DLLAccess

Word for Windows 2.0 macros
To demonstrate usage of DLLAcces.DLL

To

This document introduces the use of **DLLAcces.DLL**. **DLLAcces.DLL** is a set of very simple DLL routines that allows you to Peek and Poke values into memory. It is most useful to access data structure that are returned from other DLL or API calls, or to pass data structures to those DLL or API calls.

!! WARNING !!

These routines are not for the novice! Use at your own risk!
If you pass the wrong parameters, you can crash WinWord!
Save Often!

The idea is that you've got a pointer to a structure. If you know what the structure is, you can figure out where each field of the structure is. You just pass the pointer and the location of the field inside the structure to the **DLLAcces.DLL** routines to get or set the field. Simple. *But you better make sure your offset calculations are correct!*

Here are the standard sizes of some data types:

Data Type	Number of Bytes
Integer	2
Long	4
Float	4
Double	8
LPSTR	4
LPCSTR	4

LP*anything*	4
WORD	2
DWORD	4
Handle	2
HWND	2
HINSTANCE	2
UINT	2

If the structure has the following definition:

```
typedef struct myStructTag {
        DWORD           lStructSize;
        HWND            hWnd;
        LPSTR           lpStr;
        DWORD           nMaxStr;
} myStruct;
```

Then the structure's overall size is 14 (4 + 2 + 4 + 4), and the following offsets are true:

lStructSize	**0**
hWnd	**4**
lpStr	**6**
nMaxStr	**10**

For most structures in Windows, the first element in the passed structure is the size of the structure, so you'd set **lStructSize** to **14** with a call like:

```
r = DA_SetLong(lpStruct, 0, 14)
```

Now comes the question of how to create the structure in the first place so that you can pass it to an API or DLL function. Well, we've provided two routines for accessing global memory—**AllocVariable** and **FreeVariable**. These routines **must** be used in pairs, or you'll lose memory!!

The calling convention goes like:

hStruct = 0 : lpStruct = AllocVariable(lStructSize, hStruct)

......use lpMem as the pointer to memory ....

FreeVariable(hStruct)

The **hStruct** is a handle to the memory that **must** be used to free the memory. It's returned by **AllocVariable**, *but you have to make sure that **hStruct** exists before the call*. **lpStruct** is the actual pointer that you'd use.

Look at the **CommDlgExample** to see how everything fits. It calls the **GetOpenFileName** routine. It sets up filters for files, and changes the title of the dialog box, etc. When it returns, a dialog box displays the filename, its path, and the extension

Save everything before running this example!.....just in case......

Double-click

To use **DLLAcces.DLL**, copy the DLL to your WINDOWS/SYSTEM directory. The following sections gives the declare statements, and explanations how they're used.

Integer Functions

Declare Function DA_GetInt Lib "dllacces.dll"(lpstruct As Long, offset As Integer) As Integer

Declare Function DA_SetInt Lib "dllacces.dll"(lpstruct As Long, offset As Integer, value As Integer) As Integer

These functions get and set integer values. **lpstruct** is the pointer to the structure, and **offset** is the location in the structure that the integer comes from or is to go. For DA_SetInt, **value** is the new value to set.

Here's how you use them:

intval = DA_GetInt(lpStruct, offset)

r = DA_SetInt(lpStruct, offset, newintval)

DA_SetInt always returns **0**.

Long Functions

Declare Function DA_GetLong Lib "dllacces.dll"(lpstruct As Long, offset As Integer) As Long

Declare Function DA_SetLong Lib "dllacces.dll"(lpstruct As Long, offset As Integer, value As Long) As Integer

These functions get and set Long integer values. **lpstruct** is the pointer to the structure, and **offset** is the location in the structure that the Long integer comes from or is to go. For **DA_SetLong**, **value** is the new value to set.

Here's how you use them:

longval = DA_GetLong(lpStruct, offset)

r = DA_SetLong(lpStruct, offset, newlongval)

DA_SetLong always returns **0**.

Float Functions

Declare Function DA_GetFloat Lib "dllacces.dll"(lpstruct As Long, offset As Integer) As Double

Declare Function DA_SetFloat Lib "dllacces.dll"(lpstruct As Long, offset As Integer, reverse As Integer, value As Double) As Integer

These functions get and set Float values. **lpstruct** is the pointer to the structure, and **offset** is the location in the structure that the Float comes from or is to go. For **DA_SetFloat**, **value** is the new value to set. Note how **DA_GetFloat** returns Double and the **value** parameter in **DA_SetFloat** is declared as Double. That's because WordBasic does not have a Float type.

There's an additional argument called **reverse**. That's to fix a bug in WordBasic. As it turns out, WordBasic passes Doubles incorrectly. Setting **reverse** to **1** fixes this problem. This makes **DLLAcces.DLL** useful for other applications, and also if and when Microsoft fixes the bug; all you have to do is to change **reverse** to **0**.

