displayed by AllenCAD.

Extended Entity Data

Extended entity data (xdata) can be attached to drawing objects (via appropriate programming functions). Xdata is stored in the drawing database as in AutoCAD. The method to access xdata is identical in both software packages.

Xdata are maintained when exchanging drawing files (via DWG and DXF) between the two programs.

Blocks

Like AutoCAD (especially Release 12 and earlier), AllenCAD organizes groups of different types of entities as blocks in the BLOCK section of the drawing database.

This includes:

- User-defined blocks.
- Dimensioning blocks, stored as anonymous blocks.
- Hatch blocks, stored as anonymous blocks.
- Application-defined anonymous blocks.
- External references (Xref).

These blocks and their organization in the AllenCAD database corresponds to the AutoCAD database.

Symbol Tables

Symbol tables store non-graphical objects in the drawing database. AllenCAD manages and organizes symbol tables like AutoCAD does.

These symbol tables are maintained in the TABLES section of the drawing database.

Table 1.4: Symbol Tables Supported by AllenCAD

Symbol Name	Comments
LAYER	Layer names and settings.
LTYPE	Linetype names.
STYLE	Text styles names and settings.
UCS	User coordinate systems.
DIMSTYLE	Dimensioning styles.
VIEW	Named views.
VPORTS	Drawing viewports.
APPID	Registered application identifiers for extended entity data.

Groups

In AutoCAD R13, a section OBJECTS has been added to the drawing database. Similar to tables, this section allows the organization of non-graphical objects in individual sub-sections called "Dictionaries." AutoCAD maintains two dictionaries: *Groups* (references of entities that belong to a group) and *Multiline Styles*. In addition, applications can store and manage non-graphical application data in separate Dictionaries.

As in AutoCAD, you can group entities in AllenCAD with the **Group** command. However, AllenCAD stores groups in a *table* named GROUPS. This is slightly different from AutoCAD, but has the advantage that you can apply the same LISP and C API functions on groups as you do to layers, linetypes, etc.

AllenCAD lets third-party applications store and manage their non-graphical application data in a table named APPDATA. Again, this is slightly different from AutoCAD but has the advantage that you can apply the same LISP and C API functions on groups as with layers and linetypes.

Notes to Application Developers

Since version 4.0, AllenCAD allows you to access symbol table entries via entity names (enames). This means that applications can now use the **entmake** and **entmod** functions to create and modify table entries, respectively.

The new **tblobjname** function returns the entity name of an existing table record.

Attaching extended entity data (xdata) to entries of symbol tables is not yet supported. This has been allowed in AutoCAD since Release13.

Planned Enhancements

The following properties of non-graphical objects were introduced with AutoCAD Release 13 but are not yet supported in AllenCAD's symbol table:

- **Dimstyle Table**: Dimension families, a secondary set of dimension styles for specific groups of dimensioning, such as linear and radial dimensions.
- **Multiline Dictionary**: Multiline styles.

System Variables

Most of AutoCAD's system variables are supported in AllenCAD. This means that you find the most recently settings restored when you exchange drawings between the two systems.

2. Command Compatibility

Most AllenCAD command names and options are comparable to those available in AutoCAD.

Entering Commands

When you want to draw a line, you start the **Line** command in AllenCAD, just as in AutoCAD:

```
> line
From point: [pick]
To point: [pick]
To point: [pick]
To point: c (as in 'Close' the shape, just like AutoCAD)
```

To relocate drawing objects, you apply the **Move** command in AllenCAD, as in AutoCAD:

```
> move
Select objects: l
1 selected.
Select objects: [Enter]
*** 1 selected. ***
Basepoint: [pick]
Second point of displacement: [pick]
```

These examples, like most of AllenCAD's drawing and editing commands, are equivalent in their names in both systems, and are essentially similar in the way they operate. On the whole, commands behave similarly in both systems.

Command Line Area and Text Window

Like AutoCAD, AllenCAD provides a command line area and a text window. Notice that the above examples show that the AllenCAD command prompt is an angle bracket, **>**, rather than AutoCAD's **Command:** prompt.

The **Command History** area allows you to view the command history. Opening the text window is also useful when evaluating and testing LISP expressions.

Figure 2.1: The AllenCAD command line.

Press **F2** to toggel the display of the **Command History** area as a separate text window or at the margins of the AllenCAD desktop.

You can use the MS Windows standard **Ctrl+C** and **Ctrl+V** commands to copy and paste text from and to the command line and text window. As an alternative, the user can click on the up and down scroll arrows located at the right side of the AllenCAD command line. This allows the user to review command history 'in place' a few lines at a time.

Command Options

As an AutoCAD user, you will feel familiar with most of the options that AllenCAD presents for commands.

The primary difference from AutoCAD is the context-sensitive pop-up menu, called the *option menu*, that displays the options available for the command. As an alternative, you can set up a *option bar* instead of the context menu.

See the Figure i.3 in the Introduction for a look at the AllenCAD context menu and option bar.

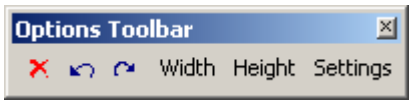
Notes to Application Developers

Programmers can use context-sensitive context menus, called "option menus," in applications. As you probably know from AutoLISP, the **initget** function allows you to define option keywords for user input, which are checked by subsequent **get***xxx* functions.

In AllenCAD's LISP, your routine displays the option menu when you set the bit-coded flag to **256**. Here is an example of its use:

```
(initget (+ 1 256) "Width Height Settings")
(getpoint "Specify point or select option: ")
```

This LISP routine displays the following option menu:



The AllenCAD system variable **InitGetFlag** turns on the option menu as the default when set to 1. This system variable was added in AllenCAD version 4.0; note that neither **InitGetFlag** nor flag 256 work in AutoCAD.

Dialog Boxes vs. Command-line Input

From the beginning, AllenCAD has used dialog boxes for any command where it made sense. For example, AllenCAD's **Layer** command has always displayed a dialog box, whereas AutoCAD didn't switch its **Layer** command to a dialog box until Release 14.

Whereas earlier releases of AutoCAD required multiple steps of command line entries to adjust its settings, AllenCAD has always offered clearly arranged dialog boxes to let you select your preferences at one go. A good example is the **Settings** command.

On the other hand, whenever a command requires precise coordinate and geometric input, AllenCAD presents the command's prompts at the command line, such as the **Line** and **Move** commands shown at the beginning of this chapter.

Command Prefixes and Modifiers

AllenCAD is compatible with many of AutoCAD's command prefixes, such as transparent commands and point filters. The following sections detail the similarities and differences between the two CAD programs.

Invoking Dialog Box Commands

Since Release 9, AutoCAD commands that invoke a dialog box usually (but not always) have the **DD** prefix (short for "dynamic dialog"), such as **DDSTYLE** and **DDLMODES** (the 'Layer' command with dialog box).

With AutoCAD Release 14, several commands have been changed to dialog boxes. In these cases, the dialog box version got the name of the command line variant, whereas the command line variant now requires a hyphen prefix, such as **-Style** and **-Layer**.

"Q" Commands

In contrast, AllenCAD versions 3.1 and 4.0 added command-line variants to existing dialog box-based commands. To continue the tradition of the command naming convention, these commands are identified by the **Q** prefix in the command line version. For example, the **Layer** command displays a dialog box and the **QLayer** command displays prompts at the command line. More example are shown in Table 2.1. We call these "Q commands."

A consistent command naming convention between AllenCAD and AutoCAD cannot be achieved for several commands, especially with introduction of the hyphen-commands in Release 14. The following table shows some of the differences:

Table 2.1: Examples of Command Naming Differences Between AutoCAD and AllenCAD

AutoCAD Command Line	AllenCAD Command Line	AutoCAD Dialog Box	AllenCAD Dialog Box
AttDef	QattDef	DDAttDef	AttDef
Insert	Qinsert	DDInsert	Insert
-Style	Qfont or –Style	Style	Font
-Layer	Qlayer or –Layer	Layer	Layer
-Mtext	Qmtext	MText	MText

Input Methods and Keywords

Coordinates and angles are specified in AllenCAD in the same way as in AutoCAD. Similarly, object snaps and object selection, during a command, correspond to the same methods that you know from AutoCAD. AllenCAD provides the same XYZ point filters for coordinate input.

Overall, AllenCAD accepts the same special keywords you know from AutoCAD. The following table summarizes these methods and keywords:

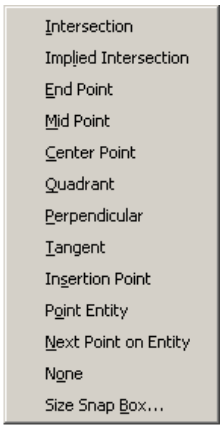
Table 2.2: Input Methods and Keyword Similarities Between AutoCAD and AllenCAD

Input Method or Keyword	Comments
Press **Spacebar** or **Enter**	At the **>** command prompt, the spacebar or **Enter** key recalls the previous command. Within a command, the keystrokes accept the default option. In some cases, the keystrokes terminate the command.
@ symbol	The @ sign references the coordinates of the last point.
From	Establishes a temporary reference point for specifying a subsequent point. The **From** keyword is used in combination with other object snaps and relative coordinates.
@10<<	If the<< expression is followed by **Enter**, the angle of the current cursor position will be used for the input.

Input Method or Keyword	Comments AllenCAD Menu	ACAD Equivalent Menu
Object snap options:		
Int	Intersection	
App	Apparent (implied) intersection	
End	Endpoint	
Mid	Midpoint	
Cen	Center	
Qua	Quadrant	
Per	Perpendicular	
Tan	Tangent	
Ins	Insertion point	
Node	Node (point entity)	
Nea	Nearest point on entity	
None	None	
Object selection options:		
Wp	Window polygon	
Cp	Crossing polygon	
F	Fence	
L	Last	
P	Previous	
R	Remove	
W	Window	
C	Crossing	
Si	Single	
All	All objects in drawing	
Group	Named group in drawing	
Point filters:	References an existing point location and filters out unwanted X, Y, or Z-coordinate values.	
X		
Y	Note that there is a small difference: In AllenCAD, you do not type the point prefix; instead you type **x**, **y**, or **z** and (optionally) the numerical coordinates, as shown in the following examples:	
Z		
x,y	`From point: 5,y` `Y-Coordinate: 6`	
y,z		

x,z	`To point: x,y,z`
	`X-Coordinate: 1.9`
	`Y-Coordinate: 2.8`
	`Z-Coordinate: 3.7`	

x,y,z		



Transparent Commands

As in AutoCAD, many AllenCAD commands can be applied transparently. This means that you can use such commands in the middle of another command using the ‘ (apostrophe) prefix. This is particularly useful in using commands like Move or Copy in conjunction with Zoom, for instance.

Note to Application Developers

AllenCAD applications which are defined by the **defun** function via **C:***command* can be called transparently. To allow this, your function must not contain the function command itself; recursion is not allowed.

