A Tutorial to ADS Programming under Windows NT

1. Background

A brief comparison between the three AutoCAD supported programming environments, AutoLISP, ADS and ARX, and an introduction to ADS programming environment are provided in the Introduction to Interactive Graphical Programming in AutoCAD. In this section

2. Developing an ADS (R 13) Application Using Windows-NT and Microsoft Visual C++

In the previous *Introduction to C programming* section, the procedure for composing, compiling and running a C program was outlined. The same procedure is used here to develop an ADS application. A sample program, *sample.c*, is used to demonstrate the procedure.

(1) Open the Microsoft developer studio
Start→programs→Microsoft visual C++4.0→Microsoft developer studio

(2) Build a new project workspace
File→New→project, workspace→Application (in type frame)→✓
(in platform)→change location to "c:\temp\"→input your workspace name "sample".

(3) Add the ADS library, *winads.lib*, to the compiling and building environment
Build→settings→Link→add: "*c:\r13\win\ads\winads.lib*"
to the object/library modules

(4) Add the path of ADS files to the compiling and building environment
Tools→options→directories→enter "*c:\r13\com\ads*" and "*c:\r13\win\ads*" in the directory frame

(5) Insert user's files into the project workspace
Insert→Files into project→insert the file "*c:\temp\sample.c*"

(6) Compile and build the executable file, <*file_name*>*.exe*
Build→compile sample.c→build sample.exe
Note: The executable file, *sample.exe*, is created automatically under the directory: *"c:\temp\simple\debug\"*

3. Variables, Types, and Values Defined in ADS

ADS defines a few data types for the AutoCAD environment. It also defines a number of symbolic codes for values passed by functions (or simply for general clarity). Finally, it declares and initializes a few global variables. The definitions and declarations appear in the ADS header files.

3.1 General Types and Definitions

The types and definitions described in this section provide consistency between applications and conformity with the requirements of AutoCAD. They also contribute to an application's legibility.

3.1.1 Real Numbers

Real values in AutoCAD are always double-precision floating-point values. ADS preserves this standard by defining the special type *ads_real*, as follows:

```
typedef double ads_real;
```

Real values in an ADS application are of the type *ads_real*.

3.1.2 Points

AutoCAD points are defined as the following array type:

```
typedef ads_real ads_point[3];
```

A point always includes three values. If the point is two-dimensional, the third element of the array can be ignored; it is safest to initialize it to 0.

ADS defines the following point values:

```
#define X 0
#define Y 1
#define Z 2
```

Unlike simple data types (or point lists in AutoLISP), a point cannot be assigned with a single statement. To assign a pointer, you must copy the individual elements of the array, as shown in the following example:

```
newpt[X] = oldpt[X];
newpt[Y] = oldpt[Y];
newpt[Z] = oldpt[Z];
```

3.1.3 Transformation Matrices

The functions *ads_draggen(), ads_grvecs(), ads_nentselp(), and ads_xformss()* multiply the input vectors by the transformation matrix defined as a 4´4 array of real values.

typedef ads_real ads_matrix[4][4];

The first three columns of the matrix specify scaling and rotation. The fourth column of the matrix is a translation vector. ADS defines the symbol *T* to represent the coordinate of this vector, as follows:

#define T 3

The matrix can be expressed as follows:

$$\begin{bmatrix} M_{00} M_{01} M_{02} M_{03} \\ M_{10} M_{11} M_{12} M_{13} \\ M_{20} M_{21} M_{22} M_{23} \\ M_{00} M_{00} M_{00} M_{10} \end{bmatrix}$$

The following function initializes an identity matrix.

```
void ident_init(ads_matrix id)
{
int i, j;
 for (i=0; i<=3; i++)
  for (j=0; j<=3; j++)
   id[i][j] = 0.0;
 for (i=0; i<=3; i++)
  id[i][i] = 1.0;
}
```

The functions that pass arguments of the *ads_matrix* type treat a point as a column vector of dimension 4. The point is expressed in homogeneous coordinates, where the fourth element of the point vector is a scale factor that is normally set to 1.0. The final row of the matrix has the nominal value of [0 0 0 1]; it is ignored by the functions that pass ads_matrix arguments. In this case the following matrix multiplication results from the application of a transformation to a point.

