II. AutoCAD Development System

AutoCAD Development System (ADS) is a compiled-language program environment for developing AutoCAD® applications with the ADS library of C functions. In the ADS program environment, applications run as executables and as separate processes from AutoCAD. In the ARX program environment, applications run as dynamic linked libraries (DLLs) and as a single process with AutoCAD.

At present, ADS is supported by both AutoCAD Release 12 (running on the Sun workstations) and AutoCAD Release 13 (running on the Window NT PCs). Many ADS applications have been developed in the past many years. We have decided to use ADS as our interactive graphical programming environment.

In the ADS program environment, the application has a *main()* and an event loop communicating with AutoCAD through *ads_link()*. The return value from *ads_link()* tells the application what to do. The application in the ARX program environment has a single function entry point instead of a *main()*. This function takes an argument that tells the application what to do.

General Requirements

Some general conditions for building a successful ADS application:

- The application must contain a preprocessor directive to include the ADS header file *adslib.h*.
- The application must be linked to the ADS object library, either at compile time or by subsequently calling a linker.
- An application must call standard C libraries. There are no known conflicts between ADS and the standard C library functions, types, and variables.
- AutoCAD is supported only on systems that have math coprocessing. If your compiler requires you to turn on math coprocessor code generation explicitly, you must do so.

Compatibility with AutoCAD Release 12 -- Windows

The new program architecture is substantially changed from Release 12, and Release 12 applications must be rebuilt to run on Release 13. The Win32™ development environment is required software. Install Win32 development tools, if necessary, and rebuild your Release 12 Windows ADS applications to work with Release 13.

Sending command strings from an ADS (or other) application is done with the WM_COPYDATA message in Release 13 rather than the WM_ACAD message that was used in Release 12.

Supported Compilers and Operating Systems

Different platforms require different compilers, linkers, and other software tools. To find out what support became available too late to be published in this document, check the *readme* files that come with AutoCAD.

Linking Object Files

Either the operating system or your compiler must have a linker capable of handling object files for the hardware platform intended for your ADS application. ADS applications must be linked to the ADS library code. Some platforms require specific support modules in addition to the ADS library.

To learn how to create and build ADS applications, begin by building new executable files from the source of the sample programs. This acquaints you with the process and quickly points up any hidden requirements of compiling and linking on your particular platform. The sample program source includes sample make files for the current platform.

ADS Files

The ADS environment is defined by library and header files. These are available for nearly all platforms that currently support AutoCAD. The ADS library provides all of the facilities necessary to communicate with AutoLISP and AutoCAD.

ADS is distributed in the form of an object library and header files. ADS also includes a number of source and object files for sample ADS applications and a *readads.txt* file, which describes how to compile applications for a particular platform. The library and headers are installed in the *ads* directory. The directory name may be different for certain platforms; see the installation guide for your platform.

- The ADS Object Library

  The ADS object library is contained in a single file, called *ads.lib* on some platforms. The name of this file varies from platform to platform and among different compilers on the same platform. See your installation guide or the compiler instructions in the *readme* files shipped with AutoCAD.

  When you build an executable file for an ADS application, the object library name must be specified in the link instructions.

- The Header Files

  The following header files are included in ADS application source files.

  adslib.h -- establishes platform-specific definitions and includes *adscodes.h* and *ads.h*.

  adscodes.h -- contains definitions of code values that are returned by (or passed to) ADS library functions.

  ads.h -- contains the ADS library type definitions and function declarations.

  adsdlg.h -- contains proteus-related declarations for creating dialog boxes.

  ol_errno.h -- contains symbolic codes for the error values used by the AutoCAD system variable ERRNO. These codes are shown in appendix A, "Error Codes."

  adsdef.h -- establishes definitions for the ADS environment.

The *adslib.h* header file contains directives for including *adscodes.h*, *adsdef.h*, and *ads.h*; therefore, an application source file needs to contain only the following directive:

```
#include "adslib.h"
```

An ADS application doesn't need to include *ol_errno.h* unless it uses the symbolic codes defined there to handle the value of *ERRNO*. The application doesn't need to include *adsdlg.h* unless it creates dialog boxes.

Building Applications in the ADS Program Environment

In the AutoCAD Release 13 ADS program environment, ADS applications running under Windows are implemented as executables and have a file extension name *.exe*. See the instructions that accompany your compiler for detailed information on building an executable file.

To build an ADS application on the Windows or NT platform from the DOS command line, see the files *mkads.bat*, *mkads.nt*, and *readads.txt* distributed with AutoCAD.

In the ADS program environment, ADS application executables for the DOS386 platform have the file-name extension *.exp*. For instructions on building ADS applications under DOS386, see *readads.txt*.

Running ADS Applications from AutoLISP

This section describes how to load an ADS application, how to invoke its functions, and how to unload it.

Loading an Application While Using AutoCAD

To load a compiled ADS application, use the AutoLISP xload function, which is similar to the load function used for applications written in AutoLISP.

You can configure AutoCAD to load applications automatically when it starts up. Automatic loading is described later in this section.

Like load, the AutoLISP xload function requires a file name. Its calling sequence is as follows:

```
(xload filename [onfailure])
```

The function searches for the file specified by filename, loads it into memory, and executes its initialization portion immediately. Unlike *load*, *xload* does not immediately execute the entire application. For example, on the DOS386 platform, you load an ADS application called *test.exp* by invoking *xload* as follows:

```
Command:  (xload "test")
```

As the example shows, *xload* automatically appends the proper extension (if any) to the ADS application name that you specify.