Command Aliases, Accelerator Keys, and Scripts

AllenCAD is compatible with many of AutoCAD's shortcuts, such as command aliases and accelerator keys. The following sections detail the similarities and differences between the two CAD programs.

Alias Names for Commands

As in AutoCAD, you can define aliases for command names. In most cases, these are one, two, or three characters, such as **L** for **Line** and **C** for **Copy**.

AutoCAD aliases are limited to executing commands; they cannot handle options. In contrast, AllenCAD allows aliases to execute commands and options, such as **ZE** for **Zoom Extents**.

Another use for command aliasing is to assign familiar AutoCAD command names to corresponding AllenCAD commands. For example, you could assign **POLY3D** to the **3DPOLY** command.

The **Config** command displays **Configuration** dialog box; the **Alias Commands** tab lets you create, modify, and delete aliases for command names.

There is a difference in where and how AutoCAD and AllenCAD store the alias names of commands. AutoCAD users will be familiar with the ACAD.PGP file which contains aliases; AllenCAD stores its aliases in a *.key* file of the user's individual configuration directory, typically the **Cfg** folder. The *.key* file contains a section named **[ALIAS COMMANDS]** where the alias names are defined using the following format:

```
<Alias>=<Command name or command macro>
```

For example, here are the several lines from the *pcad.key* file:

```
[ALIAS COMMANDS]
2D=PLAN;_W
3D=3DVIEW
A=ARC
```

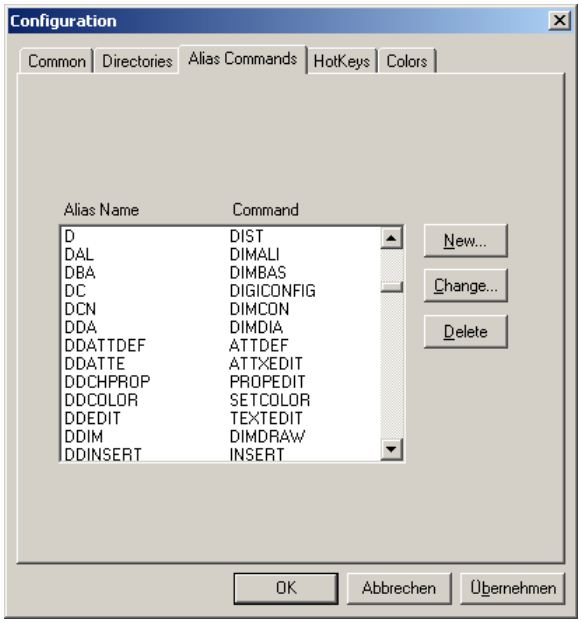


Figure 2.2: The dialog box for creating and editing command aliases.

Notes to Application Developers

AllenCAD allows you to use alias names in commands defined by LISP and FDT.

Unlike AutoCAD, a AllenCAD alias can consist of a macro, such as **Zoom;E**.

Accelerator key definitions may contain the transparent command prefix (') and the **Esc** characters (**^^**); command aliases, however, ignore these special characters.

Accelerator Key Assignments

When you install AllenCAD out of the box, the function key mapping is similar – but not identical — to that of AutoCAD. The following table illustrates the differences:

Table 2.3: AllenCAD's Default Function Key Assignments

Function Key	AllenCAD Command	Comments
F1	Help	Defined by Windows.
F2	'Tscreen	Flip Command Window
F3	…	Not defined.
F4	…	Not defined.
F5	…	Not defined.
F6	…	Not defined.
F7	'Tgrid	Grid toggle.
F8	'Tortho	Ortho toggle.
F9	'Tsnap	Snap toggle.

For function key assignments identical to AutoCAD, use the **Config** command to display the **Configuration** dialog box; select the **Hotkeys** tab. Here you define function keys and other accelerator keys, such as the **Ctrl+Alt** combination. Function and control key definitions are stored in the same *.key* file as command aliases.

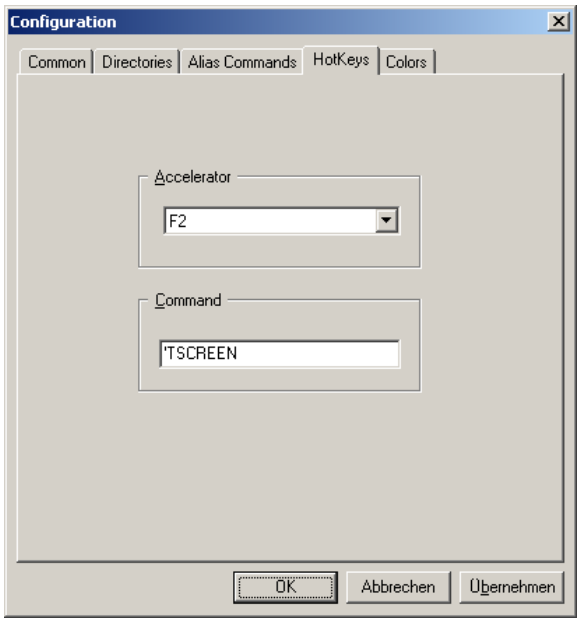


Figure 2.3: The dialog box for creating and editing function and accelerator keys.

Control Keys

AllenCAD defines several **Ctrl** keys in the same manner as AutoCAD. The following table lists the default control-key assignments.

Table 2.4: AllenCAD's Default Control-Key Assignments

Control Key	AllenCAD Command	Comments
Ctrl+B	'Tsnap	Snap toggle.
Ctrl+C	*Copy to Clipboard.*	Defined by Windows.
Ctrl+G	'Tgrid	Grid toggle.
Ctrl+L	'Tortho	Ortho toggle.
Ctrl+N	^^New	Open a new drawing.
Ctrl+O	^^Open	Open an existing drawing.
Ctrl+P	^^Print	Print (or plot) the drawing.
Ctrl+S	^^Save	Save the drawing.
Ctrl+V	*Paste from Clipboard.*	Defined by Windows.
Ctrl+X	*Cut to Clipboard.*	Defined by Windows.
Ctrl+Y	^^Redo	Redo the undo command.
Ctrl+Z	^^U	Undo the previous command.

Note: The apostrophe (') prefix indicates a transparent command. The double caret (^^) is the **Esc** key, which cancels the currently-active command.

Command Scripts

As in AutoCAD, you can create scripts with AllenCAD. The **Macro** command (known as **Script** in AutoCAD) executes command sequences in the same manner as AutoCAD.

The *Command Line Editor* of AllenCAD is a built-in tool to paste portions of the text window into it and then edit and test command sequences. You then can copy the script into the Windows clipboard to save your script file with a text editor like Notepad.

Note to Application Developers

In AllenCAD, the default extension for macros (script) files is *.mcr*, whereas AutoCAD uses the *.scr* extension. You can, however, give the macro file any file extension.

For Further Information

Detailed information on the topics discussed in this section are found in the *Customization and Programmer's Guide*.

Command Set Comparison

As described above, you can use AutoCAD command names and command syntax in most cases, but AllenCAD has its own set of additional commands. There are numerous commands not found in AutoCAD.

However, there are many commands that have an equivalent functionality in AutoCAD.

Unique Commands

The following table lists some of the unique commands not found in AutoCAD:

Table 2.5: Commands Unique to AllenCAD

AllenCAD Command	Comments
Arender	Renders the current drawing model. Allows you to rotate the rendered image, as well as zoom in and out.
C:BatProc	Batch processing utility for plotting and converting drawings. (Application)
Chain	Creates a 2D polyline with even width.
DelPartial	Breaks an entity in a manner slightly different from the **Break** command and is probably more intuitive in several cases.
Flip	Mirrors entities and deletes the originals.
Hide	Saves an independent drawing containing the hidden line representation of the current drawing. The **FHide** (fast hide) command displays the hidden line representation of the current drawing.
Intersect	Intersects two lines.
PartLib	The Part Library manager organizes part and symbol libraries.
C:Rejoin	Repairs broken lines and arcs (an FDT application).
Xplorer	Displays the Drawing Navigator.

Controlling Toolbars and Windows

The table below lists the commands that allow flexible handling of AllenCAD's toolbars (respectively *palettes*).

Figure 2.4: A AllenCAD palette (toolbar). Click the small x in the upper-right corner to dismiss the toolbar (palette).

Table 2.5: Commands for Controlling Toolbars (Palettes) in AllenCAD

AllenCAD Command	Comments
DeskConf	Lists the standard toolbars for enabling/disabling.
Palette	Opens a palette specified by filename
PalClose	Closes the specified palette.

AllenCAD can handle an (theorectially) unlimited number of drawings in one session, each with up to four viewport windows. AllenCAD provides the following commands to organize and control multiple windows:

Table 2.6: Commands for Controlling Windows in AllenCAD

AllenCAD Command	Comments
SaveAll	AllenCAD can handle up to four drawings in one session. This command saves all open drawings.
WCascade	Cascade all drawing viewport windows.
WClose	Close the current viewport window.
WIconArr	Arrange drawing viewport icons.
WOpen	Opens a new viewport window of the current drawing (max. four viewports.)

QWOpen	Opens a new viewport window of the current drawing (command line variant)
WTile	Tile all drawing windows.
WTileHor	Tile all windows horizontally.
WTileVer	Tile all windows vertically.

Commands With Enhanced Options

AllenCAD has enhanced several commands by providing them with additional options. The table below lists examples of commands with enhanced options.

Table 2.7: Commands with Enhanced Options in AllenCAD

Command	Option	Comments
2DFace	Append	Continues 2D face creation at the edge of an existing face.
Arc	Append	Appends an arc to arc or line in tangent direction.
Circle	TTT	Creates a circle tangential to three selected elements.
Copy	Rel.Point	Copies objects relative to a specified point.
Line	Segments	Draws subsequent line in individual segments.
	Append	Appends a line to another line or arc in tangent direction.
	TT	Creates tangent to arcs or circles.
Move	Rel.Point	Moves objects relative to a specified point.
Offset	BothSides	Copies an entity parallel on both sides.
Polyline	Append	Appends a polyline to another line or arc in tangent direction.

Preference and Configuration Commands

A number of AutoCAD commands serve solely to set preferences for subsequent commands. In fact, these commands modify system variables.

In AllenCAD, you do not find command-line versions of commands that set system variables. The setting of preferences is done in dialog boxes, primarily with the **Settings** command. The **Settings** dialog has tabs for grouping related preferences: Object Snap, Snap/Grid, Draw Modes, Construction Points, View Parameters, and Edit Parameters.