$$\begin{bmatrix} X' \\ Y' \\ Z' \\ 1.0 \end{bmatrix} = \begin{bmatrix} M_{00} M_{01} M_{02} M_{03} \\ M_{10} M_{11} M_{12} M_{13} \\ M_{20} M_{21} M_{22} M_{23} \\ M_{00} M_{00} M_{00} M_{10} \end{bmatrix} \begin{bmatrix} X \\ Y \\ Z \\ 1.0 \end{bmatrix}$$

or,

$$X' = M_{00}X + M_{01}Y + M_{02}Z + M_{03}$$
$$Y' = M_{10}X + M_{11}Y + M_{12}Z + M_{13}$$
$$Z' = M_{20}X + M_{21}Y + M_{22}Z + M_{23}$$

3.1.4 Entity and Selection Set Names

In AutoLISP the names of entities and selection sets are pairs of long integers. ADS preserves this standard by defining such names as an array type, as follows:

typedef long ads_name[2];

As with *ads_point* variables, *ads_name* variables are always passed by reference but must be assigned element by element.

You can also copy an entity or selection set name by calling the *ads_name_set()* macro. As with *ads_point_set()* and ADS functions, the result is the second argument to the macro.

The following sample code sets the name *newname* to equal *oldname*.

```
ads_name oldname, newname;
if (ads_entnext(NULL, oldname) == RTNORM)
ads_name_set(oldname, newname);
```

3.1.5 Useful Values

ADS defines the following preprocessor directives:

```
#define TRUE 1
#define FALSE 0
#define EOS'\0' /* String termination character */
```

The PAUSE symbol, a string that contains a single backslash, is defined for the ads_command() and ads_cmd() functions, as follows:

```
#define PAUSE "\\" /* Pause in command argument list */
```

3.2 Result Buffers and Type Codes

A general-purpose result buffer (*resbuf*) structure handles AutoCAD entities and other objects.

3.2.1 struct resbuf

The following result-buffer structure, *resbuf*, is defined in conjunction with a union, *ads_u_val*, that accommodates the various AutoCAD and ADS data types, as follows:

```
union ads_u_val {
ads_real rreal;
ads_real rpoint[3];
short rint; /* Must be declared short, not int */
char *rstring;
long rlname[2];
long rlong;
struct ads_binary rbinary;
};

struct resbuf {
struct resbuf *rbnext; /* Linked list pointer */
short restype;
union ads_u_val resval;
};
```

3.2.2 Result Type Codes Defined by the ADS Library

The *restype* field of a result buffer is a short integer code that indicates which type of value is stored in the *resval* field of the buffer. For results passed to and from ADS library functions, the ADS library defines the result type codes listed in the following table.

Code	Description
RTNONE	No result value
RTREAL	Real (floating-point) value
RTPOINT	2D point (X and Y; Z == 0.0)
RTSHORT	Short (16-bit) integer
RTANG	Angle
RTSTR	String
RTENAME	Entity name
RTPICKS	Selection set name
RTORINT	Orientation
RT3DPOINT	3D point (X, Y, and Z)
RTLONG	Long (32-bit) integer
RTVOID	Void (blank) symbol
RTLB	List begin (for nested list)
RTLE	List end (for nested list)
RTDOTE	Dot (for dotted pair)
RTT	AutoLISP t (true)
RTNIL	AutoLISP nil

RTDXF0	Group code zero for DXF lists

3.3 ADS Application Request and Result Type Codes

This section lists the AutoLISP request and ADS application result codes returned by ads_link() and described in "ADS Application Request and Result Codes".

3.3.1 Request Codes

The following table shows the possible AutoLISP request codes.

Code	Description
RQXLOAD	Loading the application. Define external functions. Equivalent to the ARX message kLoadADSMsg.
RQXUNLD	Unloading the application. Equivalent to the ARX message kUnloadADSMsg.
RQSUBR	Evaluating an external function. Related to the ARX message kInvkSubrMsg.
RQSAVE	Saving the drawing (SAVE or SAVEAS). Equivalent to the ARX message kSaveMsg.
RQEND	Ending the drawing (END, NEW, or OPEN). Equivalent to the ARX message kEndMsg.
RQQUIT	Quitting the drawing¾no save (QUIT). Equivalent to the ARX message kQuitMsg.
RQCFG	Returning from CONFIG. Equivalent to the ARX message kCfgMsg.

3.3.2 Result Type Codes

The following table shows the valid application result type codes.

