The ABCs of APIs Lesson 6
Using Structs

Table of Contents

In Lesson 2 we learned how to use the **MoveWindow** API call to relocate and resize a window or control. We can retrieve the coordinates of a window or control with **GetWindowRect**. This function requires a STRUCT. A STRUCT is similar to the PTR we discussed in Lesson 5 because it is a pointer to a location in memory. The similarity ends there, though.

Defining a STRUCT

A struct must be defined before it can be used. A STRUCT statement looks like this:

```
STRUCT StructName, firstMember as type, secondMember as type
```

The struct name is listed first, followed by a list of its parts, or **members** separated by commas. The struct may have many members, or just one member. Here is a simple struct with four members.

```
    STRUCT RECT, left as long, top as long, right as long,
 bottom as long.
```

In an earlier lesson we said that a CALLDLL statement must be on a single line. A STRUCT statement must also be on a single line. It can be placed on multiple lines if the underscore line continuation character is used. Liberty BASIC then sees the struct statement as a single line. Here is the struct as it is typed on multiple lines:

```
    STRUCT RECT,_
        left as long,_
        top as long,_
        right as long,_
        bottom as long.
```

Struct members can be the same types we used the CALLDLL statement. See Lesson 1 . Here is an example that has a PTR string member and a numeric long member.

```
    STRUCT PERSON, name$ as PTR, age as long
```

STRUCTs as Arguments

A struct is passed into a calldll statement as an argument with type "as struct". Here is a generic calldll statement that includes a struct.

```
    STRUCT StructName, firstMember as long secondMember as ulong

    CALLDLL FunctionName, handle as ulong, StructName as struct,
 result as long
```

Assigning Values

Some API calls require the programmer to place values into struct members before the function is called. We assign values to the struct members with the equals sign. The struct members are accessed with the name of the struct, followed by a dot, then the member name, another dot, then the word STRUCT. Here is a simple example that creates a struct with a PTR member and a LONG member, then assigns values to both members.

```
    STRUCT PERSON, name$ as PTR, age as long
```

```
PERSON.name$.struct = "John Smith"
PERSON.age.struct = 45
```

Size of STRUCT

Some API functions that have struct arguments have an argument for the size of the struct. We can get the size of a struct with Liberty BASIC's native **LEN()** function with the name of the struct followed by a dot, then the word "struct" inside the parentheses, like this:

```
StructSize = LEN(PERSON.struct)
```

Note that a bug in Liberty BASIC requires the word "struct" to be in lowercase letters. If instead "STRUCT" is used with LEN() a "type mismatch" error is generated by Liberty BASIC.

Here is a code snippet to illustrate using **LEN()** to get the size of a struct:

```
STRUCT PERSON,name$ as PTR, age as long

StructSize = len(PERSON.struct)
print "The size of the struct is ";StructSize
```

Retrieving Values

We retrieve values from structs in a similar way to the method used to assign values. To assign a value:

```
STRUCT PERSON,name$ as PTR, age as long

PERSON.age.struct = 45
```

To retrieve the value:

```
STRUCT PERSON,name$ as PTR, age as long

PERSON.age.struct = 45
ageValue = PERSON.age.struct
print "Age is "; ageValue
```

Getting a Window's Coordinates

We now know enough to make the **GetWindowRect** API call to retrieve a window's coordinates. The function requires a RECT struct, which looks like this:

```
    STRUCT RECT,_
        left as long,_
        top as long,_
        right as long,_
        bottom as long.
```

The Microsoft documentation for this structure gives us this information:

Members
left
Specifies the x-coordinate of the upper-left corner of the rectangle.
top
Specifies the y-coordinate of the upper-left corner of the rectangle.
right
Specifies the x-coordinate of the lower-right corner of the rectangle.
bottom
Specifies the y-coordinate of the lower-right corner of the rectangle.

The API function looks like this:

```
calldll #user32, "GetWindowRect",_
    hWindow as ulong,_  'window handle
    RECT as struct,_    'struct containing info
    result as long      'nonzero = success
```

Here is a small program that uses this function. It first sets up a RECT struct and obtains the handle of the window with Liberty BASIC's **HWND**() function. It then makes the **GetWindowRect** API call. After the function returns, the window's coordinates are contained in the struct. The program prints them in the mainwin for us to see.

```
open "Test Window" for window as #1
#1 "trapclose [quit]"

'get the window's handle
hWindow = HWND(#1)
```

```
'declare a struct:
    STRUCT RECT,_
        left as long,_
        top as long,_
        right as long,_
        bottom as long

'retrieve window coordinates:
calldll #user32, "GetWindowRect",_
    hWindow as ulong,_  'window handle
    RECT as struct,_    'struct containing info
    result as long      'nonzero = success

'print coords in mainwin:
print "The left coordinate is ";RECT.left.struct
print "The top coordinate is ";RECT.top.struct
print "The right coordinate is ";RECT.right.struct
print "The bottom coordinate is ";RECT.bottom.struct

wait
[quit] close #1:end
```

Using Values Retrieved from Structs

The previous example printed the values in the struct. We can also assign their values to variables. If we use the struct again in another API call, the values will change. We can save the values in variables. To save the top coordinate of the window in the variable WindowTop, we do this:

```
    WindowTop = RECT.top.struct
```

The values in structs can be used in any way that variables and literal values can be used. Since we know the window's coordinates, we can obtain its width and height by subtracting values. We subtract the left value from the right value to get the width. We subtract the top value from the bottom value to get the height.

```
'do math on retrieved values to get width and height:
width = RECT.right.struct - RECT.left.struct
height = RECT.bottom.struct - RECT.top.struct
```

Here it is, used in the sample program:

```
open "Test Window" for window as #1
#1 "trapclose [quit]"

'get the window's handle
hWindow = HWND(#1)

'declare a struct:
    STRUCT RECT,_
        left as long,_
        top as long,_
        right as long,_
        bottom as long

'retrieve window coordinates:
calldll #user32, "GetWindowRect",_
    hWindow as ulong,_  'window handle
    RECT as struct,_    'struct containing info
    result as long      'nonzero = success

'do math on retrieved values to get width and height:
width = RECT.right.struct - RECT.left.struct
height = RECT.bottom.struct - RECT.top.struct

print "Width is ";width
print "Height is ";height


wait
[quit] close #1:end
```

More?

You can improve your STRUCT skills by trying these methods on other types of windows and controls. What happens if you try to obtain the coordinates of a button? A dialog window? Try it and see!

What's Next?

Lesson 7 will discuss the use of structs as pointers to numeric values.

Written by Alyce Watson. For more on APIs, see:
APIs for Liberty BASIC