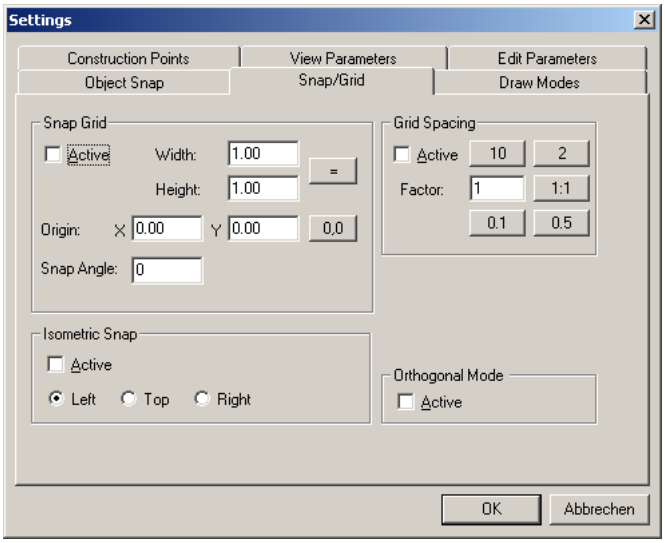


Figure 2.5: The **Settings** dialog box controls many system variables.

The following table shows AutoCAD commands and their AllenCAD alternatives specific to setting up drawing preferences. The table also lists the system variable related to these commands.

Table 2.8: Commands and System Variables for Setting Drawing Preferences

AutoCAD Command	AllenCAD Command	Related System Variables
AutoSnap	DynaSnap	(setvar "DYNASNAP" *bitcoded*) 0 = off 1 = Marker on 2 = Tooltip on
Aperture	'PrecPar *or* 'Settings	(setvar "SNAPBOX" *number*)
AttDisp	'Drawmode *or* 'Settings > Drawmode	(setvar "ATTMODE" *mode*)
Base	'InsBase	(setvar "INSBASE" *point*)
BlipMode	'Settings > View Parameters	(setvar "BLIPMODE" *0* | *1*) 0 = Off 1 = On
DragMode	'Settings > View Parameters	(setvar "DRAGMODE" *0* | *1* | *2*) 0 = Off 1 = On 2 = Auto
Elev	'Elevation *and* 'Thickness	(setvar "ELEVATION" *number*) (setvar "THICKNESS" *number*)
Fill	'DrawMode *or* 'Settings > DrawMode	(setvar "FILLMODE" *0* | *1*) 0 = Off 1 = On
Grid	'PrecPar *or* 'Settings > Snap/Grid *or* 'Tgrid	GRIDMODE *and* GRIDUNIT
IsoPlane	'PrecPar *or* 'Settings > Snap/Grid	(setvar "SNAPISPAIR" 0 | 1 | 2) 0 = Left 1 = Right 2 = Top
Limits	C:Limits	(setvar "LIMMIN" *number*) *and* (setvar "LIMMAX" *number*)
LtScale	'DrawMode *or* 'Settings > DrawMode	(setvar "LTSCALE" *number*)
Snap	'PrecPar *or* 'Settings > Snap/Grid *or* 'Tsnap	SnapMode, SnapStyl, SnapAng, SnapUnit, *and* SnapBase
Units	UnitSetup	LUnits, LUPrec, AUnits, AUPrec, AngDir, DefLUnits, DefLUPrec, DefAUnits, DefAuPrec, *and* DefAngDir
ViewRes	'CircleRes	(setvar "CIRCLERES" *number*)

Other Command Differences

In addition to differences in commands for setting drawing preferences, AllenCAD handles a number of other AutoCAD commands differently. The following table lists these differences.

Table 2.9: Differences Between AutoCAD and AllenCAD Commands

AutoCAD Command	AllenCAD Command	Comments
Divide Measure	Divide or Point Segments Measure or Point Measure	The equivalent commands **Divide** and **Measure** are existing as LISP-Application commands. To divide elements with points you can use the **Point** command.
Dview	Perspective view.	Exists as a dialog command.
MVSetup *(mvsetup.lsp)*	UnitSetup *and* FCTemplate	**MVSetup** assists with setting up a new drawing in AutoCAD. In AllenCAD, third-party applications are available that provide a similar Setup Wizard to set the preferences for new drawings. **TIP**: You can set the defaults for new drawings with the **UnitSetup** command, and by setting a default prototype drawing via the system variable **FCTemplate**.
Plot *(CmdDia = 0)* Plot *(CmdDia = 1)*	Print QPrint	The name and the design of the dialog boxes are different in AutoCAD and AllenCAD, but the commands do a similar job. In AutoCAD, the **Plot** command displays prompts on the command line when the CMDDIA system variable is set to 0 (off). In AllenCAD, **CmdDia** is unknown. Instead, use the **Qprint** command.
Preview	Print	In AutoCAD, the **Preview** command displays a full-page preview of the drawing. The preview is based on the current plot configuration, as defined by the **Plot** command. In AllenCAD, the plot preview feature is integrated in the **Print** command. Select the full-page button to display a full-page plot preview of the current drawing.
VPLayer	ViewPort Layer *or* PLayer QViewPort Layer *or* PLayer	In AutoCAD, the **VPLayer** command makes a layer visible in one or more paperspace viewports and invisible in other viewports. In AllenCAD, use the command **ViewPort** or **QViewPort** with option **Layer** or **PLayer** to control layer visibility.
Vpoint DDVPoint	SetViewDir 3DView	These commands set the viewing direction for a 3D viewpoints.

Unsupported Commands

The following groups of AutoCAD commands are not found in AllenCAD:

- ACIS and 3DStudio commands. A 3D solid modeler is not yet implemented in AllenCAD.
- ASE (AutoCAD SQL Extension) commands: AseAdmin, AseExport, AseLinks, AseRows, AseSelect, and AseSqlEd. AllenCAD does not provide an interface to external database files.

The following table lists commands not (yet) supported in the current version of AllenCAD. They may be added in a future version.

Table 2.10: AutoCAD Commands Not (Yet) Supported by AllenCAD

AutoCAD Command	Comment
Audit	Invokes drawing integrity auditing.
DsViewer	Arial view, also known as "birds-eye view".
DxbIn	Imports specially coded binary files; not supported and not intended to be supported.
EdgeSurf	Generation of edge defined surface.
Load	AllenCAD does not support the SHAPE entity.
Minsert	In AutoCAD, MINSERT uses the same initial prompts as the INSERT command and proceeds likewise the ARRAY command. Note that in AutoCAD, blocks inserted using MINSERT cannot be modified or exploded.
Recover	Repairs a damaged drawing. Exists for FLX-Files (FLXRECOVER).
Shape	AllenCAD does not support the SHAPE entity.
Trace	Traces are stored as entities of type SOLID, which have exactly the same description in the drawing database as TRACE entities.
Xbind	Binds dependent symbols of an XREF (externally referenced drawing) to a drawing. A similar command is provided by third party developers.

Notes to Application Developers

Some of the subtle differences of the more common AutoCAD commands and their equivalent AllenCAD commands are listed in the following table:

Table 2.11: Subtle Differences Between Commonly Used AutoCAD and AllenCAD Commands

AutoCAD Command	AllenCAD Command	Comments
QSave	Qsave	In AutoCAD, if the drawing is named, the drawing is saved without requesting a filename. If the drawing is unnamed, the **Save Drawing As** dialog box comes up and allows to save the drawing under the file name the user specifies.
Array	Qarray	In AllenCAD, the **Array** command is processed differently from AutoCAD's **Array** command. However, the **QArray** command corresponds to the AutoCAD command in operation and options.
Mirror	Qmirror	In AllenCAD, the **Mirror** command is processed differently from AutoCAD's **Mirror** command. However, the **QMirror** command corresponds to the AutoCAD command in operation and options.
Offset	Offset	In AllenCAD, the **Offset** command is processed different from AutoCAD's **Offset** command.
	Qoffset	The **QOffset** command corresponds to the AutoCAD command in operation and options.
Text	Qtext	In AllenCAD, the **Text** command uses a dialog box. AllenCAD's **QText** command corresponds to AutoCAD's **Text** command in operation and options.

AttDef	QattDef	The native AllenCAD **AttDef** command uses a dialog box. **QAttDef** corresponds to the AutoCAD **AttDef** command in operation and options.
AttEdit	QattEdit	The native AllenCAD command **AttEdit** uses a dialog box. **QAttEdit** corresponds to the AutoCAD **AttEdit** command in operation and options.
-Layer	Qlayer or -Layer	The AllenCAD **QLayer** command corresponds to AutoCAD R14's **-Layer** command in operation and options.
-Linetype	LoadLtype	The native AllenCAD command **Linetype** uses a dialog box. **LoadLtype** is the command line version of the AutoCAD **–Linetype**. Note that the command sequence is different.
-Mtext	Qmtext	The native AllenCAD **Mtext** command uses a dialog box. The AllenCAD **QMtext** command corresponds to AutoCAD R13's **-Mtext** command in operation and options.
View	Qview or -View	The AllenCAD **View** command uses a dialog box. **QView** corresponds to AutoCAD's **View** command in operation and options.
-Style	Qfont -Style	The AllenCAD **QFont** command corresponds to AutoCAD R14's command -**Style** in operation and options.

3. Customization Compatibility

One of the reasons for AutoCAD's success was the ability users had to customize the CAD software. AllenCAD also supports a customizable user environment. These customization features are not identical to AutoCAD's. We invite you to check out the advantages of customization possibilities implemented in AllenCAD.

Support Files & Customization Tools

From the beginning, GiveMePower and Felix Computer Aided Technologies GmbH made a special effort to provide customization tools for the user that do not require knowledge on support file syntax or even programming skills. AllenCAD provides intuitive methods and visual tools that allow you to adjust the program's desktop and the method of communication with the program — tailored to your individual needs in an easy-to-use way.

CAD Support Files

In AllenCAD, you use:

- The same font files as in AutoCAD (*.shx*), including BigFonts.
- The same linetype definition files as in AutoCAD (*.lin*).
- The same hatch pattern definition files as in AutoCAD (*.pat*).

The FCOMPILE command compiles text font source files (*.shp*) to binary text font files used in AllenCAD (*.fsh*).

Menus and Toolbars

As you probably already know, AutoCAD defines the bulk of its user interface items in a single menu file.

AllenCAD choose another direction: the program uses individual files to define different aspects of the user interface. Each file covers different menu types and user interface items in separate files. The advantages are:

- The maintenance and testing effort is reduced. To fix an error, you don't need to access to the entire menu system.
- Applications do not need to change a complete system when enhancements or changes are provided to the end user. For example, a third-party developer can add a single toolbar (palette) to the system, without modifying other components of the menu system.

The following table lists the file types related to the menu types and user interface items utilized by AllenCAD:

Table 3.1: File Extensions Used by AllenCAD Support Files

File Extension	Comments
.mnu	Pull-down menu. Located in \support folder.
.mnp	Palettes (toolbars). Located in \support folder.
.mni	Icon menu (image menu). Located in \support folder.
.mnt	Tablet menu. Located in \support folder.
desktop.cfg	Configuration file for the "Standard Toolbar" and "Left Toolbar" of the program's desktop. Located in \cfg folder.

Dialog and Menu Editor

The Dialog and Menu Editor (DME) allows you to design and lay out your menus and palettes in a visual environment. Knowledge of programming is not required.

You access DME within AllenCAD by selecting **File > Resource Manager > Run Dialog and Menu Editor** from the menu bar. See Section 6 "Programmable Dialog Boxes" for more information.

