Result Buffers

What Problem Is The struct resbuf Designed To Address?

The problem is how to represent values of different data types that comprise an AutoCAD entity or table record, or header variable, with one mechanism.

The resbuf structure is designed to accomodate one item, or a chain of items expressed as a linked list, whose data type can be any of those available in AutoCAD.

Real

Point

Integer

String

Entity name or selection-set

Long integer

Binary data

AutoCAD entity and table records are composed of combinations values that have different data types. AutoCAD header variables, or "system variables", are each composed of one item (or an array of items) of a particular data type.

The ads_getvar() function retrieves the value of a system variable; therefore, it must return the value within a data structure capable of handling all of the different data types found in the system variable table. For example, the value retrieved from SNAPMODE will be an integer, while the value retrieved from DWGNAME will be a string.

The ads_entget() function returns a data structure that fully represents an AutoCAD entity. Since an AutoCAD entity record is composed of a variety of values of different data types (e.g., a string for the entity's layer name, an array of three reals for a defining point, etc.), ads_entget() must return a data structure that logically connects an arbitrary number of different items of different data types. This is accomplished through the use of a **linked list** of result buffer structures. ads_entget() returns a pointer to a linked list of resbuf structures that comprise the entity's defining data.

How Is The resbuf Structure Defined?

```
union ads_u_val {
   ads_real rreal;
   ads_real rpoint[3];
   short rint;
   char *rstring;
   long rlname[2];
   long rlong;
   struct ads_binary rbinary;
}


struct resbuf {
   struct resbuf *rbnext;
   short restype;
   union ads_u_val resval:
}
```

resval

The resval union is composed of the different data types that can be passed between AutoCAD and ADS.

The value of an instance of a resbuf is in one of the members of the resval union.

restype

The value of the restype member indicates which data type will be found in the resval union for an instance of a resbuf. The value of restype will be either a **symbolic code** from *adscodes.h* or a **DXF group code**.

Symbolic codes

See adscodes.h

DXF group codes

See Appendix B of the *Advanced Tools* manual for AutoCAD Release 12.

The data type for restype is a short integer.

rbnext

A pointer to the next resbuf structure in a linked list. This value should be NULL for a single result buffer, or for the last result buffer in a linked list.

The rbnext member provides a way in which to chain multiple resbuf structures together.

The result buffers can be created by static or dynamic means.

Automatic or static

Declare a variable of type struct resbuf. Assign appropriate values to the restype, rbnext and resval members. Automatic or static creation does not automatically set the rbnext member to NULL.

```
struct resbuf rb;
rb.restype = RTSHORT;
rb.rbnext = NULL;
rb.resval.rint = 0;
```

Dynamic

Declare a variable to be of type pointer to resbuf.

Allocate a resbuf by calling the ADS library function ads_newrb(). Give ads_newrb() an argument corresponding to the value you with to place in the restype member. ads_newrb() returns a pointer to a dynamically allocated resbuf whose restype member is set to its argument, and whose rbnext member is set to NULL.

Assign the appropriate value to the resval member.

Free the resources dynamically allocated for the resbuf by calling ads_relrb() before the function returns.

```
struct resbuf *rb;
rb = ads_newrb(RTSHORT);
rb->resval.rint = 0;
 ...
ads_relrb(rb);
}
```

EXERCISE

Exercise: Write, Compile, Link, Load and Execute sysvar1.c

New functions required for the exercise:

ads_getvar()

Write a function named sysvar1() that retrieves the value of the system variable SNAPMODE by calling the ADS library function ads_getvar(), and prints the value to the command prompt area.

Declare an automatic variable of type struct resbuf.

Pass the address of the resbuf as an argument to ads_getvar().

Call ads_printf() to print the value of the resval member of the resbuf.

```
struct resbuf rb;
ads_getvar("SNAPMODE", &rb);
ads_printf("%d", rb.resval.rint);
}
```

EXERCISE

Exercise: Write, Compile, Link, Load and Execute sysvar2.c

New functions required for the exercise: **ads_setvar(), ads_newrb(), ads_relrb()**

Write a function named sysvar2() that sets the value of the system variable USERS1 by calling the ADS library function ads_setvar(). USERS1 stores a string.

Declare a variable of type pointer to struct resbuf.

Declare a variable of type pointer to char and assign it a value. Use any string value that you wish.

Allocate a new result buffer with a restype value of RTSTR and assign its address to the pointer variable. Use the ADS library function ads_newrb().

Assign the variable of type pointer to char to the resval.rstring member of the resbuf.

Pass the resbuf pointer as an argument to ads_setvar().

Set the resval.rstring member of the resbuf to NULL.

Free the resbuf by calling ads_relrb().

```
struct resbuf *rb;
char *s = "User-defined value.";
rb = ads_newrb(RTSTR);
rb->resval.rstring = s;
ads_setvar("USERS1", rb);
rb->resval.rstring = NULL;
ads_relrb(rb);
```

Interactive String Acquisition

The ADS library function ads_getstring()

ads_getstring() allows an application to prompt the user for a string value. It takes three arguments: a Boolean argument which, if true, allows the inclusion of spaces in the string; a pointer-to-char prompt for the user; and a pointer-to-char result variable into which the value entered by the user is copied.

The result argument must be pre-allocated. The maximum length of the string returned by ads_getstring() is 132 characters, so an array of 133 characters will always be sufficient.

ads_getstring() returns RTNORM if successful, RTERROR on error, or RTCAN if the user cancels the request with Ctrl-C.

