The ABCs of APIs Lesson 5
Retrieving Text Strings from API calls.

Table of Contents

In Lesson 4 we learned how to use the **SetWindowTextA** API call to send a new caption to a window as a string of text. We can also retrieve the text contained in windows and controls with **GetWindowTextA**. Liberty BASIC allows us to retrieve text from a textbox, texteditor and from the text field in a combobox, but not from other controls and windows. We can expand Liberty BASIC's capabilities with **GetWindowTextA**.

String Buffers

In the previous lesson, we learned that strings are sent into API calls as pointers to the location of the text in memory. Because the API function knows the memory address, it can modify the data at that address.

We must set up a string variable for the PTR argument in **GetWindowTextA**. The function retrieves the text and places it in this string variable. The variable can also be called a buffer.

Here's how we set up an empty string. Notice that we are using the handy **SPACE$()** function. Liberty BASIC allows us to create a string with a specified number of blank spaces.

```
Caption$ = space$(128)
```

The **GetWindowTextA** function also requires an argument that specifies the length of the string buffer. We can get the length with the **LEN()** function, like this:

```
    length=len(Caption$) + 1
```

We add 1 to the length argument because Liberty BASIC adds a null termination character to strings passed into API calls.

We'll retrieve the window handle as we've done in previous lessons with **HWND()**. **GetWindowTextA** looks like this:

```
    CallDll #user32, "GetWindowTextA",_
        h as ulong,_        'window handle
        Caption$ as ptr,_   'string buffer
        length as long,_    'size of buffer
        result as long
```

Here is a small program that retrieves the caption of the window and gives us a notice:

```
nomainwin
Open "Window Caption" for window as #1
    #1 "trapclose [quit]"

    h = hwnd(#1)

    'create a string buffer:
    Caption$ = space$(128)
    length=len(Caption$) + 1

    CallDll #user32, "GetWindowTextA",_
        h as ulong,_        'window handle
        Caption$ as ptr,_   'string buffer
        length as long,_    'size of buffer
        result as long

    notice "Caption is: ";Caption$
wait
[quit] close #1:end
```

Trimming the String

Our string is 128 characters long -- much longer than the text retrieved. **GetWindowTextA** places a null termination character at the end of the text string it returns. A null character is CHR$(0). We can trim the string of blank spaces and null characters with the **TRIM$()** function, like this:

```
    notice "Caption is: ";TRIM$(Caption$)
```

Length of Return String

The **GetWindowTextA** function returns the length of the text. This number is in the "result" part of the API call. Many text retrieval API functions tells us the length of the string and this is very handy. Here is the function again, with the result commented.

```
    CallDll #user32, "GetWindowTextA",_
        h as ulong,_        'window handle
        Caption$ as ptr,_   'string buffer
        length as long,_    'size of buffer
        result as long      'length of returned string
```

And here is our small program again, but this time it gives the user a notice of the length of the caption, rather than the text in the caption:

```
nomainwin
Open "Window Caption" for window as #1
    #1 "trapclose [quit]"

    h = hwnd(#1)

    'create a string buffer:
    Caption$ = space$(128)
    length=len(Caption$) + 1

    CallDll #user32, "GetWindowTextA",_
        h as ulong,_        'window handle
        Caption$ as ptr,_   'string buffer
        length as long,_    'size of buffer
        result as long      'length of returned string

    notice "Length of caption is: ";result
```

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

Since we know the length of the returned string, we can also trim it with the **LEFT$()** command, like this:

```
    Caption$ = left$(Caption$, result)
```

Resetting the Buffer

Once the **GetWindowTextA** function has placed a null terminated string of text in the buffer, the size of the buffer has changed. If we plan to use the same variable as a string buffer again in the program, each time we use it we must set aside sufficient memory for it.

```
    Caption$ = space$(128)
```

Size of Buffer

When we attempt to retrieve text with **GetWindowTextA** we must create a large enough buffer, or the function cannot return all of the text. There is another API function that returns the length of the text. We can use this number to create a buffer that is large enough to hold the text.

The function looks like this:

```
    calldll #user32, "GetWindowTextLengthA",_
    h as ulong,_        'window or control handle
    numberChars as long 'returns length of text
```

Creating a buffer with the information is done like this:

```
    Caption$ = Space$(numberChars)
```

And here is the small program with that added function:

```
nomainwin
Open "Window Caption" for window as #1
    #1 "trapclose [quit]"
```

```
    h = hwnd(#1)

    calldll #user32, "GetWindowTextLengthA",_
        h as ulong,_        'window or control handle
        numberChars as long 'returns length of text
    notice numberChars
    'create a string buffer:
    Caption$ = space$(numberChars)
    length=len(Caption$) + 1

    CallDll #user32, "GetWindowTextA",_
        h as ulong,_        'window handle
        Caption$ as ptr,_   'string buffer
        length as long,_    'size of buffer
        result as long

    Caption$ = left$(Caption$, result)
    notice "Caption is ";Caption$
wait
[quit] close #1:end
```

PTRs in API Calls

Things to Remember

- A PTR is a pointer to an address in memory
- We pass strings as arguments in API calls by passing a pointer to their location in memory.
- If we want the function to place text into our string variable, we must make it large enough to hold the text.
- We make a string buffer by creating a string variable and filling it with empty spaces with the **SPACE$()** function.

Most of the API calls that deal with strings work like the examples in Lesson 4 and Lesson 5.

Experiment!

If you want to hone your skills with text in API calls, take what you've learned in Lesson 4 and Lesson 5 and change or retrieve text in other Liberty BASIC windows or controls. Try retrieving the caption of a button. Try giving a new caption to a groupbox. You'll find that string API functions are very helpful and not difficult to use.

What's Next?

Lesson 6 will discuss the use of structs in API calls.

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