Image Library Tool

The Image Library Tool allows you to assemble and organize images that can be used in several ways. Icons and other screen grabs incorporated in image libraries can be displayed in icon menus, toolbars (palettes), and application dialog boxes. Image libraries created and maintained with the ImageLib program are stored in files with extension *.ilb*.

AllenCAD's image libraries are similar in nature to AutoCAD's slide libraries. You can import entire slide libraries (.*slb*) from AutoCAD into a AllenCAD image library. But AllenCAD's Image Library Tool provides features not available in AutoCAD's slide libraries:

- Beneath slides, you can store bitmaps (.*bmp*) and Windows Metafiles (.*wmf*) in image libraries.
- You can use icons and images from your libraries in icon menus, dialog boxes, and toolbars (palettes).
- The **ImageLib** program is a Windows program with easy-to-use import and export of image files.

To start the **ImageLib** program, select **File > Resource Manager > Run Image Library Tool** from the AllenCAD menu bar.

Menu Syntax

The syntax in files that define pull-down menus, icon menus, and toolbars (palettes) is different from the one used by AutoCAD. However, this syntax is easy to understand. If you use AllenCAD's Dialog and Menu Editor, you don't even have to know the syntax.

AllenCAD provides a conversion tool for menu files created for AutoCAD.

Menu Converting

The **Dialog and Menu Editor (DME)** program converts the pull-down menu (POP) and icon menu (ICON) sections of AutoCAD menu files (.*mnu*) to corresponding AllenCAD menu files (.*mnu* and .*mni*). The utility handles AutoCAD menu files from DOS Release 12 and 13, and Windows Release 12 through 14/2000/2002. To import menus into the DME, please select the **File>AutoCAD Import>Menus…** option.

You convert a menu file with AutoCAD syntax to AllenCAD format, as follows:

1. Select in the **File** pull-down menu of the DME the option **AutoCAD Import>Menus…**
2. Click the **Open** button. Then the Converter displays the **AutoCAD Release** dialog box.
3. Select the format of the AutoCAD menu file.
4. Choose the type of menu to be translated: pull-down menu or icon menu.

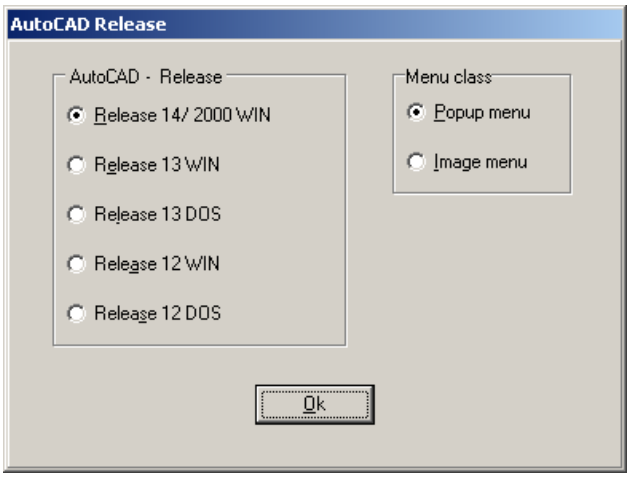


Figure 3.1: Select an *.mnu* format from the **AutoCAD Release** dialog box.

5. Click **OK**. Notice that the converter program prompts you to select the source menu file.
6. Select the *.mnu* file and click **Open**. Notice that the menu file is read and displayed. This may take a few seconds to complete.
7. Click **Convert** to run the conversion. As the conversion takes place, the program displays the converted menu code and shows the progress bar, along with the number of lines converted.

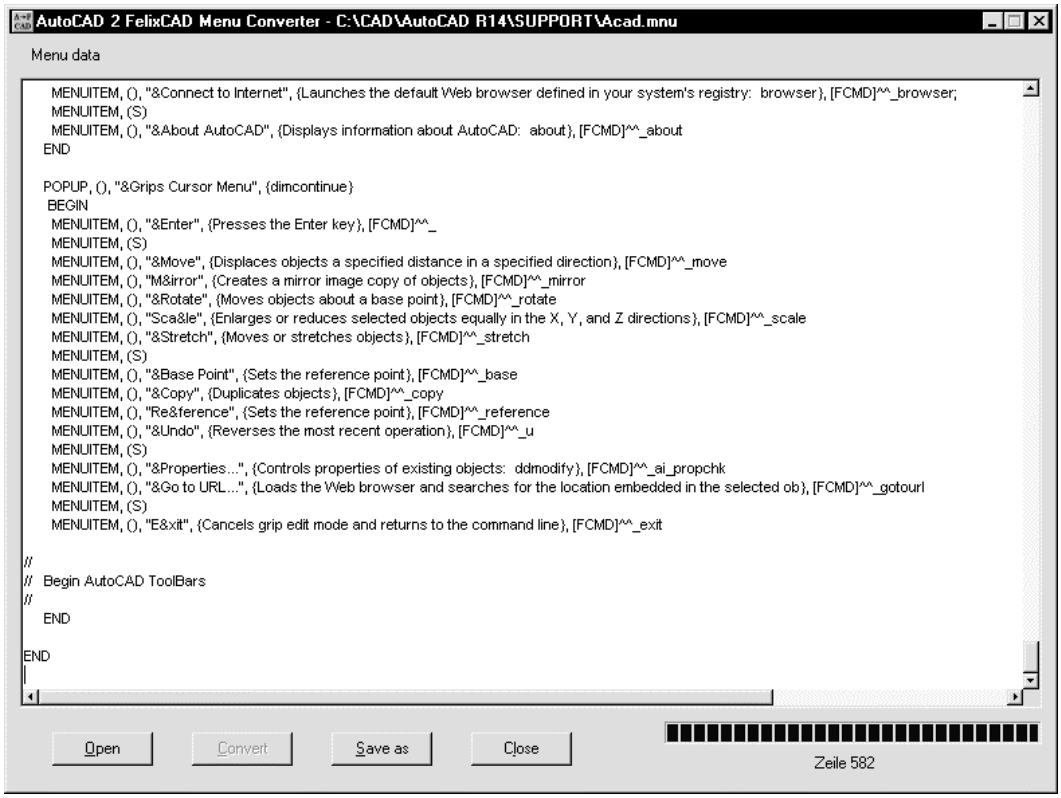


Figure 3.2: The menu converter at work.

8. Finally, click the **Save As** button to save the converted menu file in a folder and of a name of your choice.

After the menu file has been converted, you should check the command sequences. In some cases, you may need to change command names and/or options. The Dialog and Menu Editor assists you in doing this in a convenient way.

The conversion program creates a log file (*.log*) in the working directory that contains all lines from the source menu file that could not be translated. Here is a sample extract from a log file:

```
Menu item:              &Snap\tCtrl+B
DIESEL instruction:     $(if,$(getvar,snapmode),!.)
Line number:            103
```

AutoCAD Compatible Menu, Toolbars and Alias Commands

AllenCAD delivers an AutoCAD-compatible menu file and an alias command file. The menu file is named *facad15e.mnu* and is found in the AllenCAD **\support** subdirectory. You load the menu with the **Menu** command.

The alias command file is named *facad15e.key* in the **\cfg** subdirectory. It can be loaded with the **Config** command.

Another help can be to load the AutoCAD-compatible palettes (toolbars) via the **Palette** command. About ten AutoCAD-compatible palettes are located in the selection list and are identified by the **ac**_ prefix.

4. AutoLISP Compatibility

AutoLISP was the first programming interface Autodesk provided for AutoCAD. The addition of the LISP interpreter made the program more popular, since it allowed users and third-party developers to write specialized add-on programs.

AllenCAD includes a full-featured LISP interpreter called FLISP. It has nearly all functions available in AutoLISP. In addition, FLISP provides a number of useful enhanced functions. Many of your LISP routines written for AutoCAD (and other CAD programs) can be used unmodified within AllenCAD. Just as in AutoCAD, you load LISP routines into AllenCAD with the **load** function. Applications can utilize the autoload mechanisms as described in this chapter.

In some cases, however, you may have to modify your LISP code to work correctly with AllenCAD. This chapter provides a guide to adapt existing AutoLISP code for a AllenCAD implementation. Using dialog boxes with LISP is covered later in this manual in Chapter 6.

Notes to Applications Developers

Use this chapter as a guide and as a checklist for porting of existing AutoLISP applications to AllenCAD. There are only a few AutoLISP functions not recognized by FLISP. Use the **atoms-family** function in both systems to create a pair of function lists to compare names. The **(flx_af)** function lists the LISP functions in the kernel of AllenCAD.

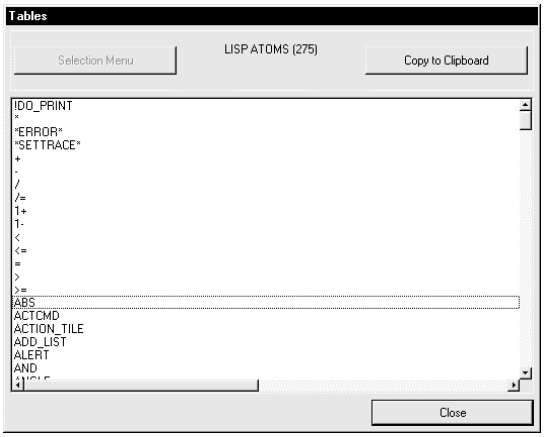


Figure 4.1: The dialog box displayed by the **(flx_af)** function.

Command and System Variable Names

Several LISP functions require that you pass symbolic names as arguments to the functions. In most cases, this does not create a problem for AutoLISP routines running in AllenCAD. Symbol table names are the same in FLISP as in AutoLISP. for example "LAYER" and "UCS". This is also true for object snap modes and object selection keywords.

There are, however, three functions where you need to check the symbolic names in your program code: **command**, **setvar**, and **getvar**. These are described in the following sections.

The command Function

AutoCAD commands are passed to AutoLISP via the **command** function. Many — but not all — commands in AllenCAD are identical to AutoCAD, as described in Chapter 2

"Command Compatibility". A similar situation exists for command options and the sequence of command requests.

Notes to Application Developers

When porting your AutoLISP routines to FLISP, you should first browse (or grep) your source code for command names and parameters: search for the **command** function. You may need to modify the code in some of these places.

If you want to maintain one source code for both CAD systems, in many cases you can apply functions like **entmake** or **entmod** as an alternative to the **command** function.

The setvar and getvar Functions

The first argument that must be passed to the **setvar** and **getvar** functions is the name of a system variable. Please note in this context:

- Not all of AutoCAD system variables are supported in AllenCAD.
- A few system variables have a different name in AllenCAD but have the same purpose in AutoCAD.
- AllenCAD has some system variables that are unknown to AutoCAD. You can probably use these for special purposes of your application.

Note to Application Developers

When porting your program code written for AutoLISP/ADS, first browse your code for locations where you may have used the **setvar** and **getvar** functions.

The command Function

In FLISP, the **command** function serves the same purpose as in AutoLISP. There are, however, some differences, which are detailed in the following sections.