EXERCISE

Exercise: Modify sysvar2.c

New functions required for the exercise:

ads_getstring()

Write a function named sysvar3() that sets the value of the system variable USERS1 by calling the ADS library function ads_setvar().

Prompt the user for the new value of USERS1 by calling the ADS library function ads_getstring().

```
struct resbuf *rb;
char s[133];

if (ads_getstring(1, "\nNew value for USERS1: ", s) != RTNORM)
{
    ads_fail("Error getting string.");
    return RTERROR;
}
if ((rb = ads_newrb(RTSTR)) == NULL) {
    ads_fail("Error allocating result buffer.");
    return RTERROR;
}
rb->resval.rstring = s;
if (ads_setvar("USERS1", rb) != RTNORM) {
    ads_fail("Error setting system variable.");
    ads_relrb(rb);
    return RTERROR;
}
rb->resval.rstring = NULL;
ads_relrb(rb);
```

Result Buffer and String Allocation

Result buffers and text strings may be allocated as static or automatic variables, or allocated dynamically through the ADS function library. In the latter case, the ADS library functions will call the standard C library function malloc() to allocate memory from the system. The ADS application is responsible for releasing the memory, typically by calling the ADS library function ads_relrb() for result buffers and the standard C library function free() for text strings.

Case 1

String: static/automatic and Resbuf: static/automatic

In this case, the ADS application should not explicitly free either variable. The system will manage it automatically.

```
struct resbuf rb;
char *s = "Hello, world."
rb.restype = RTSTR;
rb.rbnext = NULL;
rb.resval.rstring = s;
ads_setvar("USERS1", &rb);
```

Case 2

String: static/automatic and Resbuf: dynamic

In this case, the ADS application must explicitly free the result buffer, but not the string.

The application should set the resval.rstring member of the resbuf to NULL prior to releasing the resbuf with a call to ads_relrb(). This is because ads_relrb() will attempt to free the resval.rstring member of the resbuf if it is not NULL.

```
struct resbuf *rb;
char *s = "Hello, world."
rb = ads_newrb(RTSTR);
rb->resval.rstring = s;
ads_setvar("USERS1", rb);
rb->resval.rstring = NULL;
ads_relrb(rb);
```

Case 3

String: dynamic and Resbuf: static/automatic

In this case, the ADS application must explicitly free the string, but not the result buffer.

```
struct resbuf rb;
char *s;
s = malloc( (strlen("Hello, world.") + 1) * sizeof (char) );
strcpy(s, "Hello, world.");
rb.restype = RTSTR;
rb.rbnext = NULL;
rb.resval.rstring = s;
ads_setvar("USERS1", &rb);
free(s);
```

Case 4

String: dynamic and Resbuf: dynamic

In this case, the ADS application must explicitly free both the string and the result buffer. A call to ads_relrb() will free both a result buffer and a dynamically-allocated string which is assigned to the resval.rstring member of the result buffer.

```
struct resbuf *rb;
char *s;
s = malloc( (strlen("Hello, world.") + 1) * sizeof (char) );
strcpy(s, "Hello, world.");
rb = ads_newrb(RTSTR);
rb->resval.rstring = s;
ads_setvar("USERS1", rb);
ads_relrb(rb);
```

If the string in the resval.rstring member is explicitly released by a call to free(), then resval.rstring must be set to NULL prior to a call to ads_relrb(). This prevents ads_relrb() from attempting to free the string a second time.

```
struct resbuf *rb;
char *s;
s = malloc( (strlen("Hello, world.") + 1) * sizeof (char) );
strcpy(s, "Hello, world.");
rb = ads_newrb(RTSTR);
rb->resval.rstring = s;
ads_setvar("USERS1", rb);
free(s);
rb->resval.rstring = NULL;
ads_relrb(rb);
```

A dynamically-allocated result buffer can be recycled. (So can one allocated static or automatic, for that matter.)

```
struct resbuf *rb;
char *s;
s = malloc( (strlen("Hello, world.") + 1) * sizeof (char) );
strcpy(s, "Hello, world.");
rb = ads_newrb(RTSTR);
rb->resval.rstring = s;
ads_setvar("USERS1", rb);
free(s);
s = malloc( (strlen("What's new?") + 1) * sizeof (char) );
strcpy(s, "What's new?");
rb->resval.rstring = s;
ads_setvar("USERS2", rb);
ads_relrb(rb);
```

Linked Lists of Result Buffers

What are they used for?

Linked lists of result buffers are pervasive throughout ADS applications. These data structures are used to represent:

Arguments passed from the AutoLISP environment to ADS

AutoCAD entity records

AutoCAD table records or "named objects"

Arguments to and return values from several ADS library functions

How do you create a linked list of result buffers?

Either manually, by allocating two or more instances of struct resbuf and chaining them together yourself by modifying the rbnext member values; or, by calling the ADS library function ads_buildlist().

ads_buildlist() returns a pointer to a linked list of result buffers. It is the ADS application's responsiblity to explicitly release this linked list by calling ads_relrb() and passing it a pointer to the first resbuf in the chain. ads_relrb() will walk down the linked list and free each result buffer in turn, until it finds a resbuf whose rbnext member is set to NULL. A NULL value in rbnext indicates the end of the linked list.

ads_buildlist() takes pairs of values as arguments. The first value in each pair is either a symbolic code from *ads.h* or a DXF group code, indicating the data type of the second value. The last argument must be either zero or the symbolic code RTNONE (note that this should be a single argument rather than a pair).Manual creation (the resbufs could be allocated as automatic or static variables as in the example, or dynamically with ads_newrb()).

```
struct resbuf rb1, rb2, rb3;
char *s = "An int, a real and a string in a linked list.";
rb1.restype = RTSHORT;
rb2.restype = RTREAL;
rb3.restype = RTSTR;
rb1.resval.rint = 1;
rb2.resval.rreal = 2.0;
rb3.resval.rstring = s;
rb1.rbnext = &rb2;
rb2.rbnext = &rb3;
rb3.rbnext = NULL;
```