Code	Description
RSRSLT	Valid result
RSERR	Error in evaluation; no result

3.4 ADS Library Function Result Type Codes

The following result type codes are the status codes returned by most ADS library functions to indicate success, failure, or special conditions (such as user cancellation).

Library function result type codes

Code	Description
RTNORM	User entered a valid value
RTERROR	The function call failed
RTCAN	User entered Ctrl + C
RTREJ	AutoCAD rejected the request as invalid
RTFAIL	AutoLISP communication failed
RTKWORD	User entered a keyboard or arbitrary text

The meanings of these codes, summarized in the table, are as follows:

RTNORM	The library function succeeded.
RTERROR	The library function did not succeed; it encountered a recoverable error.

The RTERROR condition is exclusive of the following special cases:

RTCAN	The AutoCAD user entered Ctrl + C to cancel the request. This code is returned by the user-input (ads_getxxx) functions and by the following function ads_command, ads_cmd, ads_entsel, ads_nentselp, ads_nentsel, and ads_ssget .
RTREJ	AutoCAD rejected the operation as invalid. The operation request may be incorrectly formed, such as an invalid ads_entmod() call, or it simply may not be valid for the current drawing.
RTFAIL	The link with AutoLISP failed. This is a fatal error that probably means AutoLISP is no longer running correctly. If it detects this error, the application should exit. (Not all applications check for this code, because the conditions that can lead to it are likely to hang AutoCAD, anyway.)
RTKWORD	The AutoCAD user entered a keyword or arbitrary input instead of another value (such as a point). The user-input ads_getxxx() functions, as well as ads_entsel, ads_nentselp, ads_nentsel, and ads_draggen, return this result code.

4. General Utility Functions

The ADS library provides a variety of functions for examining the drawing currently loaded in the drawing editor, for example, modifying this drawing, interacting with the AutoCAD user or with AutoLISP functions, and so on. This section provides a description of several of the most commonly used utility functions. Many of the utility functions are analogous to functions built in to the AutoLISP language.

ads_command() -- This is the function for accessing AutoCAD. The argument pairs of this function either specify options or hold data, whatever the specified as a returning value. For example, a circle centered at (0,0) and passing through (3,3) can be generated by:

```
ads_command(RTSTR,"circle",RTSTR,"0,0",RTSTR,"3,3",0)
```

ads_getvar()and ads_setvar() -- These two library functions enable ADS applications to inspect and change the value of AutoCAD system variables. A string is used for specifying the variable name, and a result buffer is used for giving the type and value of this variable. For instance:

```
ads_getvar("TEXTSTYLE",&rb);
```

ads_getint(), ads_getreal(), ads_getpoint(), etc. -- The ADS user-input or ads_get*() functions are counterparts to the (get*)functions in AutoLISP. Each of these functions pauses for the user to enter data of a specified type, and passes the entered value to an argument. For example, a real number, an integer and a point can be interactively obtained from the user by:

```
ads_getreal("Enter the radius of the nut:",&radius1);
ads_getint("Specify the number of sides:",&N_Sides);
ads_getpoint("Specify the center:",center);
```

ads_entlast -- Finds the last entity in the drawing.

```
int ads_entlast(ads_name result);
```

Sets result to the name of the last main entity in the drawing database. The last entity is selected even if it is not on screen or is on a frozen layer, but a nongraphical object cannot be selected. The last entity is the most recently created entity, so ads_entlast() can be used to obtain the name of an entity that has just been added by a call to ads_command(), ads_cmd(), or ads_entmake() (a complex entity does not appear in the database until it is complete).

If ads_entlast() succeeds, it returns RTNORM; otherwise, it returns RTERROR. When ads_entlast() fails, it sets the system variable ERRNO to a value that indicates the reason for the failure.

5. Loading, Listing AND Unloading ADS Application

◊ **Loading ADS Application**

To load a compiled ADS application, use the AutoLISP (xload) function, which is analogous to the (load) function used for application written in AutoLISP. The AutoLISP (xload) function requires the file name of the compiled ADS program. There are two ways to load a application.

(1) Loading programs manually in command line, for example:
(xload"c:/temp/sample/debug/sample").

(2) Use mouse in AutoCAD drawing editor to load ADS applications, for example:
Tools→Applications→File→*.exe(in list files of type)→c:(in drives)→c:/temp/sample/debug/(in directories)→sample.exe(in File name)

◊ **Listing loaded ADS Application**

To list the names of all ADS programs that are currently loaded on screen, enter the AutoLISP function (ads) function returns a list of strings. Each string is the name of a loaded ADS program.