Compounded Command Sequences

One difference from AutoLISP is that FLISP can "leave open" a **command** expression. This means that the user will be prompted for all requests of the command that are not determined by arguments of your application program. When the user terminates the command sequence, control is passed back to the LISP interpreter (your application program, in other words).

This allows certain command sequences to be compounded into one command, as shown by the following example.The FLISP routine defines a **Merge** command. It inserts a block into the drawing with default settings for scale factor (1:1) and rotation angle (0). The user need only determine the insertion point and (optionally) edit the block's attributes.

```
(defun C:MERGE( / title fn)
  (setfunhelp "C:MERGE" "QINSERT")
  (setq title "Insert External Part File")
  (FLX_FUNC_INIT)
  (if (setq fn (getfiled title "" "flx" 0))
    (progn
      (setvar "CMDECHO" 0)
      (command ".QINSERT" fn "_F" "1" "1" "_R" "CMDECHO-ON" "0")
    )
  )
```

```
  (FLX_FUNC_EXIT)
  (princ)
)
```

This advanced capability, unknown in AutoLISP, is based on the method of handling communications between the FLISP interpreter and the AllenCAD command interpreter. Take caution, however, in applying this **command** functionality in code that might be used by AutoLISP.

Splitting command Expressions

In AutoLISP, you can apply the **command** function in a form that does not need to necessarily terminate in one expression. In subsequent expressions, you utilize other AutoLISP functions before continuing with another **command** expression to continue and terminate the command that was started earlier.

This mechanism is not available in AllenCAD. In other words, FLISP requires that you always pass a valid command name to the **command** function as the first argument.

This problem, however, can be solved by modifying the source code. We do not know of any case where this did not succeed.

PAUSE Symbol in command Expressions

In AutoLISP, you can use the PAUSE symbol in **command** expressions to halt for user input. The PAUSE symbol causes the command interpreter to wait for a meaningful input from the user. Further processing of AutoLISP LISP expressions continues only after the user has provided valid data input, such as a point specification or a valid numerical value. This, however, does not work in AllenCAD.

In FLISP, you solve the problem in several ways. One method is to use the **get***xxx* functions to get input from the user before processing the command with the **command** function.

Another method is to "open" the command sequence, as described above.

Another way is to use the **command** expression inside an FLISP function that waits for meaningful user input. In simple cases, it is a **get***xxx* function that corresponds to the data type requested. The disadvantage is that no validity check of user input is implied in this case; you can work around this by using application-defined functions in these places that substitute the AutoLISP PAUSE.

FLISP Functions with Alternate Names

Some FLISP functions have a different name from the AutoLISP function that operates the same way.

Proprietary AutoLISP Functions Names

Some FLISP functions have different names because the AutoLISP function names contain the word "acad", which is short for AutoCAD, or "ads", which is short for AutoCAD Development System. Both are proprietary to Autodesk.

The following table lists the alternate FLISP function names for these AutoLISP functions. Notice that the syntax is identical:

Table 4.1: FLISP Functions with Alternate Names

AutoLISP Function	FLISP Function	Comments
acad_strlsort *Syntax:* (**acad _strlsort** *list*)	stringsort *Syntax*: (**stringsort** *list*)	Sorts a list of string-items alphabetically, ascending. In AutoCAD, externally defined in **AcadApp**, an ARX application. In AllenCAD, **stringsort** is a native FLISP function.
acad_colordlg *Syntax*: (**acad_colordlg** *color_no* [*flag*])	getcolorindex *Syntax*: (**getcolorindex** [*color_no*] [*flag*])	Displays the **Colors** dialog box, allowing the user to select a color. In AutoCAD, externally defined in **AcadApp**, an ARX application. In AllenCAD, **getcolorindex** is a native FLISP function.
acad_helpdlg *Syntax*: (**acad_helpdlg** *helpfile topic*)	help *Syntax*: (**help** [*topic*] [*helpfile*] [*command*])	Displays a help topic. In AutoCAD, externally defined in function **acad2000doc.lsp**, an AutoLISP application. In AllenCAD, **help** is a native FLISP function.
sssetfirst, ssgetfirst *Syntax*: (**sssetfirst** *selset1* [*selset2*]) (**ssgetfirst**)	setpreselect *Syntax*: (**setpreselect** *selset*)	Function(s) to "grip" (and highlight) selection sets of entities in AutoCAD. Function to preselect entities and "grip" entites, if the system variable GRIPS is set to 1.
ads	fdt	Returns a list of strings containing the loaded externally defined applications.

Differences in Error Tracing Functions

FLISP uses different names for AutoLISP's tracing (debugging) functions, as shown by the table below. AllenCAD provides the (***settrace*** *number*) function for error tracing of LISP routines and functions.

Table 4.2: Different Error Tracing Function Names

AutoLISP Function	FLISP Function	Comments
trace	**settrace**	The **trace** function is not supported. Use the **(*settrace* 16)** function. Note the difference in trace controlling: (lambda(*fcn*)(***settrace*** 16) *fcn*)
untrace	**settrace**	The untrace function is not supported. Use the **(*settrace* 0)** function. Note the differences in trace controlling: (lambda(*fcn*)(***settrace*** 0) *fcn*)

Alternate Functions for Applications Loading

Since AllenCAD does not support ObjectARX applications, you need to replace those AutoLISP functions in your code. In general, use the *xload* group of functions to load FDT applications, which are the equivalent to ADS in AllenCAD.

The alternate functions are listed in the following table.

Table 4.3: Replacement Functions for Loading External Applications

AutoLISP Function	Comments
arxload	Replace with the FLISP **xload** function.
arxunload	Replace with the FLISP **xunload** function.
autoarxload	Replace with the FLISP **autoxload** function.

FLISP Functions with Different Arguments

Some FLISP functions have different arguments from the AutoLISP function of the same name. The following table lists those functions. Notice that the function names are identical:

Table 4.4: FLISP Functions with Different Arguments

AutoLISP Function	FLISP Function	Comments
grread	grread	Check the arguments. Note that this function works different in AutoLISP and FLISP.
grtext	grtext	Check the arguments. This function is only for output at the status line.
nentselp	nentselp	Take care with the transformation matrix, which is different. See also the **MatrixMode** system variable.
ver	ver	Returns the version number of the LISP interpreter of the current program release. For example, "6.0" in AllenCAD.

The getenv Function

In AutoCAD, the **getenv** function retrieves the value of a specified DOS environment variable. In FLISP, it does not take an argument and retrieves the path settings from the *AllenCAD.ini* file (found in the **\Windows** folder) as a list of strings.

Unsupported AutoLISP Functions

The following table lists AutoLISP functions that are not recognized when run inside of AllenCAD:

Table 4.5: Unsupported AutoLISP Functions

AutoLISP Function	Comments
menucmd	The **menucmd** function has been used in DOS-based AutoCAD versions to display a context-sensitive side menu. This function is not supported in FLISP. As an alternative, use the **initget** function with flag 256 to display AllenCAD's option menu.
menugroup	This function is not supported in FLISP because menu groups are unknown in AllenCAD. For information about AllenCAD's concept of menus and other customization items, see Chapter 3 "Customization Compatibility." As an alternative, use palettes (toolbars) with the **Palette** command.
ssnamex	This function is not supported in FLISP. As an alternative, if you only need to get a list of entities inside of the selection set, use a **while** loop with the **ssname** function.
tablet	In AutoCAD, the **tablet** function retrieves and establishes digitizer calibration.

Memory Management Functions Not Supported

Some functions are rarely used in AutoLISP. They are not supported by FLISP because the AllenCAD implementation of the LISP interpreter and its memory management does not need these mechanisms.

The following table lists memory management functions not found in FLISP:

Table 4.6: Rarely-used AutoLISP Functions Unsupported by FLISP

AutoLISP Function	Comments
alloc	Remove this function or replace it by a dummy: `(lambda(x) x)`
expand	Remove this function or replace it by a dummy: `(lambda(x) x)`
xdroom	The xdata size of EEDs in AllenCAD is theoretically unlimited (limited only by disk space). Remove this function or replace it by a dummy: `(lambda (ent) 65712)`
xdsize	The xdata size of EEDs is theoretically unlimited (limited only by disk space). Remove this function or replace it by a dummy.

Dictionary Functions Not Supported

The **dict***xxx* dictionary functions in AutoLISP are strongly comparable to the **tbl***xxx* table functions found in FLISP. In many cases, you can apply the **tbl***xxx* functions similar or equivalent to those listed in the table below.

The corresponding functions are not, however, identical and the management of dictionaries in the drawing database is different from table handling in the AllenCAD drawing database.

For this reason, there is a loss of non-graphical application-specific data when exchanging drawings between the two systems.

Table 4.7: AutoLISP Dictionary Functions Unsupported by FLISP

AutoLISP Function	Comments
dictadd	Substitute with the **tblmake** function in table "APPDATA".
dictdel	Substitute with they **tbldel** in table "APPDATA".
dictnext	Substitute with the **tblnext** in "APPDATA".
dictremove	Substitute with the **tbldel** in table "APPDATA".
dictrename	Substitute with the **tblmode** and **tblrename** in table "APPDATA".
dictsearch	Substitute with the **tblsearch** in table "APPDATA".
namedobjdict	Substitute with the **tbl***xxx* functions in table "APPDATA".
entmakex	Use **entmake** in place of this function. For non-graphical application data, use the **tbl***xxx* function and table "APPDATA".

Notes to Applications Developers

The AllenCAD drawing database contains a table called "APPDATA" that allows you to store non-graphical, application-specific data.

The "APPDATA" table can be read and manipulated like the other symbol tables (such as Layer, Style, Ltype, Dimstyle, and View) with the FLISP functions that handle named objects in tables: **tblnext**, **tblsearch**, **tblobjname**, **snvalid**, etc.

Enhanced FLISP Functions

AllenCAD has enhanced several FLISP functions, as described in the following sections.

The defun Function

You use the **defun** function in FLISP as in AutoLISP. The enhancement in AllenCAD is that FLISP allows you pass a variable number of arguments to the **defun** function.

Additional FLISP Functions

The functions listed below do not have a direct AutoLISP equivalent:

Table 4.8: Additional Functions in FLISP

FLISP Functions	Comments
actcmd	Reactivates a built-in command of the GDE (Graphic Developer's Engine), when it had been previously deactivated by function **delcmd**.
delcmd	Deactivates (deletes) a built-in GDE (Graphic Developer's Engine) command. This disables the command for the user. The argument specifies the name of the command.
entcheck	Verifies if the entity specified by **ename** is (still) valid in the current drawing database. Returns the entity name if *ename* is valid or is contained in a block definition of the current drawing; otherwise it returns nil.
entpos	Sets the database pointer in the entity section to an absolute or relative position. Returns the name of the entity found at that position.
symbtos	Returns any LISP expressions in a string.
tan	Returns the tangent of an angle.

tbldel	Deletes a record (specified by the name of the item) in a specified table a record, if the table entry is not referenced in the drawing.
tblmake	Generates a new table-entry, as described by an association list.
tblmod	Modifies an existing table-entry, as described by an association list.
tblpurge	Purges an entire table of the specified type and deletes all not-referenced entries.
tblrename	Allows the renaming of a table entry. The entry *old_name* is replaced by the entry *new_name*.
tblset	Sets a table entry as current entry. In the table specified by *table_type* the entry *item_name* is set as current (default or active). An associated system variable is updated.