Creation using ads_buildlist

```
struct resbuf *rb;
rb = ads_buildlist(RTSHORT, 1, RTREAL, 2.0, RTSTR, "An int, a real and a string in a
linked list.", RTNONE);
ads_relrb(rb);
}
```

EXERCISE

Exercise: Write, Compile, Link and Execute args.c

New functions required for the exercise:

ads_getargs()

Returns a pointer to the head of a linked list of result buffers that constitute the arguments passed in from AutoLISP. ads_getargs() cleans up its own memory allocation, so the ADS application should **not** free the linked list returned by the function.

Write a function named args() that retrieves the arguments passed in from AutoLISP.

Declare an automatic variable of type pointer to struct resbuf.

Assign the pointer to the value returned by a call to ads_getargs().

Test whether the pointer is NULL. If not, determine the data type of the value in the resbuf by checking the restype member. Call ads_printf() with an appropriate format string to print the value of the member of the resval union.

```
struct resbuf *rb;
rb = ads_getargs();
if (rb != NULL) {
   switch(rb->restype) {
   case RTSHORT:
      ads_printf("\n%d", rb->resval.rint);
      break;
   case RTREAL:
      ads_printf("\n%f", rb->resval.rreal);
      break;
   case RTSTR:
      ads_printf("\n%s", rb->resval.rstring);
      break;
   default:
      ads_printf("\nNot an integer, real or string.");
      break;
   }
```

After loading the ADS program into AutoCAD, call the ADS-defined function with a number of different argument lists.

```
(args 0)

(args 1.0)

(args "Hello, world.")

(args (list 1 2 3))
```

EXERCISE

Exercise: Add Tests For Multiple Arguments

Add code to args.c that reads multiple arguments passed in from AutoLISP.

Use a while loop to continue to process result buffers in the linked list until the end of the list is reached.

```
struct resbuf *rb;
rb = ads_getargs();
while(rb != NULL) {
   switch(rb->restype) {
   case RTSHORT:
      ads_printf("\n%d", rb->resval.rint);
      break;
   case RTREAL:
      ads_printf("\n%f", rb->resval.rreal);
      break;
   case RTSTR:
      ads_printf("\n%s", rb->resval.rstring);
      break;
   default:
      ads_printf("\nNot an integer, real or string.");
      break;
       }
   rb = rb->rbnext;
}
```

After loading the ADS program into AutoCAD, call the ADS-defined function with a number of different argument lists.

(args 0 1.0 "Hello, world.")

(args 1 2 3)

(args "a" "b" "c")

(args nil nil nil)

EXERCISE

Exercise: Write, Compile, Link and Execute *mult.c*

Write a function that requires two arguments. Multiply the two arguments together and return the value to AutoLISP.

New functions required for the exercise: **ads_retreal()**

Return a real number to AutoLISP.

The arguments must be real numbers.

```
struct resbuf *rb;
ads_real a, b;
rb = ads_getargs();
if (rb == NULL) {
    ads_fail("Expected two arguments.");
    return RTERROR;
}
if (rb->restype != RTREAL) {
    ads_fail("Expected a real number.");
    return RTERROR;
}
a = rb->resval.rreal;
rb = rb->rbnext;

if (rb == NULL) {
    ads_fail("Expected two arguments.");
    return RTERROR;
}
if (rb->restype != RTREAL) {
    ads_fail("Expected a real number.");
    return RTERROR;
}
b = rb->resval.rreal;
rb = rb->rbnext;
if (rb == NULL) {
    ads_fail("Too many arguments.");
    return RTERROR;
}
ads_retreal(a * b);
return RTNORM;
}
```

After loading the ADS program into AutoCAD, call the ADS-defined function with a number of different argument lists.

(args 1.0 2.0)

(args 1 2)

(args)

(args 1.0 2.0 3.0)

Entity Records

Creating New Entities

Group codes and defining data

The defining data for a red Circle entity with a center point of 5,5,0 and a radius of 1.0 on the current layer consists of the following DXF group codes:

0 for entity type (char *)

62 for color (int)

10 for center point (ads_point)

40 for radius (ads_real)

A call to ads_buildlist() will create a linked list of result buffers with the appropriate restype and resval union values.

Since 0 is an acceptable value for terminating the argument list to ads_buildlist(), the special symbolic code RTDXF0 must be used for the group 0 code of an entity's defining data.

The ADS library function ads_entmake()

The ADS library function ads_entmake() takes a single argument: a pointer to the first result buffer in a linked list of result buffers. It attempts to create a new entity in the AutoCAD database according to the defining data within the linked list. It returns RTNORM if it succeeds; otherwise, it returns a different result code, usually RTERROR or RTREJ.

ads_entmake() can return RTNORM even if it only partially succeeds in modifying an existing entity.

```
struct resbuf *newent;

newent = ads_buildlist(RTDXF0, "CIRCLE",
  62, 1,
  10, center,
  40, 1.0,
  RTNONE);
ads_entmake(newent);
ads_relrb(newent);
```

With error checking added, the code might look like this.

```
struct resbuf *newent;
newent = ads_buildlist(RTDXF0, "CIRCLE",
 62, 1,
 10, center,
 40, 1.0,
  RTNONE);
if (newent == NULL) {
   ads_fail("Error making Circle entity list.");
   return RTERROR;
}
if (ads_entmake(newent) != RTNORM) {
   ads_fail("Error making Circle entity.");
   ads_relrb(newent);
   return RTERROR;
}
ads_relrb(newent);
return RTNORM;
```