◊ **Unloading ADS Application**

The opposite of (xload) is (xunload). This takes the application out of the memory of AutoCAD. There is no need for the full path name, just the name of the application. For example: (xunload"sample")

6. A Sample ADS Program

In this section a sample ADS program, *sample.c*, is used to illustrate ADS functions and the way in which an ADS program communicates with AutoCAD using ADS libraries.
As a matter of fact, the communication is very straightforward; we can just send string commands to AutoCAD in an ADS program just like we would enter these commands at the AutoCAD command line, followed by coordinates and numerical values. The sample program is written using only ADS functions to avoid potential system crash. When your ADS program crashes AutoCAD will still be functional.

This sample program draws polygons, nuts, frame and squares as well as displays the modeled geometry in 2D and 3D. A step by step explanation of the program is given in this section.

6.1 ADS Communication, Variable and Environment Setup

A major part of an ADS program is used to set up the proper communication between AutoCAD and the developed ADS application, the ADS variables and the connections to external functions.

◊ Head files

The program starts by introducing four header files. Among those, *stdio.h*, *string.h* and *math.h* are normal C include files. The other head file, *adslib.h*, defines functions and constants used for communicating with AutoCAD using ADS library. This file can be found under the directory: *c:\r13\com\ads*.

◊ Declare functions

Following the included header files are function declarations. These are called *function prototypes*. The role of these function declarations is to allow the compiler to detect errors in these functions. It is always a good idea to use *function prototypes* to ensure the correctness the composed functions. Some modern compilers will warn the user if function prototypes are not present for every function called. The body of each function is specified later in the program file.

These functions are declared *static*, thus they are only accessible in the program *sample.c*, and are not visible to *other* C files. Some compilers are able to use this information to produce smaller executable files, stopping the functions being visible to unrelated C files. Large ADS applications often have many user commands. Keeping as many functions as local as possible (using the static keyword) leads to a shorter compilation and linking cycle.

◊ Elements

The definition ELEMENTS allow the number of elements in an array to be extracted and later used in the program.

◊ Command name

The *struct func_entry* associates a string with a function. The string is the command, which a user will type at the AutoCAD command line to call the function. The *func_table* of the sample ADS program consists of five entries: “polys”, “nuts”, “frame”, "squares" and “test3d”. To extend the program, *sample.c,* with new functions, you need to add the name of these functions to the contents and add your new ADS functions as new commands.

The sign, *C:*, at the beginning of each string indicates that the function is callable simply by typing the string at the command line, i.e. *C:* is equivalent to the statement: "this is a user command". AutoLISP will recognize this convention sign.

ADS functions that are callable from outside the ADS application are called "external functions". These function can be called either by a user (if defined with the C: prefix) or by another application.

◊ Main function

The main function, *main (),* starts with an *ads_init()* (ignoring argc and argv) line, which initiate the communication between ADS and AutoCAD. Following this line is an infinite loop of the program. The program monitors and accepts instructions from AutoCAD and carries out the received instructions. When a user types in a command at the AutoCAD command line (or calls one via a menu or dialog box), for example "POLYS", AutoCAD sees whether the entered line is one of its own commands, or a command of an application program.

The ADS program is a slave of the AutoCAD. The function *ads_link()* at the beginning of the for loop is always running, while our ADS program is not doing anything most of the time. The ADS program only execute its tasks when *ads_link()* returns control from AutoCAD to the ADS program with a status value. That status value that *ads_link()* returns tells us what AutoCAD is requesting the ADS program to do. In this example, there are only three things to do:

RQXLOAD: Define the ADS functions, which are known to AutoCAD and/or available to a user, by calling *func_load()*. This value will appear when AutoCAD starts up and a new drawing is loaded. *func_load()* is where one can carries out initializations required by every drawing load. For simple ADS programs initialization is not required for every drawing load.

RQSUBR: Execute one of the user defined ADS functions. This value will be returned, for example, when a user types in one of the new commands that were introduced in the ADS program.

RQXUNLD: The user defined ADS application is being unloaded (with the AutoLISP (xunload filename) function), or AutoCAD is ending. At this time "tidy up" operations, such as closing opened files, freeing allocated memory etc. needs to be carried out. Any selection sets should be released.