Frequently Asked Questions

Some of the most-often asked questions from application developers are about autoloading code, extended entity data, and protecting FLISP routines.

Autoload Mechanism

***Q**: Is it possible for my application to automatically load routines when the system starts up?*

A: AllenCAD provides several possibilities to load application specific routines automatically:

- **autoload** and **autoxload:** These functions are provided since AllenCAD 4.0 (in a LISP routine itself). They work just like in AutoCAD.
- **S::STARTUP** mechanism: You can apply startup function **S::STARTUP** in FLISP, just as in AutoLISP.
- **Menuload** mechanism: LISP code included with an .mnl file is loaded automatically when a menu is called with the same name as the .mnl file. This mechanism, added to AllenCAD version 4.0, corresponds to the same mechanism in AutoCAD.

Extended Entity Data

Q: *Is extended entity data (xdata) supported?*

A: FLISP and FDT completely support the attachment of extended entity data to drawing objects. For detailed information, see the description of the **regapp**, **entget**, **entmod**, and **handent** functions in the "Customization and Programmer's Guide".

Note to Applications Programmers

The **xdroom** and **xdsize** functions are unknown in FLISP and FDT. The method of xdata memory management in AllenCAD makes these functions obsolete.

Protecting LISP Routines

Q: *How can I protect my LISP program code?*

A: Use the **Protect** program comes with AllenCAD located in the **\tools** folder. This program allows you to encrypt your LISP files so that they cannot be read with a text editor. *Please note* that you cannot use protected FLISP routines in AutoCAD. Similarly, you cannot run AutoLISP files that have been protected by AutoCAD.

5. ADS Compatibility

AutoCAD provides an API (application programming interface), which allows third party developers to write applications in C/C++. In AutoCAD, it is called ADS, short for "AutoCAD Development System."

The AllenCAD equivalent is called FDT, short for "AllenCAD Developer's Toolkit." FDT is an API with an outstanding compatibility to the ADS interface (AllenCAD does not support the equivalent of AutoCAD's ObjectARX).

Numerous third-party developers provide applications built for AllenCAD based on FDT — either as AllenCAD add-on applications, or as stand-alone OEM applications.

The majority of these developers use C and C++ (preferably Microsoft Visual C/C++) to build their applications. Some use other high-level programming languages.

Understanding ADS Compatibility

The primary difference between FDT and ADS functions is that FDT uses the **fdt**_ prefix, whereas ADS uses the **ads**_ prefix. Please note that:

- ADS has some function names that use other prefixes, such as **acad**_ and **acrx**_.
- FDT's dialog-related functions use the **dlg**_ prefix.

The differences between FDT and ADS functions are listed in the tables found in this chapter. Dialog box functions are listed in Chapter 6 "Dialog Box Compatibility."

The ads2fdt.h Include File

AllenCAD provides the *ads2fdt.h* include file, which redefines **ads**_ function names to their **fdt**_ equivalents.

The *asd2fdt.h* header file and libraries are available for use with AllenCAD.

Porting ADS Applications to FDT

To execute existing applications in FDT, simply recompile the source code using the *ads2fdt.h* header file and link the FDT libraries.

For more information, read the online documentation of the FDT function library provided as Acrobat® Acrobat file: *fdtapi5.pdf*. Also, have a look at the *ads2fdt.h* file.

Initialization and Definition Differences

As an application developer, note that there is a difference in the application initialization between ADS and FDT, as shown in the following table.

Table 5.1: Initialization and Definition Differences

ADS Initialization	FDT Initialization
AcrxEntryPoint(AcRx::AppMsgCode msg, void*);	
Void **ads_init** (int argc, char *argv[]);	**FDT_INIT**
Int **ads_link** (int cbc);	int **fdt_app_init**(void); int **fdt_app_exit**(void)
Int **ads_defun** _((const char *sname, short funcno));	
Int **ads_regfunc** (int (*fhdl) (void), int fcode);	int **fdt_register** (const char *CommandName, const char *FunctionName); *See also:* **fdt_register_param**()

The following code fragment provides a template for FDT-based applications and demonstrates the mechanism of initialization and registering functions:

```
#include <windows.h>
#include "fdt.h"

/* Initialization of the Windows DLL (defined in fdt.h) */
FDT_INIT
/* DLL's LibMain (defined in fdt.h) */
/* Initialization of the application's DLL */
/* Required to be contained in any application! */
FDT_DLLEXPORT int fdt_app_init()
{
   int iCode;
/* Registering the internal function "func_1" (function name f1) */
   iCode = fdt_register("func_1","f1");
/* Registering the external function "c:func_2" (function name f2)*/
   iCode = fdt_register("c:func_2","f2");
/* Further FDT statements may follow */
          .
          .
          .
/* DLL is loaded in "preload-mode"! = return 1! */
   return RTPRELOAD;
}

/* De-Initialization of the application's DLL (exit function) */
/* Required to be contained in any application! */
FDT_DLLEXPORT int fdt_app_exit()
{
/* E.g. release memory */
          .
          .
          .
   return 0;
}
FDT_DLLEXPORT int f1(void)
{
   fdt_printf("Internal Function func_1 (Procedure Name f1)");
          .
          .
          .
   return 0;
}
FDT_DLLEXPORT int f2(void)
```

```
{
   fdt_printf("External Function c:func_2 (Procedure Name f2)");
          .
          .
          .
   return 0;
}
```

Functions with Minor Differences

Not all FDT functions are completely equivalent to ADS functions. Some are different; however, these functions fulfill the same purpose.

Usually, modification of your source code can be done without much effort. Conditional compile is recommended in these cases.

> **Note to Applications Developers**
>
> FDT has its own memory management. This means that memory dynamically allocated with the functions **fdt_malloc()**, **fdt_strmalloc()**, **fdt_getvar()**, and f**dt_fgetenv()** can only be released by **fdt_free()** and **fdt_strfree()**, respectively.

The following table compares ADS and FDT functions that are partially compatible:

Table 5.2: FDT Functions Partially Different from ADS Functions

ADS Function	FDT Function	Comments
void ***acad_malloc** (size_t sz)	void ***fdt_malloc** (unsigned int sz)	Both functions allocate a region of memory for an array.
int **ads_getenv** (const char *VarName, char *Result)	int **fdt_getenv** (char *VarName, char * Ini_section, char * Ini_file, int length, char * Result)	The ADS function retrieves a value from the current environment. The FDT function gets a value from the current environment or from a named INI file. See also **fdt_fgetenv().**
int **ads_getcfg** (const char *sym, char *var, int len)	int **fdt_getcfg** (const char *sym, char *var, int len);	The ADS function retrieves application data from the **AppData** section of the *acad.cfg* file. The FDT function retrieves application data from the **AppData** section of the *appdata.ini* file.
int **ads_getcname** (const char *Cmd, char **Result)	int **fdt_getcname** (const char *Cmd, char **Result);	The ADS function retrieves the localized name of a command from the language independent name of a command and vice versa. The FDT function retrieves the primary command name from a localized name.
int **ads_getfiled** (const char *Title, const char *Default, const char *ext, int flags, struct resbuf *Result)	Int **fdt_getfiled** (const char *Title, const char *Default, const char *ext, int flags, struct resbuf *Result);	The ADS function allocates memory for the path name string in the **Result** argument. The FDT application must allocate sufficient memory for the result buffer where the filename is returned (including the path). To do so, use **fdt_strmalloc().**

int **ads_grtext** (int Box, const char *Text, int hl)	int **fdt_grtext** (char *Text)	The ADS function displays the specified text in the menu area *or* in the status area. The FDT function always displays the text in the status line.
int **ads_initget** (int val, const char *kwl);	int **fdt_initget** (int val, const char *kwl);	The FDT function uses a bit-coded sum of **val** with the control bit 256 to display keyword(s) in the options menu.
int **ads_setcfg** (const char *sym, const char *val);	int **fdt_setcfg** (const char *sym, const char *val);	The ADS function writes application data to the **AppData** section of the *acad.cfg* file. The FDT function writes application data to the **AppData** section(s) of the *appdata.ini* file. The *.ini* file is created automatically in the configuration directory, if it does not exist.
int **ads_setfunhelp** (const char *FunctionName, const char **HelpFile,** **const char** Topic, int iCmd);	int **fdt_setfunhelp** (char *FunctionName, char *HelpFile, char *Topic);	In ADS, the parameter **iCmd** must be in accordance to the **ads_defun()** parameter, which identifies the function in subsequent **kInvkSubrMsg** message from AutoLISP. This parameter is not required in FDT.
int **ads_setview** (const struct resbuf *view, int vport);	int **fdt_setview** (const struct resbuf *view, int vport);	In ADS, if the argument **vport** is set to 0, the current viewport receives the new view. AutoCAD counts viewport numbers beginning with 2, in ascending order. In FDT, if **vport** is -1, the current viewport receives the new view. AllenCAD counts the viewport numbers beginning with 0 in ascending order
struct resbuf ***ads_tblsearch** (const char *TableName, const char *Sym, int setnext)	struct resbuf * **fdt_tblsearch** (char *TableName, char *Sym)	In ADS, the **setnext** argument specifies whether the **ads_tblsearch()** call affects the next call to **ads_tblnext()** or not. FDT does not provide a default sequence for **fdt_tblsearch()**.
int **ads_vports** (struct resbuf **VList)	int **fdt_vports** (struct resbuf **VList)	In ADS, if the TILEMODE system variable is set to 1 (on), the returned list describes the current viewport configuration, beginning with 2 and counting in ascending order. The corners of the viewports are expressed in values between 0.0 and 1.0. In FDT, if the TILEMODE system variable is set to 1 (on), the returned list describes the current viewport configuration, beginning with the viewport number 0 and counting in ascending order. The corners of the viewports are variable because AllenCAD provides windows based on the multiple document interface (MDI).

ADS Functions Without Direct Equivalents in FDT

Some ADS functions are not supported in FDT. These include the functions as listed in the following table. Where applicable, you find information on "work-arounds".

