

Calling C functions from FileMaker

For some years we have our [CLibrary](#) and [CFunction](#) functions in our [MBS FileMaker Plugin](#). They can be used to load a library file, dylib on macOS, DLL on Windows and so on Linux. Then you can inspect the library for offered functions and load one. You need to have the declarations for C to know what parameters and return types are used. But once you translate them to our plugin's notation, you can load the function and call it.

Our first example is the [version](#) function from SQLite:

```
const char *sqlite3_libversion(void);
```

The return value is a C string, so from our documentation we pick Z as notation for this data type. No parameters are given, so the signature is just "()Z". We can load the libsqlite3.dylib file on macOS with the full path. Please notice this file doesn't exist on disk in modern macOS since the library is loaded from a cache file provided by Apple. Once we loaded the function, we can keep the object around and call it anytime. Our call function takes whatever parameters you like to pass and returns the result, which is converted to a value in FileMaker.

The following script brings it all together to query the SQLite version shipped by Apple with macOS:

```
Set Variable [ $library ; Value: MBS( "CLibrary.Load"; "/usr/lib/libsqlite3.dylib" ) ]
If [ MBS("IsError") ]
    Show Custom Dialog [ "Failed to load library." ; $library ]
Else
    Set Variable [ $function ; Value: MBS( "CLibrary.LoadFunction"; $library;
"sqlite3_libversion"; "()Z" ) ]
    If [ MBS("IsError") ]
        Show Custom Dialog [ "Failed to load function." ; $function ]
    Else
        Set Variable [ $version ; Value: MBS( "CFunction.Call"; $function ) ]
        Show Custom Dialog [ "SQLite version:" ; $version ]
        Set Variable [ $r ; Value: MBS( "CFunction.Release"; $function ) ]
    End If
    Set Variable [ $r ; Value: MBS( "CLibrary.Release"; $library ) ]
End If
```

If you'd like to adapt it for Windows, you only need to change the path to the sqlite3.dll somewhere on disk. Please make sure the 32/64bit version matches the application. FileMaker Pro is 64-bit only since version 19. On linux you may just refer to "/usr/lib/aarch64-linux-gnu/libsqlite3.so.0" to load it. This is the path for ARM and the Intel version has a bit different path.

Now we can do a more advanced example. Windows comes with a `GetUserNameW` function:

```
BOOL GetUserNameW(  
    [out] LPWSTR lpBuffer,  
    [in, out] LPDWORD pcbBuffer  
);
```

This function returns the current user name in a buffer. The function is defined in `Advapi32.dll`, so that is the DLL to load. Since this is a system DLL, we don't need to specify the full path as Windows finds it for us. The function itself returns a `BOOL`, which is a 4 byte boolean value, so we can use type "i" for our signature, since that is a 32-bit (4 byte) integer. The two parameters are both pointers. First pointer is the buffer where to put the name, so we need to allocate memory for this. The second parameter is the length of the buffer on input and the number of characters used for output. We pass the pointer, which means the address of where the value is stored.

Our plugin has extra functions for arrays. We use [CFunction.AllocateArray](#) to allocate an array for the buffer with 256 unicode characters. Since these characters are 16bit, we can allocate an array of 16 bit short values. Actually the type doesn't matter much as long as the size of the array is calculated correctly, or being larger than needed. The other array is allocated with just one number value to hold the length. We use [CFunction.SetArray](#) to put in the length of the buffer. Since strings in C have an end character, we allocate 256 characters, but pass 255 to Windows to have a save buffer on the end.

We call the function using [CFunction.Call](#) and check the result. Since we allocate arrays for some parameter indexes, the call function will use these instead of whatever we pass as parameters. The result is either 1 on success or 0 on failure. Then we can read the array values for the length array to get the actual length used in the array. This is actually one higher due to the end character. We use [CFunction.GetArray](#) again to query characters as numbers. We loop over them to get characters using `Char()` function.

Finally we can cleanup with [CFunction.FreeArray](#) function, then later release the function with [CFunction.Release](#) and the library with [CLibrary.Release](#). Technically you can decide to not release the library as it will happen automatically when the app closes. Same way with the function object, which could also stay in memory until FileMaker quits.

The final script looks like this and shows a dialog with the user name:

```
# we load the library and the function
```

```

Set Variable [ $library ; Value: MBS( "CLibrary.Load"; "Advapi32.dll" ) ]
If [ MBS("IsError") ]
    Show Custom Dialog [ "Failed to load library." ; $library ]
Else
    Set Variable [ $function ; Value: MBS( "CLibrary.LoadFunction"; $library;
"GetUserNameW"; "(pp)i" ) ]
    If [ MBS("IsError") ]
        Show Custom Dialog [ "Failed to load library." ; $function ]
    Else
        # since 1st parameter is WChar Array, we allocate space for it
        Set Variable [ $r ; Value: MBS( "CFunction.AllocateArray"; $function;
0; "S"; 256 ) ]
        # and for the count variable as ptr, we allocate a one value array
        Set Variable [ $r ; Value: MBS( "CFunction.AllocateArray"; $function;
1; "I"; 1 ) ]
        # put in 255 for the length of the first array (minus one character for
end character)
        Set Variable [ $r ; Value: MBS( "CFunction.SetArray"; $function; 1; "I";
255) ]
        # Now call function
        Set Variable [ $result ; Value: MBS( "CFunction.Call"; $function ) ]
        If [ $result = 1 ]
            # now we read how many characters we got
            Set Variable [ $r ; Value: MBS( "CFunction.GetArray"; $function;
1; "I" ) ]

            Set Variable [ $length ; Value: GetValue($r; 1) ]
            # and get the characters
            Set Variable [ $characters ; Value: MBS( "CFunction.GetArray";
$function; 0; "S" ) ]
            #
            # we loop over the values to build the text
            Set Variable [ $text ; Value: "" ]
            Set Variable [ $index ; Value: 1 ]
            Loop [ Flush: Always ]
                Set Variable [ $char ; Value: GetValue($characters;
$index) ]

                Set Variable [ $text ; Value: $text & Char($char) ]
                #
                Set Variable [ $index ; Value: $index + 1 ]
                Exit Loop If [ $index = $length ]
            End Loop
            #
            Show Custom Dialog [ "Windows user name" ; $text ]
            #
            # cleanup

```

```
        Set Variable [ $r ; Value: MBS( "CFunction.FreeArray";  
$function; 0) ]  
        Set Variable [ $r ; Value: MBS( "CFunction.FreeArray";  
$function; 1) ]  
    End If  
    Set Variable [ $r ; Value: MBS( "CFunction.Release"; $function) ]  
End If  
Set Variable [ $r ; Value: MBS( "CLibrary.Release"; $library) ]  
End If
```

This is just an example to show you how to use these functions. Not all libraries and functions can be used in FileMaker this way. Usually it is better to just write little helper applications and call them via [Shell](#) functions. If you pass parameters incorrectly, you can easily crash the application. But sometimes it is certainly worth the effort!

Please contact us if you have questions about these functions.