◊ ROXLOAD
RQXLOAD and the other constants are defined in the header files under the ADS directory of AutoCAD. By including <*adslib.h*> you have access to all the defined data types and function prototypes required to communicate with AutoCAD. The use the angle brackets < > can avoid the specification of the full directory path. The path can be specified either using the DOS SET command in the AUTOEXEC.BAT file, or through the OPTIONS menu of the C compiler. Alternatively, one can use the full pathname in quotes, which will depend on where AutoCAD was installed, e.g.

```
include "C: \R13\COM\ADS\ADSLIB.H"
```

The first thing that AutoCAD demands the ADS program is to define the functions to be added to AutoCAD through RQXLOAD. *funcload()* is called and the result of this attempt is passed to *scode*.

The function, *funcload(),* consists of a simple for loop repeatedly calling *ads_defun()* (similar to (defun) of AutoLISP). *ads_defun()* associates the index of the loaded functions with the name of the loaded functions. When AutoCAD calls the new ADS functions it will use this index to identify the function. In this sample program, the function, "C:polys", is associated with the index 0, and "C:nuts" is associated with the index 1, etc.

In *funcload()* the ADS print function, *ads_printf()*, rather than the C print function, *printf()*, is used. When we run the program under Windows the standard C *printf()* function will not print anything. If we run the program under Extended DOS *printf()* will probably print over the graphics screen, if at all. The ADS print function, *ads_printf()*, will ensure the output to be printed on the text window of AutoCAD and/or the command line of the graphics window.

ads_defun() returns a value to indicate whether the function is successfully registered in AutoCAD. An error at any point will cause *funcload()* to abandon the loading return the error message, RTERROR. If all goes well it returns RTNORM. An error may occur if you intend to use a command that has the same name as an existing AutoCAD command.

In the main function, *main()*, the result code, *scode*, recorded the success or failure of the function load process.

Suppose that one has successfully loaded all ADS functions. The next time in the infinite loop, *ads_link()* will probably return the status value, RQSUBR. This indicates that AutoCAD is calling one of the ADS functions after the user has typed in one of the ADS commands. In the ADS program the function *dofun()* is called. The function, in turn, calls *ads_getfuncode()* to find which function has been requested. In this sample program the function code is recorded in the variable *Func_Code*, and its value will be either 0 or 1, since we have only defined two functions previously. A check is made on *Func_Code* to ensure it is within the defined range recorded in the ELEMENTS.

The composed ADS function is now executed by the code:

```
Ret_Val = (*func_table[Func_Code].func)();
```

If *Func_Code* is *0*, this line of code is equivalent to

```
RetVal = polys_func();
```

If *Func_Code* is *1*, this line of code is equivalent to

```
Ret_Val = squares_func();
```

We actually called one of the functions in the function table, *func_table*, using *Func_code* to select the called function. The remainder of the *dofun()* simply returns the value of *Ret_Val*, which signals to AutoCAD that the execution of the function is completed.

◊ Summary of this section

The communication setup between the ADS program and AutoCAD as well as the way various functions are handled seem quite complicated. This is determined by the nature of the ADS programming environment. Since the ADS program is a slave of the AutoCAD program, the ADS program only takes instructions from AutoCAD and has minimum influence to the operation of AutoCAD. A fatal error in the ADS program will just crash the ADS program itself. AutoCAD and its computer model will not be influenced. This property makes ADS an ideal learning tool for interactive graphical programming. To develop a more efficient program within AutoCAD, the ARX programming environment is a better tool. An ARX function will be equivalent to a built-in AutoCAD function.

As a ADS programmer, one can ignore the complicated communications between ADS and AutoCAD, simply copy this part of the program UNCHANGED, and concentrate on the user defined ADS functions.

For the sample program, we need only to worry the user defined functions: *polys_func(), nut_func(), frame_func(), squares_func()* and *test3d_func()*. The ways that these function are called and the returned values are passed to AutoCAD are described by the previously stated procedures. You can add more ADS functions as you wish from now on without having to change a single line of code in the ADS/AutoCAD communication section. For this sample program, almost all the codes between the first line of *main()* and the last line of *dofun()* can remain the same.

6.2 Drawing Nested Polygons

The role of the sample ADS function, *poly_func()*, is to obtain the information of a group of polygons from a user interactively in AutoCAD and draw these nested polygons on the screen.

A user can start AutoCAD, load the ADS program by typing in:

(xload "c:/temp/sample/debug/sample") at the AutoCAD command line.
and then type in *polys* to run the program.