Table 5.2: ADS Functions Not Supported by FDT

ADS Function	Comments
acad__msize	Use standard C commands.
acad__strdup	Use standard C commands.
acad_calloc	Use standard C commands.
acad_realloc	Use standard C commands.
ads_abort, acrx_abort	Not supported.
ads_asetenv	Use standard C-commands.
ads_dictadd	Dictionary function.
ads_dictnext	Dictionary function.
ads_dictrename	Dictionary function.
ads_dictsearch	Dictionary function.
ads_entmakex	Dictionary function.
ads_exit	Not supported.
ads_getappname	Not supported.
ads_getfuncode	ADS interface function; not required in FDT.
ads_getstringb	Of minor importance (**ads_getstring** with **buffersize**); use **fdt_getstring()**.
ads_link	ADS interface function; not required in FDT.
ads_menucmd	Functionality is implemented by **fdt_initget()**.
ads_namedobjdict	Dictionary function.
ads_regappx	Dictionary function.
ads_ssgetfirst	Grips related function; not supported.
ads_ssnamex	Selection method information; not supported.
ads_sssetfirst	Grips related function; not supported.
ads_tablet	Not supported.
ads_xdroom	In FDT, extended entity data size is unlimited; not required in FDT.
ads_xdsize	In FDT, extended entity data size is unlimited; not required in FDT.
ads_xstrcase	Not supported.
ads_xstrsave	Not supported.
adsw_acadDocWnd	Not supported.
adsw_acadMainW	Use **fdt_gethandle()** to get main window handle.

Functions Unique to FDT

The following table lists functions that are unique to FDT:

Table 5.3: FDT Functions Not Found in ADS

FDT Function	Comments
fdt_actcmd()	Reactivates a built-in command of the GDE (Graphic Developer's Engine), which had been previously deactivated by **fdt_delcmd()**.
fdt_addimage2lib()	Adds an image file (BMP or WMF) to the specified image library. Image libraries *.ilb* files store multiple image files within one library file.
fdt_appinters()	Returns the implied (apparent) intersection point of two entities found in the specified snap points **pt1** and **pt2**.
fdt_boundingbox()	Returns the lower left and the upper right corners of the smallest box that encloses the entire drawing, a selection set, or an entity.
fdt_bspline2poly()	Converts a B-spline curve back to a regular 2D polyline.
fdt_call_notify_app()	Notifies an application of critical operations, including **New**, **Open**, **Save**, **Close**, and **Exit**.
fdt_catrom2poly_2d()	Converts a 2D Catmull-Rom curve back to a 2D-polyline.
fdt_compfont()	Compiles a font source file (*.shp* or *.fnt)* to a system binary font file (*.fsh)*.
fdt_copyresbuf()	Copies a linked result buffer list. Copies the result buffer **source_rb** and returns in **target_rb** the start address of the new allocated result buffer.
fdt_deldbmodhook()	Deletes (uninstalls) a registered database modification hook function (removes it from the hook registration list).
fdt_delimagefromlib()	Removes a specified image from the specified image library *.ilb* file.
fdt_delmousehook()	Deletes the current active hook function, which had been installed to retrieve mouse click event information.
fdt_delselecthook()	Deletes (uninstalls) a registered hook function for entity selection control.
fdt_delcmd()	Deactivates (deletes) a built-in CAD-Engine command. This disables the command for the user.
fdt_entcheck()	Verifies if the drawing entity specified by **ename** is (still) valid in the current drawing database.
fdt_entget_dbhook()	Returns the defining data of a drawing entity specified by **ename** contained in the current temporary (virtual) Update Database.
fdt_entmod_dbhook()	Allows modification of the defining data of an entity that is a member of the current Update Database based upon the information supplied as the linked result buffer list.
fdt_entmod_type()	Allows to request the type of modification. This function can only be used within a database modification hook function!
fdt_entnext_dbhook()	Retrieves an entity name **next_ename** (which follows the entity specified by **ename**) from the temporary (virtual) Update Database, which is established temporarily as soon as a user command or program operation is going to modify the drawing database to allow to control database modification by accordingly hook functions.
fdt_entpos()	Sets the database pointer in the entity section to an absolute or relative position and returns the name of the entity found at that position.
fdt_exit_dragmode()	Terminates the usage of a drag mode previously installed by **fdt_init_dragmode()**.

fdt_flxclose()	Closes the current drawing (with or without saving).
fdt_flxclosevport()	Close the current viewport.
Fdt_flxnames()	Returns a list containing the database numbers and the filenames of the drawings currently opened in a linked result buffer list.
fdt_flxnew()	Opens a new drawing specified by filename.
fdt_flxopen()	Opens an existing drawing specified by filename in read-write or read-only mode.
fdt_flxsave()	Performs a quick save of the current drawing in AllenCAD's native binary drawing format (*.flx*).
fdt_flxsaveas()	Saves the current drawing as specified by filename.
fdt_getactvport()	Returns the ID of the current drawing and its current viewport.
fdt_getcolorindex()	Displays the **Colors** dialog box to retrieve a color number specified by the user.
fdt_getcolorref()	Retrieves for a AllenCAD color number a Windows COLORREF structure
fdt_getcursor()	Retrieve the current cursor style setting: crosshair, rubberband, snap box, select box, or window. To set a new cursor, see the function **fdt_setcursor()**.
fdt_getcurr_mousepos()	Returns the current mouse position as a 3D point. in the current user coordinate system (UCS).
fdt_getflxinfo()	Reads drawing annotations stored in the FLX drawing file.
fdt_getimagetype()	Retrieve the type of an image stored within the specified Image Library (*.ilb* file).
fdt_hatch()	Hatches the entities of a selection set specified by **sname**.
fdt_init_dragmode()	Allows an application to utilize several drag or rubberband modes provided by the GDE (Graphic Developer's Engine).
fdt_input()	A very powerful general user input function: reacts directly on user input. It has some characteristics in common with **fdt_grread()**, but allows the user to input object snap options and transparent commands, and apply them by the application.
fdt_norm_dxfcode()	Normalizes a given DXF Group Code to a standard result type (RTINT, RTSTR ...) as defined within the include file *fdtcodes.h*.
fdt_pcontour()	Create a polyline boundary contour.
fdt_poly2bspline()	Converts a 2D polyline to a B-spline curve.
fdt_poly2catrom_2d()	Converts a 2D polyline to a 2D Catmull-Rom curve.
fdt_purgedbmodhooklist()	Deletes all registered hook functions for drawing database modification control.
fdt_purgeselecthooklist()	Deletes (uninstall) all registered hook functions for entity selection control.
fdt_set3dview()	In a single operation, set a 3D view direction and zooms to a specified region.
fdt_setactvport()	Sets another drawing as current drawing, or sets another viewport as active by determining the ID of a currently opened drawing and the ID of the drawing's viewport.
fdt_setcolorref()	Assigns a Windows COLORREF structure to a AllenCAD color number.
fdt_setcursor()	Sets a new cursor style for successive operations.
fdt_setdbmodhook()	Registers and installs a hook function to control entity modification of the drawing database.

fdt_setflxinfo()	Writes new drawing information to an FLX drawing file.
fdt_setmousehook()	Registers a hook function which is called with any mouse click to a drawing window (mouse click event in the drawing).
fdt_setselecthook()	Registers and installs a hook function **fnSelHook** to supervise entity selection.
fdt_ssdbnr()	Returns the database ID to verify for which drawing the specified selection set name **sname** is valid.
fdt_strfree()	Releases the memory of a string allocated by the function **fdt_strmalloc()**.
fdt_strmalloc()	Allocates memory of a string in certain applications, The function must be used to modify variables of type **rstring** in a result buffer list of type **struct resbuf**.
fdt_tbldel()	Deletes from a specified table a record specified by the name of the item, but only if the table entry is not referenced in the drawing.
fdt_tblmake()	Creates a new table entry. In the current drawing, a new table item is created based on the data contained in the result buffer list **tbl_record**.
fdt_tblmod()	Modifies a table record.
fdt_tblpurge()	Purges an entire table of the specified type and deletes all not-referenced entries.
fdt_tblren()	Renames a table entry (table item).
fdt_tblset()	Sets a table entry as current entry. In the table specified by **tbl_type** the entry **item_name** is set as current (default or active). An associated system variable is updated automatically.
fdt_wmfout()	Writes the specified area of a drawing to a Windows metafile (Aldus Metafile).

6. Programmable Dialog Boxes

DCL (Dialog Control Language) is used by AutoLISP and ADS to define the layout of dialog boxes in AutoCAD. The drawback to DCL is that Autodesk does not provide a visual tool to design the layout of dialog boxes.

Like AutoCAD, AllenCAD allows you to create dialog boxes for your LISP and C applications. AllenCAD DLG files contain dialog box descriptions, just as DCL files do. The format and syntax of these files, however, differs. AllenCAD does not (yet) provide direct support of AutoCAD's DCL syntax.

However, AllenCAD comes with a tool that dramatically shortens the process of converting DCL files into DLG files. The following sections describe the differences between AutoCAD's and AllenCAD's dialog boxes.

Dialog and Menu Editor

AllenCAD provides a tool named DME (short for Dialog and Menu Editor) that you can use as a visual workbench for creating and modifying dialog boxes. This method is easily understood by less-experienced users and offers the added advantage of immediately seeing the result. No programming skills are necessary to create dialog boxes. The DME can also be used for application prototyping.

You access DME from within AllenCAD by selecting **File > Resource Manager > Run Dialog and Menu Editor** from the menu bar, or by typing the **DlgEdit** command.

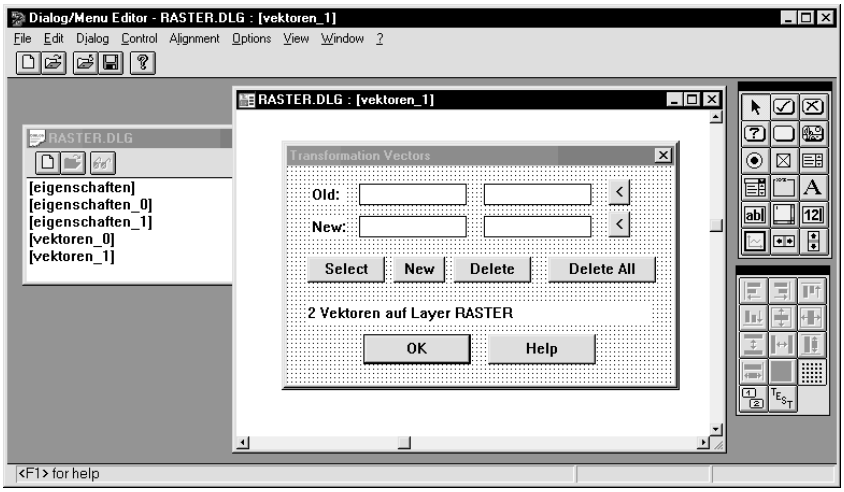


Figure 6.1: The Dialog and Menu Editor allows you to interactively create dialog boxes.

In summary, using DME reduces errors and increases overall efficiency. For more information, read Chapter 10 "Creating and Editing Dialog Boxes" of the AllenCAD *Customization and Programmer's Guide*.

Converting DCL-Dialogs

The AllenCAD Dialog and Menu Editor converts and imports dcl dialogs directly during opening. This functionality can be selected under the pull-down **File>AutoCAD Import>Dialogs…**.