If *xload* finds the application and loads it successfully, *xload* returns the application name string supplied in the function call: in the example, *xload* returns *"test"*. If it cannot load the program successfully, *xload* displays an error message, unless you specified the optional argument *onfailure*, in which case it returns the value of *onfailure*. The *onfailure* argument can be any valid string or atom; if it is a function, it is evaluated when *xload* returns.

When it loads the application, AutoLISP verifies that the version of the ADS library for the application is compatible with the current version of AutoCAD. The ADS application must be compiled and linked into an executable file before it can be run from AutoLISP.

You can load multiple ADS applications at the same time (by multiple calls to *xload*) up to a maximum of 255.

<u>The Library Search Path</u>

If you don't specify a search path, *xload* searches for the application in the directories specified by the AutoCAD library path. The AutoCAD library path comprises the following directories in the order shown:

1. The current directory
2. The directory that contains the current drawing file
3. The directories specified by the support path (see "Support Path" on page 19 in the AutoCAD Customization Guide)
4. The directory that contains the AutoCAD program files

This path is the same path that AutoCAD follows to search for menu and other support files, and that load follows to search for AutoLISP applications. Depending on the current environment, two or more of the library path directories may be the same.

If you provide *xload* with a full path name, such as *"d:/c5/test"*, *xload* does not search other directories.

Path-name formats vary, depending on the platform. For systems that separate directory names with backward slashes, the AutoCAD interpreter accepts forward slashes as a substitute. If you enter the path name with backslashes, type each backslash twice (for example, *"d:\\c5\\test"*). (Double backslashes are required for AutoLISP string values and in C string constants; they are not required in C strings, because they are stored in memory.)

<u>Listing Loaded ADS Applications</u>

To see the names of all the ADS programs currently loaded, use the AutoLISP ads function (this function accepts no arguments). The ads function, (ads) returns a list of strings. Each string is the (fully qualified) name of a loaded ADS program.

The Way ADS Works

AutoLISP accesses an ADS application according to the following sequence of events:

- AutoLISP loads the ADS application at initialization or when *(xload)* or *ads_xload()* is invoked.
- The ADS application initializes communications with AutoLISP by calling *ads_init()*.
- The ADS application indicates that it is ready to process a request from AutoLISP by calling *ads_link()* with an application result code of *RSRSLT*.
- AutoLISP returns from the *ads_link()* call with a request code of *RQXLOAD*.
- The application defines its external functions by calling *ads_defun()* once for each function.
- The application can define help for functions that will be called from the AutoCAD command line by calling *ads_setfunhelp()*.
- The application calls *ads_link()* again with a result code of *RSRSLT* (unless it has detected an error, in which case it returns *RSERR*).
- AutoLISP returns from *ads_link()* with a request code of *RQSUBR* when the user or an AutoLISP function evaluates one of the application's external functions.
- After it evaluates the external function, the application calls *ads_link()* with a result code of *RSRSLT* (or *RSERR*, if the function failed).

An ADS application is a "slave" of AutoLISP that remains inactive (in the *ads_link()* call) until AutoLISP requests that it execute an external function or perform another operation. While an ADS application is responding to an AutoLISP request, both AutoCAD and AutoLISP are inactive: they wait for requests from the ADS library functions, and cannot respond to user input.

Because *ads_link()* is called repeatedly, the way to maintain the required sequence is to call it from the top of an "infinite" dispatch loop that contains a switch statement to handle the various AutoLISP requests. The dispatch loop is typically placed in the application's *main()* function, as shown in the following application prototype:

```
/*    Prototype for an ADS application */
 #include <stdio.h>
 #include "adslib.h"
/* MAIN -- the main routine */
void
main(argc, argv)
      int argc;
      char *argv[];
{
      int stat;
      short scode = RSRSLT; /* Default result code */
      ads_init(argc, argv); /* Initialize the interface */
      for ( ;; ) {          /* Infinite loop */
            if ((stat = ads_link(scode)) < 0) {
      printf("TEMPLATE: bad status from ads_link() = %d\n", stat);
/* Can't use ads_printf() to display this message, because the
link failed*/
fflush(stdout);
                  exit(1); /* exit() only req'd for abnormal
                              termination */
            }
            scode = RSRSLT; /* Default return value */
 /* The cases in this switch check for AutoLISP request codes */
            switch (stat) {
                  case RQXLOAD:
                        scode  =  loadfuncs()  ==  GOOD  ?  RSRSLT  :
RSERR;
                        break;

                  case  RQSUBR:  /*  Usually  implemented  to  call  an
external function */
                        break;/
                  case   RQXUNLD:   /*   Usually   implemented   just   to
return RSRSLT, */
case RQSAVE: /* not often needed */
                  case RQEND:
                  case RQQUIT:
                  default: /* Return RSRSLT */
                        break;
            }
      }
}
/* LOADFUNCS -- Define external functions by calling ads_defun */
static int loadfuncs()
{
      return GOOD;
}
```

The values GOOD and BAD, which appear as return values in the code samples throughout this guide (especially in error-handling code), are not defined by the ADS library. You can define them if you want, or substitute a convention that you prefer.

A version of this example is supplied with AutoCAD as the *template.c* sample program file. Refer to it when writing your own ADS applications to ensure that your program's interface to AutoLISP is correct.

Because AutoCAD and AutoLISP are idle while an ADS application is executing, keep your application responsive. Avoid long computations that involve no interaction with the AutoCAD user. If long computations are necessary, let the user interrupt the application by pressing *[CTRL]+[C]*; an application can check for a user interrupt by calls to *ads_usrbrk()*.