The program will prompt on the number of the sides of the polygon, the total number of the polygons, the center location of the polygons, and the size (or radius) of the outer polygon. The location and size of the polygons can be specified either by coordinates and numbers, or by mouse controlled cursor positions (click at the center point and drag the cursor to specify the desired size). The Control-C key under DOS or ESCAPE key under Windows will terminate the "polys" commands and return to AutoCAD.

The *poly_func()* Function

- ◊ Definition of a constant *TWO_PI* of type *ads_real*.
  This is an ADS compatible real number to ensure the portability of an ADS program between compilers. Usually *ads_real* is defined as a double length floating point number.
- ◊ Function definition of *polys_func()*
  The defined function returns an integer that indicates whether the function has been successfully executed.
- ◊ Specification of local variables
- ◊ A call to *ads_retvoid()*
  The call tells AutoCAD that this function returns nothing to AutoLISP. Without this call, you will get a NIL printed on the command line after you called "polys". This is not a serious problem, but is irritating to the user. AutoLISP programmers might like to know that *ads_retvoid()* can be thought of as the C equivalent of the (princ) function.
- ◊ Data input
  The number of sides of the polygon
  The user's input is acquired using ADS get integer function, *ads_getinc()*. The line prompt "Enter the number of sides of the polygon" and wait for the user to enter a value. The first returned value is recorded in the integer variable *Res*. *Res* indicates the success or failure of the function. As usual with AutoCAD, a user can abort the operation by hitting *ESCAPE* in Windows, the *ads_getint()* will then return *RTCAN* to *Res*. If a polygon side number is entered instead, *RTNORM* is returned to *Res*. The second returned parameter, the number of sides of the polygon, is recorded by another integer variable, *N_Sides*. The program checks whether *RTNORM* has been returned. This second use of *RTNORM* is to let AutoCAD know that the function ended OK, there were no errors, and it is just that the user hit *ESCAPE*.
  The number of the nested polygons
  Again *ads_getint()* is used to acquire the number of the nested polygons that the user wants to draw.
  The location of the polygons
  The location or center of the polygons is acquired by calling the ADS function, *ads_getpoint()*. The function accepts a 3D point in AutoCAD of type *ads_point*, defined as: `typedef ads_point ads~rea1[3].`
  The 3D point is specified by its x, y and z coordinates, each of which is a real number. *ads_getpoint()* has three parameters.
  The first parameter of *ads_getpoint()* is either *NULL* or a valid *ads_point*. If it is an *ads_point* then a rubber-band line is drawn from that point to the cursor controlled by the mouse. If the first parameter is NULL no rubber band line is drawn and the user sees only the cursor. The rubber band line can be used to help orient the user if a point has to be selected *relative* to another point.
  The second parameter of *ads_getpoint()* is the prompt issued to the user.
  The last parameter is the place to put the result, i.e. the point selected by the user (either by mouse or by typing), in our example *Center*.

The radius (or size) of the outer polygon

The ADS program acquire the dimension of the polygons by asking the radius of the outer polygon using ADS get distance function *ads_getdist()*. This function also has the rubber-banding option for mouse input. The Center acts as the starting point of the rubber band line, and as the user moves the mouse the other end of the rubber-band line moves with the cursor. A click on the mouse button determines the radius of the outer polygon.

Both *ads_getpoint()* and *ads_getdist()* return *RTCAN* if the user aborts the command.

A following *for* loop draws the sides of the polygons by repeatedly executing the AutoCAD command "_LINE" through the ADS command:

```
Res = ads_command(RTSTR, "_LINE"....
```

ads_command() is a very important ADS function. It can be used to execute most AutoCAD commands and pass the data needed by these commands. The command and its prompted data are provided in a list, with *RTNONE* signaling the end of the list (functioning as a return key).

In this list, *RTSTR* tells AutoCAD that the program is sending a command string: "*_LINE*", while *RT3DPOINT*, tells AutoCAD that the program is sending a 3D data point, either the start point *Start* or the end point, *End.* An empty string is sent to exit from the AutoCAD *_LINE* command, and *RTNONE* to signal the end of the list.

The *ads_command()* returns a status value: *RTNORM* if AutoCAD understood the command, or *RTERROR* if AutoCAD failed to recognize the entered command. Although this information is not used here. It is a good practice to check the returned value of every *ads_command()* call.