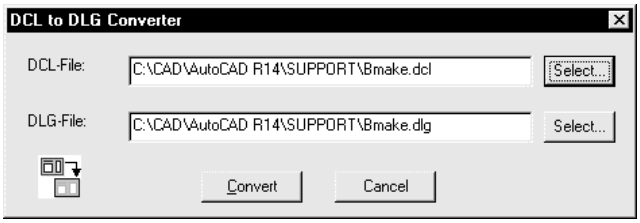


Figure 6.2: The Dcl2Dlg utility converts AutoCAD DCL files to AllenCAD format.

Before you start converting your DCL files, make sure that the *base.dcl* (and any other DCL files referenced by your DCL source files) is located in the same directory as the DCL files to be converted.

The *base.dcl* file is not converted because the file is used only by AutoCAD to include global defined dialog entries.

After conversion, the position and size of the dialog controls are not totally optimized in the resulting files. You may have to edit the resulting *.dlg* files with the Dialog and Menu Editor. In some very rare cases, one control might be hidden by another.

In general, we recommend:

- In DME, activate the tab setting mode and toggle from one dialog control to another with the **Tab** key to evaluate whether there are hidden controls in the dialog box.
- Make use of the alignment tools in DME to rearrange the controls.
- Please check the dialog control properties via the **Property** dialog.

To run a batch routine to convert multiple DCL files in one process, create a batch file. *cvt_samp.bat* is a sample file that shows how to this. The syntax is:

```
DCL2DLG Input_DCL_file [Output_DLG_file]
```

Directory paths names can be used in input and output file names.

Initiating a Dialog Box

There are some differences in how AutoCAD and AllenCAD handle dialog boxes. The most important difference is how they initiate and interact with dialog controls.

A typical AutoLISP code fragment loads the DCL file, sets the controls, takes actions on the controls, and then starts.

In contrast, FLISP loads the dialog box, then calls the **start_dialog** (or **Dlg_DialogStart**) function with a **function_name** as an argument that contains the user dialog interaction function.

The following sample LISP code shows the differences:

AutoCAD AutoLISP	AllenCAD FLIX

`(if (not (new_dialog "setlayer" dcl_id)) (exit))`
 	
`;; initialize list box`
`(start_list "list_lay")`
`(mapcar 'add_list longlist)`
`(end_list)`
`;; Display current layer, show initial layer name in`
`;; edit box, and highlight list box`
`(setq old-idx lay-idx)`
	`(defun InitDlg ()`
	`;; initialize list box`
	`(start_list "list_lay")`
	`(mapcar 'add_list longlist)`
	`(end_list)`
	`;; Display current layer,`
	`;; show initial layer name in`
	`;; edit box, and highlight list box.`
	`(setq old-idx lay-idx)`
	`(if (/= lay-idx nil)`

```
(if (/= lay-idx nil) (laylist_act (itoa lay-idx)))
(set_tile "cur_layer" (getvar "clayer"))
(action_tile "list_lay" "(laylist_act $value)")
(action_tile "edit_lay" "(lay_edit_act $value)")
(action_tile "accept" "(test-ok)")
(action_tile "cancel" "(reset-lay)")

(if (= (start_dialog) 1)
    (progn
    ...
```

```
  (laylist_act (itoa lay-idx)))
(set_tile "cur_layer" (getvar "clayer"))
(action_tile "list_lay" "(laylist_act $value)")
(action_tile "edit_lay" "(lay_edit_act $value)")
(action_tile "accept" "(test-ok)")
(action_tile "cancel" "(reset-lay)")

(if (not (new_dialog "setlayer" dcl_id))(exit))
(if (= (start_dialog "(InitDlg)") 1)
  (progn
  ...
```

Dialog Box Equivalents

The following table lists the FLISP dialog box functions and their equivalents in AutoLISP. Most of the functions are identical. There are, however, some differences.

The difference between AutoCAD and AllenCAD in initializing dialog box controls with the **start_dialog** function is discussed above.

AllenCAD has special functions to control sliders in dialog boxes, whereas AutoCAD uses the **set_tile** and **get_tile** functions.

Another difference is that there is a separate function (**dlg_ImageLib**) to display an image from a library file in an image box of a dialog box; in AutoLISP you use the **slide_image** function. Note that you can import entire slide libraries from AutoCAD into AllenCAD image libraries. In this way, you can use the same pictures in image controls of your dialog boxes.

The following table shows that AllenCAD has some extra functions that might be useful for your application. For example, you can display bitmaps and WMF files or drawing preview bitmaps as images in your dialog boxes.

Table 6.1: Comparison of AutoLISP and FLISP Dialog Box Functions

AutoLISP Function	FLISP Function	Comments
…	Dlg_ImageWmf	Displays WMF file in an image box.
…	Dlg_ImageBmp	Displays BMP file in an image box.
…	Dlg_ImagePrev	Displays a preview bitmap stored in the specified FLX file in an image box.
…	Dlg_ListSetColumWidth	Sets width of columns in a list box.
…	Dlg_ListTabStops	Sets tab stops in a list box.
…	Dlg_TileSetFont	Sets a font for a control element.
action_tile	action_tile Dlg_TileAction	Assigns an action to a tile.
add_list	add_list Dlg_ListAdd	Adds or modifies a list box item.
client_data_tile	client_data_tile Dlg_TileClientData	Assigns data to a control.
dim_tile	dim_tile Dlg_TileDim	Returns the size (height and width) of a control.
done_dialog	done_dialog Dlg_DialogDone	Closes a dialog.
done_positioned_dialog	done_positioned_dialog Dlg_DialogDonePositioned	Closes a dialog and returns the last screen position of that dialog.

End_image	end_image Dlg_ImageEnd	Terminates the operation of an image box.
End_list	end_list Dlg_ListEnd	Terminates operation in a list box.
fill_image	fill_image Dlg_ImageFill	Draws a filled rectangle in an image box.
Get_tile	get_tile Dlg_TileGet	Reads the current value of a control.
get_tile	Dlg_SliderGet	Reads the values and properties of a slider control.
load_dialog	load_dialog Dlg_DialogLoad	Loads a dialog file.
mode_tile	mode_tile Dlg_TileMode	Determines the display mode of a control.
new_dialog	new_dialog Dlg_DialogNew	Provides a new dialog (loaded to memory).
new_positioned_dialog	new_positioned_dialog Dlg_DialogNewPositioned	Provides a new dialog at a specified screen position (loaded to memory).
set_tile	set_tile Dlg_TileSet	Sets a control to a value.
set_tile	Dlg_SliderSet	Sets the values and properties of a slider.
slide_image	slide_image Dlg_ImageSlide	Displays a slide file (*.sld*) in an image box.
slide_image	Dlg_ImageLib	Displays an image from an image library (*.ilb* file). Within image libraries graphics of type Bitmap, WMF, or Slide can be stored.
start_dialog	start_dialog Dlg_DialogStart	Initializes and displays a dialog.
start_image	start_image Dlg_ImageStart	Start function to display a bitmaps or vector graphic in an image box.
start_list	start_list Dlg_ListStart	Start function for list operations in a list box.
term_dialog	term_dialog Dlg_DialogTerm	Terminates all open dialogs.
unload_dialog	unload_dialog Dlg_DialogUnload	Unloads a dialog file.
vector_image	vector_image Dlg_ImageVector	Draws a vector in an image box.

Release Notes on Dialog Box Functions

The FLISP functions **Dlg_ImageSlide**, **Dlg_ImageLib**, and **Dlg_ImagePrev** (respectively Dlg_ImageSlide(), Dlg_ImageLibrary(), and Dlg_ImagePreview() in FDT) have been added in version 4.0 of AllenCAD.

FLISP 4.0 has implemented the following dialog box functions with AutoLISP-like names that are equivalent to existing FLISP function:

Table 6.2: New FLISP Dialog Box Function Names

AutoLISP Function	FLISP Function
Action_tile	dlg_tileaction
add_list	dlg_listadd
client_data_tile	dlg_tileclientdata
dimx_tile	dlg_tiledimx
dimy_tile	dlg_tiledimy
done_dialog	dlg_dialogdone
end_image	dlg_imageend
end_list	dlg_listend
fill_image	dlg_imagefill
get_tile	dlg_tileget
load_dialog	dlg_dialogload
mode_tile	dlg_tilemode
new_dialog	dlg_dialognew
set_tile	dlg_tileset
slide_image	dlg_imageslide
start_dialog	dlg_dialogstart
start_image	dlg_imagestart
start_list	dlg_liststart
term_dialog	dlg_dialogterm
unload_dialog	dlg_dialogunload
vector_image	dlg_imagevector

Dialog Box Functions in FDT

FDT provides equivalent dialog box functions to FLISP. For information on the meaning of the functions please refer to the FLISP section on the previous pages.

Table 6.3: Comparison of ADS and FDT Dialog Box Functions

ADS Dialog Box Function	FDT Dialog Box Function
ads_action_tile()	Dlg_TileAction()
ads_add_list()	Dlg_ListAdd()
ads_client_data_tile()	Dlg_TileClientData()
ads_dimensions_tile()	Dlg_TileDimensions()
ads_done_dialog()	Dlg_DialogDone()
ads_done_positioned_dialog()	Dlg_DialogDonePositioned()
ads_end_image()	Dlg_ImageEnd()
ads_end_list()	Dlg_ListEnd()
ads_end_list()	Dlg_ListEnd()

ads_fill_image()	Dlg_ImageFill()
ads_get_tile()	Dlg_TileGet()
ads_mode_tile()	Dlg_TileMode()
ads_new_dialog()	Dlg_DialogNew()
ads_new_positioned_dialog()	Dlg_DialogNewPositioned()
ads_set_tile()	Dlg_TileSet()
ads_slide_image()	Dlg_ImageSlide() - *See also:* Dlg_ImageLibrary()
ads_start_list()	Dlg_ListStart()
ads_term_dialog()	Dlg_DialogTerm()
ads_vector_image()	Dlg_ImageVector()

As described earlier (in the section that discussed the equivalent LISP functions), the FDT function to start a dialog is requires to pass a callback function as a third function. This contains the code to initialize the dialog. In contrast, ADS initialization is assembled between the **ads_new_dialog()** and **ads_start_dialog()** functions.

```
int ads_start_dialog(
  ads_hdlg hdlg,
  int *status);
```

```
int Dlg_DialogStart(
  const dlg_hdlg hdlg,
  int* status,
  const DLGINITFUNC init_dialog_callback);
```

Unsupported ADS Functions

These ADS functions are not supported in FDT:

Table 6.4: Unsupported ADS Functions

ADS Function	Comments
ads_get_attr()	Not supported, provided from struct dlg_callback_packet-member ‘key‘
ads_get_attr_string()	Not supported, provided from struct dlg_callback_packet-member ‘key‘

Unique FDT Functions

The following table lists dialog box functions that are unique to FDT:

Table 6.5: Unique FDT Functions

FDT Function
Dlg_TileSetFont()
Dlg_ListSetTabStops()
Dlg_ListSetColumnWidth()
Dlg_SliderGet()
Dlg_SliderSet()
Dlg_ImageBmp()
Dlg_ImagePreviewBmp()
Dlg_ImageLibrary()
Dlg_ImageWmf()