When creating several entities of the same type with different defining points, it makes sense to avoid the overhead of dynamic memory allocation by creating a single linked list of result buffers and changing the appropriate member values priort to each call to ads_entmake().

```
struct resbuf *newent, *rb;
newent = ads_buildlist(RTDXF0, "CIRCLE",
  62, 1,
  10, center,
  40, 1.0,
  RTNONE);
ads_entmake(newent);
/* Change radius value in existing linked list
   and use it to make another Circle */
rb = assoc_rb(newent, 40);
rb->resval.rreal = 1.5;
ads_entmake(newent);
/* Change radius value in existing linked list
   and use it to make another Circle */
rb->resval.rreal = 2.0;
ads_entmake(newent);
ads_relrb(newent);
}
```

EXERCISE

Handout: emake.c

Entity records are represented in ADS as linked lists of result buffers.

For each result buffer, the restype member contains a DXF group code that describes the type of data held in the resval union and the context of the data; that is, what the data means in terms of the entity's definition. This context is necessary because the same DXF group code can mean different things for different entities.

Ask your instructor to point out the appropriate appendix in the AutoCAD documentation that lists DXF group codes for each entity.

Exercise: Write, Compile, Link and Execute *emake.c*

Write a function creates a Line entity.

Create a linked list of result buffers to hold the Line's defining data. Add the Line to the AutoCAD database by calling the ADS library function ads_entmake() and passing the linked list as its argument.

New functions required for the exercise

ads_entmake();

Adds a new entity record to or modifies an existing record in the AutoCAD database.

It requires a single argument: a linked list of result buffers that hold the defining data for the entity.

Add code to *emake.c* to do the following things.

Declare a varaible of type pointer to struct resbuf.

Declare two variables of type ads_point named start and end. Initialize the variables with the values {1.0, 1.0, 0.0} and {5.0, 5.0, 0.0}.

Create a linked list of the entity's defining data by calling the ADS library function ads_buildlist() with the appropriate arguments. Assign the value returned by ads_buildlist() to the pointer to struct resbuf variable.

The DXF group codes and values necessary to create the Line entity are as follows:

RTDXF0 "LINE"

6 "BYLAYER"

8 "0"

10 start

11 end

62 256

Without error checking (which you must always include for ADS library functions that return result codes indicating success or failure), the code to create the Line entity might look like this.

```
struct resbuf *rb;
ads_point start = {1.0, 1.0, 0.0};
ads_point end = {5.0, 5.0, 0.0};
rb =ads_buildlist(RTDXF0, "LINE", 8, "0", 6, "BYLAYER", 10,
start, 11, end, 62, 256, RTNONE);
ads_entmake(rb);
ads_relrb(rb);
```

Reading and Modifying Existing Entities

Reading defining data for existing entities

The ADS library function ads_entget()

ads_entget() returns a linked list of result buffers that hold the defining data for an entity.

The entity is specified as an argument to ads_entget(). This argument is the entity name of the specified entity, and its data type is ads_name.

One of several different functions can be used to obtain an entity's name.

ads_entnext();

ads_entlast();

ads_entsel();

The ADS library function ads_entnext();

ads_entnext() retrieves an entity name from the database.

It takes two arguments: an entity name, and a result argument of type ads_name.

The first argument can be NULL. If it is, ads_entnext() places the name of the first entity in the database into the second argument. If there are no entities in the database, ads_entnext() returns RTERROR.

If the first argument is a valid entity name, ads_entnext() places the name of the entity that follows it in the database into the second argument. If no entities follow the entity name supplied as the first argument, ads_entnext() returns RTERROR.

}

EXERCISE

Handout: entread.c

Exercise: Write, Compile, Link and Execute *entread.c*

Write a function that retrieves the name of the first entity in the database and prints the value of the restype member from each result buffer in the entity's linked list.

New functions required for the exercise

ads_entnext();

ads_entget();

Add code to *entread.c* to do the following things.

Declare a variable of type ads_name.

Declare two variables of type pointer to struct resbuf.

Use ads_entnext() to the name of the first entity in the database.

If an entity is retrieved by ads_entnext(), get its defining data by calling ads_entget() and assigning its return value to one of the pointer variables.

Loop through the linked list and print the value of each restype member.

```
struct resbuf *elist, *rb;
ads_name ename;
if (ads_entnext(NULL, ename) != RTNORM) {
    ads_printf("No entities in database.");
    return RTNORM;
}
elist = ads_entget(ename);
if (elist == NULL) {
    ads_fail("Error getting entity data.");
    return RTERROR;
}
rb = elist;
while(rb != NULL) {
    ads_printf("\nrestype: %d", rb->restype);
    rb = rb->rbnext;
}
ads_relrb(elist);
return RTNORM;
```

Modifying defining data for existing entities

The ADS library function ads_entmod()

ads_entmod() modifies an existing entity record based on changes made to its defining data.

It takes a single argument: a pointer to a linked list of result buffers.

The first result buffer in the linked list must contain the entity's name in the resval union. Its restype value must be -1.

AutoCAD will update the entity's record based on changes made to the individual result buffer values in the entity's linked list.

Get the entity's record with ads_entget(), modify one or more of the result buffer values, then apply the changes to the database by calling ads_entmod() with the modified linked list as its argument.

Looping through a linked list for a restype value

Use a function like assoc_rb(), defined below, to find a result buffer within a linked list with a restype member of a certain value.