Here's how you use these functions:

floatval = DA_GetFloat(lpStruct, offset)

r = DA_SetFloat(lpStruct, offset, 1, newfloatval)

DA_SetFloat always returns **0**.

Double Functions

Declare Function DA_GetDouble Lib "dllacces.dll"(lpstruct As Long, offset As Integer) As Double

Declare Function DA_SetDouble Lib "dllacces.dll"(lpstruct As Long, offset As Integer, reverse As Integer, value As Double) As Integer

These functions get and set Double values. **lpstruct** is the pointer to the structure, and **offset** is the location in the structure that the Double comes from or is to go. For **DA_SetDouble**, **value** is the new value to set.

There's an additional argument called **reverse**. That's to fix a bug in WordBasic. As it turns out, WordBasic passes Doubles incorrectly. Setting **reverse** to **1** fixes this problem. This makes **DLLAcces.DLL** useful for other applications, and also if and when Microsoft fixes the bug; all you have to do is to change **reverse** to **0**.

Here's how you use these functions:

dblval = DA_GetDouble(lpStruct, offset)

r = DA_SetDouble(lpStruct, offset, 1, newdblval)

DA_SetDouble always returns **0**.

String Functions

Declare Function DA_GetString$ Lib "dllacces.dll"(lpstruct As Long, offset As Integer)

Declare Function DA_SetString Lib "dllacces.dll"(lpstruct As Long, offset As Integer, value$) As Integer

DA_GetString$ returns a string from the given offset. Pretty self-explanatory.

DA_SetString has questionable value. **DLLAcces.DLL** creates a copy of the input string and sets the structure field to point at the copy. The problem is that you can't free that memory. If, on the other hand, **DA_SetString** just sets the pointer, there's no guarantee that the string in WinWord will stick around. At least I couldn't get consistent behavior.

Here's a better way of creating a string to pass to a structure:

```
Declare Function lstrcpy3 Lib "Kernel"(lp As Long, lpString2$) As Long Alias
"lstrcpy"

Sub MAIN
	hStr = 0 : lpStr = AllocVariable(256, hStr)
	lp = lstrcpy3(lpStr, "This is a string to set.")
	r = DA_SetLong(lpStruct, offset, lpStr)
	....
Bye:
	FreeVariable(hStr)
End Sub
```

By the way, this is also how you can pass pointers to integers, longs, doubles, etc.

Function Pointers

Declare Function DA_GetFUNCPTR Lib "dllacces.dll"(lpstruct As Long, offset As Integer) As Long

Declare Sub DA_ExecVoidFUNCPTR Lib "dllacces.dll"(fnc As Long)

Declare Function DA_ExecIntFUNCPTR Lib "dllacces.dll"(fnc As Long) As Integer

Declare Function DA_ExecLongFUNCPTR Lib "dllacces.dll"(fnc As Long) As Long

These functions deal with pointers to functions. If a structure gives you a pointer to function, you can get it with **DA_GetFUNCPTR**. You can execute it with the other calls. Which one you use will depend upon whether that function returns a value, and what type it returns.

There's no way to really set a CallBack function to a macro, so **DA_SetFUNCPTR** is not supported. (You can use **DA_SetLong** to get the same results if you already have the function pointer.)

Licensing

If you're an in-house or commercial developer, we want to encourage you to use the **DLLAcces.DLL** routines. The licensing rules are quite simple.

An individual who owns **DLLAcces.DLL** may use the routines on their own computer(s) any way they like.

A company or organization using the routines internally must have a **DLLAcces.DLL** site license. If you distribute a WordBasic program using a **DLLAcces.DLL** Library routine to, say, 100 internal users, you should have a **DLLAcces.DLL** site license for 100 users. Site licenses are available by writing to:

Artemis Associates
3083 Rasmus Circle
San Jose, CA 95148-3140

If you want to distribute **DLLAcces.DLL** as part of a commercial package, there's a small licensing fee.

Disclaimer

Obviously, we can't guarantee that these routines are going to work in every situation. You'll have to try them and use your judgment to see if they'll work for you. The sample code contains suggestions, tips, warnings and the like, and you should take a close look at what's being said. It's worth repeating the warning here:

!! WARNING !!

These routines are not for the novice! Use at your own risk!
If you pass the wrong parameters, you can crash WinWord!
Save Often!

If you distribute code, be sure to test on as many platforms as possible!

We *can* promise two things: if you have a problem, we'll work hard to get it solved; and if you're ever dissatisfied with the **DLLAcces.DLL** Library, for any reason, our 100% no-questions-asked lifetime moneyback guarantee applies. Period.