This sort of function is useful when you want to modify a result buffer within a linked list for an existing entity and pass the result back to ads_entmod().

assoc_rb() takes two arguments: a pointer to a linked list of result buffers, and an integer. It will search through each resbuf in the linked list, returning a pointer to the buffer whose restype member value matches its integer argument, or NULL if no match is found.

Define the assoc_rb() function

/* Search a linked list of result buffers and return an item associated with the specified group code */

```
struct resbuf
*assoc_rb(struct resbuf *rb, int group)
{
   while((rb != NULL) && (rb->restype != group))
      rb = rb->rbnext;
   return rb;
}
```

assoc_rb() is defined within the utility library *util_ads.lib* in the course *student* directory. To use it, include the header file *util_ads.h* in your application source and link *util_ads.lib* with the project. Source for the library is supplied as *util_ads.c*.

}

EXERCISE

Handout: entchg.c

Exercise: Write, Compile, Link and Execute *entchg.c*

Write a function that retrieves the name of the first entity in the database and prints the value of the restype member from each result buffer in the entity's linked list.

New functions required for the exercise

ads_entmod();

assoc_rb(); - a utility function not in the ADS library

Add code to *entchg.c* to do the following things.

Declare a variable of type ads_name.

Declare two variables of type pointer to struct resbuf.

Use ads_entnext() to retrieve the name of the first entity in the database.

If an entity is retrieved by ads_entnext(), get its defining data by calling ads_entget() and assigning its return value to one of the pointer variables.

Use assoc_rb() to get a pointer to the group 0 resbuf of the entity's linked list.

If the entity's type is "LINE", find and modify the group 10 resbuf and change the coordinates of the starting point of the Line entity to 0,0,0.

Apply the changes to AutoCAD by calling ads_entmod().

```
#include <string.h>
extern struct resbuf * assoc_rb(struct resbuf *, int);
struct resbuf *elist, *rb;
ads_name ename;
int i;
if (ads_entnext(NULL, ename) != RTNORM){
    ads_printf("No entities in database.");
    return RTNORM;
}
elist = ads_entget(ename);
if (elist == NULL) {
    ads_fail("Error getting entity data.");
    return RTERROR;
}
rb = assoc_rb(elist, 0);
if (rb == NULL) {
    ads_fail("Error in entity's data.");
    ads_relrb(elist);
    return RTERROR;
}
if (strcmp("LINE", rb->resval.rstring) !=  0) {
    ads_printf("\nNot a Line entity.");
    ads_relrb(elist);
    return RTNORM;
}
rb = assoc_rb(elist, 10);
if (rb == NULL) {
    ads_fail("Error in entity's data.");
    ads_relrb(elist);
    return RTERROR;
}
for(i = 0; i < 3; i++) {
    rb->resval.rpoint[i] = 0.0;
}
if (ads_entmod(elist) != RTNORM) {
    ads_fail("Error modifying entity in database.");
    ads_relrb(elist);
    return RTERROR;
}
ads_relrb(elist);
return RTNORM;
}
```

EXERCISE

Exercise: Change All Line Entities In The Drawing

Add code to *entchg.c* that modifies every Line entity in the drawing.

Use a while loop to continue to process linked lists returned in the result argument of ads_entnext() until the end of the database is reached..

```
#include <string.h>
extern struct resbuf * assoc_rb(struct resbuf *, int);
struct resbuf *elist, *rb;
ads_name ename;
int i, stat;
stat = ads_entnext(NULL, ename);
while (stat == RTNORM) {
     elist = ads_entget(ename);
   if (elist == NULL) {
      ads_fail("Error getting entity data.");
      return RTERROR;
       }
     rb = assoc_rb(elist, 0);
   if (rb == NULL) {
      ads_fail("Error in entity's data.");
      ads_relrb(elist);
      return RTERROR;
       }
   if (strcmp("LINE", rb->resval.rstring) !=  0) {
      ads_printf("\nNot a Line entity.");
      ads_relrb(elist);
      return RTNORM;
       }
     rb = assoc_rb(elist, 10);
   if (rb == NULL) {
      ads_fail("Error in entity's data.");
      ads_relrb(elist);
      return RTERROR;
       }
   for(i = 0; i < 3; i++) {
      rb->resval.rpoint[i] = 0.0;
       }
   if (ads_entmod(elist) != RTNORM) {
      ads_fail("Error modifying entity in database.");
      ads_relrb(elist);
      return RTERROR;
       }
   stat = ads_entnext(ename, ename);
}
ads_relrb(elist);
return RTNORM;
}
```

EXERCISE

Handout: entpick.c

Interactive entity selection

The ADS library function ads_entsel() allows an ADS application to prompt the user for an interactive entity selection. ads_entsel() permits the user to select a single entity, and stores both the name of the entity selected and the point at which it was picked.

ads_entsel() takes three arguments: a pointer to char that will be displayed as a prompt for the user, a variable of type ads_name, and a variable of type ads_point. The latter two are result arguments. The entity's name and its pick point will be stored in them.

ads_entsel() returns RTNORM if successful. It returns RTERROR if it fails (for example, if the user picks on a blank part of the display), or RTCAN if the user cancels the request with a Ctrl-C.

Exercise: Write, Compile, Link and Execute *entpick.c*

Write a function that prompts the user to select a Line entity. Change the start point of the Line entity to the point at which the user selects the Line.

New functions required for the exercise

ads_entsel();

Add code to *entpick.c* to do the following things.

Declare two variables of type pointer to struct resbuf.

Declare a variable of type ads_name, named ename.

Declare a variable of type ads_point, named epoint.

Use ads_entsel() to prompt the user to select a Line entity.

If the user selects an entity, get its record with ads_entget().

Use assoc_rb() to find the resbuf whose restype is 0. Do a string comparison to determine whether this is a "LINE" entity.

If it is a Line entity, find the resbuf whose restype is 10 and update the resval.rpoint member to the values placed in the ads_point result argument by ads_entsel().

Update the Line entity with a call to ads_entmod().

```
#include <string.h>
#include "util_ads.h"
extern struct resbuf * assoc_rb(struct resbuf *, int);
struct resbuf *elist, *rb;
ads_name ename;
ads_point epoint;
int i;
if (ads_entsel("\nSelect line: ", ename, epoint) != RTNORM) {
    ads_printf("No entity selected.");
    return RTNORM;
}
elist = ads_entget(ename);
if (elist == NULL) {
    ads_fail("Error getting entity data.");
    return RTERROR;
}
rb = assoc_rb(elist, 0);
if (rb == NULL) {
    ads_fail("Error in entity's data.");
    ads_relrb(elist);
    return RTERROR;
}
if (strcmp("LINE", rb->resval.rstring) !=  0) {
    ads_printf("\nNot a Line entity.");
    ads_relrb(elist);
    return RTNORM;
}
rb = assoc_rb(elist, 10);
if (rb == NULL) {
    ads_fail("Error in entity's data.");
    ads_relrb(elist);
    return RTERROR;
}
for(i = 0; i < 3; i++) {
    rb->resval.rpoint[i] = epoint[i];
}
if (ads_entmod(elist) != RTNORM) {
    ads_fail("Error modifying entity in database.");
    ads_relrb(elist);
    return RTERROR;
}
ads_relrb(elist);
return RTNORM;
```

Interactive Point Acquisition

The ADS library function ads_getpoint()

ads_getpoint() allows an ADS application to interactively prompt the user for a point specification. The function's syntax and usage is similar to that of ads_getstring().

The user can respond with any of AutoCAD's point specification methods to a request for a point from ads_getstring().

ads_getpoint() takes three arguments: a base point of type ads_point (which can be NULL), a pointer-to-char prompt for the user, and a result variable of type ads_point into which the point selected by the user will be copied.

ads_getpoint() returns RTNORM, RTERROR or RTCAN. A prior call to the ADS library function ads_initget() can enable return values of RTNONE or RTKWORD.

The ADS library includes a variety of functions that prompt the user for interactive data entry.

String	ads_getstring()
Point	ads_getpoint(), ads_getcorner()
Angle	ads_getangle(), ads_getorient()
Distance	ads_getdist()
Filename	ads_getfiled()
Integer	ads_getint()
Keyword	ads_getkword()
Real	ads_getreal()

Initializing An ads_getxxx() Function Call

In some cases, an ADS application may need to expand the user's choice of options during an ads_getxxx function call and permit the entry of **keywords**. In others, it may need to restrict the user's options; for example, disallow negative or zero input during an ads_getint() call. The ADS library function ads_initget() allows an application to initialize the next ads_getxxx function call, expanding its options to include keywords, or restricting them to disallow certain values.

ads_initget() takes two arguments: a bit-coded integer, and a pointer-to-char keyword string.

See the *ADS Programmer's Reference* for a description of the various bit-code values.

The keyword string is a space-delimited string, where each keyword has a unique sequence of uppercase letters. This allows to user to identify each keyword with a minimum number of unique keystrokes.

The keyword string is optional, i.e., it can be a NULL argument.

EXERCISE

Exercise: Write, Compile, Link and Execute *entpick2.c*

Write a function that prompts the user to select a Line entity. Prompt the user separately for the new start point of the Line, and update the entity. Disallow null input on the user's part.

New functions required for the exercise: **ads_initget(), ads_getpoint()**

Use the finished code from *entpick.c* as a base (it's supplied for you in the file *entpick2.c*). Add code to *entpick2.c* to do the following things.

Declare a variable of type ads_point, named start.

Use ads_initget() to disallow null input on the next ads_getpoint() call.

Get a point from the user with a call to ads_getpoint(). Use the existing start point of the Line entity (group code 10 in the restype member of the resbuf) as the base point of the call to ads_getpoint().

Update the Line entity's start point with the point specified by the user.

```
#include <string.h>
#include "util_ads.h"
extern struct resbuf * assoc_rb(struct resbuf *, int);
struct resbuf *elist, *rb;
ads_name ename;
ads_point epoint, start;
int i;
if (ads_entsel("\nSelect line: ", ename, epoint) != RTNORM) {
    ads_printf("No entity selected.");
    return RTNORM;
}
elist = ads_entget(ename);
if (elist == NULL) {
    ads_fail("Error getting entity data.");
    return RTERROR;
}
rb = assoc_rb(elist, 0);
if (rb == NULL) {
    ads_fail("Error in entity's data.");
    ads_relrb(elist);
    return RTERROR;
}
if (strcmp("LINE", rb->resval.rstring) !=  0) {
    ads_printf("\nNot a Line entity.");
    ads_relrb(elist);
    return RTNORM;
}
rb = assoc_rb(elist, 10);
if (rb == NULL) {
    ads_fail("Error in entity's data.");
    ads_relrb(elist);
    return RTERROR;
}
if (ads_initget(RSG_NONULL, NULL) != RTNORM) {
    ads_fail("Error initializing point acquisition.");
    ads_relrb(elist);
    return RTERROR;
}
if (ads_getpoint(rb->resval.rpoint, "\nNew start point: ",
start) != RTNORM) {
    ads_fail("Error getting point.");
    ads_relrb(elist);
    return RTERROR;
}
for(i = 0; i < 3; i++) {
    rb->resval.rpoint[i] = start[i];
}
if (ads_entmod(elist) != RTNORM) {
    ads_fail("Error modifying entity in database.");
    ads_relrb(elist);
    return RTERROR;
}
ads_relrb(elist);
return RTNORM;
```

Selection-sets

What is a selection-set?

A collection of AutoCAD entity names. Selection-sets can be of any size. Each is stored in a temporary file. Up to 128 selection-sets can be active at any one time, but this number tends to be lower in practice due to system resource limits.

The ADS library function ads_ssget() creates a new selection-set.

For every successful call to ads_ssget(), a matching call must be made to the ADS library function ads_ssfree(). This call releases the system resources allocated to the selection-set.

Many AutoCAD editing commands and special ADS library functions can act upon a selection-set, applying an operation uniformly to all the entities within the set.

How can selection-sets be specified?

Interactively.

The user can specify objects to add to a selection-set by choosing from a variety of keywords and specifying points on the screen that fall on or around AutoCAD entities.

With **filters**.

AutoCAD can apply filters to properties of entities gathered interactively, or to all the entities in the database. Only those entities that match the filter specification will be included in the selection-set.

With **Boolean expressions** and **relational tests**.

Interactive selection and filters can be combined with Boolean expressions such as AND, OR and NOT, and with relational tests on the values of entity properties, tests such as "greater than" and "equal to".

The ADS library function ads_ssget()

The ADS library function ads_ssget() takes 5 (five) required arguments.

A string describing the selection mode, or NULL if interactive.

An ads_point describing the first point of a window, or a result buffer list describing a polygon or fence, or NULL.

An ads_point describing the second point of a window, or NULL.

A result buffer list describing the filters through which candidate entities must pass, or NULL.

An ads_name which will be set to the resulting selection-set.

ads_ssget() returns RTNORM if it successfully creates a new selection-set. It returns RTERROR if no selection-set was created, or if the filter list contains improper group codes. Note that the first case is not really an error. ads_ssget() sets the system variable ERRNO to a value that indicates the reason it returned RTERROR.

Interactive selection

This sample code implements user-interactive selection-set acquisition.

```
ads_name ss = { 0L, 0L };
if (ads_ssget(NULL, NULL, NULL, NULL, ss) != RTNORM) {
   ads_printf("\nNo selection-set was created.");
   return RTNORM;
}
ads_printf("\nSelection-set was created.");
ads_ssfree(ss);
```

Interactive selection with filters

This sample code implements user-interactive selection-set acquisition with filters. Only Line entities with an entity color of red (or 1) chosen by the user will be selected.

```
ads_name ss = { 0L, 0L };
struct resbuf *filterlist;
if ((filterlist = ads_buildlist(RTDXF0, "LINE", 62, 1, RTNONE))
== NULL) {
    ads_fail("Error creating filter list.");
    return RTERROR;
}
if (ads_ssget(NULL, NULL, NULL, filterlist, ss) != RTNORM) {
    ads_printf("\nNo selection-set was created.");
    ads_relrb(filterlist);
    return RTNORM;
}
ads_printf("\nSelection-set was created.");
ads_relrb(filterlist);
ads_ssfree(ss);
```

Interactive selection with filters using Boolean operations and relational tests

This sample code implements user-interactive selection-set acquisition with filters, Boolean operations and relational tests. Only Line entities with an entity color of red (or 1) and Circle entities with a radius value greater than or equal to 1.0 chosen by the user will be selected.

```
ads_name ss = { 0L, 0L };
struct resbuf *filterlist;
if ((filterlist = ads_buildlist(-4, "<OR", -4, "<AND", RTDXF0,
"LINE", 62, 1, -4, "AND>", -4, "<AND", RTDXF0, "CIRCLE", -4,
">=", 40, 1.0, -4, "AND>", -4, "OR>", RTNONE)) == NULL) {
    ads_fail("Error creating filter list.");
    return RTERROR;
}
if (ads_ssget(NULL, NULL, NULL, filterlist, ss) != RTNORM) {
    ads_printf("\nNo selection-set was created.");
    ads_relrb(filterlist);
    return RTNORM;
}
ads_printf("\nSelection-set was created.");
ads_relrb(filterlist);
ads_ssfree(ss);
```

Fixed selection mode acquisition

This sample code implements selection for all entities within a Crossing window defined by the corner points 0,0,0 and 12,9,0 in the current UCS.

```
ads_name ss = { 0L, 0L };
ads_point pt1 = { 0.0, 0.0, 0.0 }, pt2 = { 12.0, 9.0, 0.0 };
if (ads_ssget("C", pt1, pt2, NULL, ss) != RTNORM) {
    ads_printf("\nNo selection-set was created.");
    return RTNORM;
}
ads_printf("\nSelection-set was created.");
ads_ssfree(ss);
```

Selection of all entities in the drawing database

This sample code implements selection for all entities within drawing, regardless of their properties (including layer state).

```
ads_name ss = { 0L, 0L };
if (ads_ssget("X", NULL, NULL, NULL, ss) != RTNORM) {
   ads_printf("\nNo selection-set was created.");
   return RTNORM;
}
ads_printf("\nSelection-set was created.");
ads_ssfree(ss);
```

Selection-set processing

There are a number of ADS library functions that process selection-sets.

Add entities to an existing selection-set: **ads_ssadd();**

Find the number of entities within a selection-set: **ads_sslength();**

Find an entity within a selection-set using a zero-based index: **ads_ssname();**

Determine whether an entity is a part of an existing selection-set: **ads_ssmemb();**

Remove an entity from a selection-set: **ads_ssdel();**

Transforming selection-sets

Using AutoCAD commands

After a selection-set has been acquired, the entities within the selection-set can be uniformly acted upon by an AutoCAD command such as MOVE.

This sample code shows how to acquire a selection-set interactively, then pass the selection-set to the MOVE command the move the selection-set 1 unit in positive X and Y in the current UCS.

```
ads_name ss = { 0L, 0L };
if (ads_ssget(NULL, NULL, NULL, NULL, ss) != RTNORM) {
   ads_printf("\nNo selection-set was created.");
   return RTNORM;
}
if (ads_command(RTSTR, "._MOVE", RTPICKS, ss, RTSTR, "", RTSTR, "0,0,0", RTSTR,
"1,1,0", RTNONE) != RTNORM) {
   ads_fail("Move command failed.");
   ads_ssfree(ss);
   return RTERROR;
}
ads_ssfree(ss);
return RTNORM;
```

The ADS library function ads_xformss()

Selection-sets can be **tranformed** by the ADS library function ads_xformss().

ads_xformss() takes two arguments: a selection-set, and a transformation matrix of type ads_matrix. It applies the 4x4 tranformation matrix uniformly to all the entities within a selection-set. This allows an ADS application to transparently move, scale or rotate a selection-set in 3-dimensional space without incurring the overhead of calling a built-in AutoCAD command, or calling ads_entmod() for each entity in the selection-set.

ads_xformss() can perform only uniform scaling of entities. It returns RTNORM on success or RTERROR on failure.

This matrix will move a selection-set 1 unit in positive X and Y in the current UCS.

1.0	**0.0**	**0.0**	**1.0**
0.0	**1.0**	**0.0**	**1.0**
0.0	**0.0**	**1.0**	**0.0**
0.0 0.0 0.0 1.0			

This matrix will scale a selection-set by a factor of 0.5 (moving it toward the origin).

0.5	**0.0**	**0.0**	**0.0**
0.0	**0.5**	**0.0**	**0.0**
0.0	**0.0**	**0.5**	**0.0**
0.0 0.0 0.0 1.0			

This matrix is a combination of the two previously shown, and does what you expect: move and scale the selection-set.

0.5	**0.0**	**0.0**	**1.0**
0.0	**0.5**	**0.0**	**1.0**
0.0	**0.0**	**0.5**	**0.0**
0.0 0.0 0.0 1.0			

This sample code shows how to acquire a selection-set interactively, then apply the transformation matrix previously illustrated to the selection-set by a call to ads_xformss().

```
ads_name ss = { 0L, 0L };
ads_matrix matrix = { {0.5, 0.0, 0.0, 1.0},
                      {0.0, 0.5, 0.0, 1.0},
                      {0.0, 0.0, 0.5, 0.0},
                      {0.0, 0.0, 0.0, 1.0} };

if (ads_ssget(NULL, NULL, NULL, NULL, ss) != RTNORM) {
    ads_printf("\nNo selection-set was created.");
    return RTNORM;
}
if (ads_xformss(ss, matrix) != RTNORM) {
    ads_fail("Error transforming selection-set.");
    ads_ssfree(ss);
    return RTERROR;
}
ads_ssfree(ss);
return RTNORM;
```

Dragging selection-sets

The ADS library function ads_draggen()

Selection-sets can be **dragged** by the ADS library function ads_draggen().

ads_draggen() takes five arguments: a selection-set, a prompt for the user, an integer code specifying the cursor type to display, a pointer to a user-defined function that updates the transformation matrix used to update the drag image of the selection-set, and a variable of type ads_point into which the user's final point selection will be copied.

The user-defined function called by ads_draggen() must have two arguments: a variable of type ads_point, and a variable of type ads_matrix. ads_draggen() sends the current cursor location to the function as an argument, and then applies the 4x4 tranformation matrix (which was probably changed in some manner within the user-defined function) uniformly to all the entities within a selection-set. This allows an ADS application to drag a selection-set without invoking an AutoCAD command.

ads_draggen() returns RTNORM on success, RTERROR on failure, or RTCAN if the user enters Ctrl-C. It can also return either RTNONE, RTKWORD or RTSTR, based on a previous call to ads_initget().

This sample code shows how to acquire a selection-set interactively, then drag it by a call to ads_draggen().

```
int
tdrag(struct resbuf *rb)
{
    int stat;
    ads_name ss;
    ads_point return_pt;
    stat = ads_ssget(NULL, NULL, NULL, NULL, ss);
    if (stat != RTNORM) {
        ads_printf("\nNo selection-set was created.");
        return RTNORM;
    }
    stat = ads_getpoint(NULL, "\nBase point: ", basept);
    if (stat != RTNORM) {
        ads_fail("Error getting base point.");
        ads_ssfree(ss);
        return RTERROR;
    }
    stat = ads_draggen(ss, "\nMove 'em...", 0, drag1,
return_pt);
    if (stat != RTNORM) {
        ads_fail("Error in dragging.");
        ads_ssfree(ss);
        return RTERROR;
    }
    ads_ssfree(ss);
    return RTNORM;
}
int
drag1(ads_point usrpt, ads_matrix matrix)
{
    ident_init(matrix);
   matrix[0][T] = usrpt[X] - basept[X];
    matrix[1][T] = usrpt[Y] - basept[Y];
    matrix[2][T] = usrpt[Z] - basept[Z];
    return RTNORM;
}
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;
}
```

Extended Entity Data