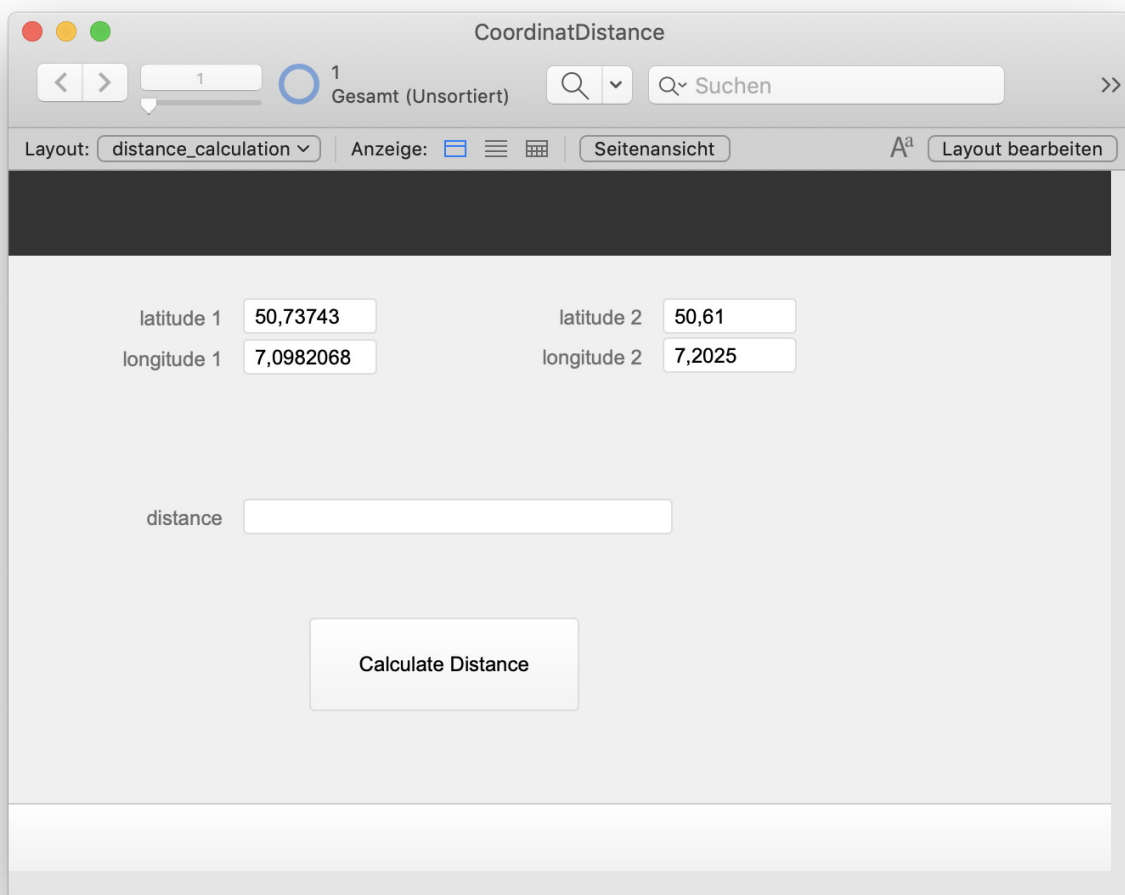


FileMaker and JavaScript - the perfect combination

In the newest version of the [MBS FileMaker Plugin](#) 10.0 we offer functions for the use of JavaScript without the use of a web viewer. See [WebView.RunJavaScript](#) and [WebView.Evaluate](#) to run or evaluate JavaScript in a web viewer. The new [JavaScript](#) functions increase your functionality in FileMaker enormously. You can write your own Javascript, use a JavaScript snippet that fix a problem or fast up a process of your solution. In this article I want to show you how to use JavaScript code in FileMaker: Setting values in the JavaScript with functions in FileMaker, add functions and run them.

If you have a problem that you want to solve, you don't have to reinvent the wheel again and again. Perhaps someone has already found a solution to this problem and shared his solution on the Internet. So start a search engine and try your luck. Special thanks to André Rinas and his website [andrerinas.de](#) that supply very useful JavaScript snippets to everyone. This example is based on one of this JavaScript codes.



The screenshot shows a web viewer window titled "CoordinatDistance". The interface includes a navigation bar with a search field containing "Suchen" and a "Seitenansicht" button. Below the navigation bar, there are input fields for "latitude 1" (50,73743), "latitude 2" (50,61), "longititude 1" (7,0982068), and "longititude 2" (7,2025). A "distance" field is empty. A "Calculate Distance" button is positioned below the input fields.

In the example I show you a JavaScript that calculates the difference between two geo coordinates. We create a new FileMaker table with 5 fields. Two fields contains the longitude and latitude of the first coordinate, two other fields store the longitude and latitude of the other coordinate. The last field is a result field, that shows us the difference between the coordinates after we calculate it. Then we start with the JavaScript code. Javascript is a programming language that was develop as an extension of HTML and CSS on the Internet. But is finding its way into more and more domains.

For the calculation we need the longitude and latitude in radian and not in the unit of degree. Because of this we need a function that convert between this both units. The formula for the calculation is $y = x * \text{Pi} / 180$.

A [JavaScript](#) function for that looks like this:

```
function DegToRad( deg ) {  
    return deg * Math.PI / 180;  
}
```

We need the keyword "function" to say that we defined a function. Then we need a function name. Function names in Javascript are only one string and can contains numbers. In the brackets we defined the parameters that we need in this function. We set them in the function call. The calculation is surrounded by curly brackets. The keyword return defines that the value of this calculation is the result of the function and returns the value. Math.Pi is a function that defines Pi, so we use it here to do the math.

Then we can write the function for the distance calculation:

```
function distance( lat1, lon1, lat2, lon2 ) {  
    var R = 6371; // The earth's radius in km  
  
    lat1 = DegToRad(lat1);  
    lat2 = DegToRad(lat2);  
    lon1 = DegToRad(lon1);  
    lon2 = DegToRad(lon2);  
  
    var x = (lon2 - lon1) * Math.cos((lat1 + lat2) / 2);  
    var y = (lat2 - lat1);  
    var d = Math.sqrt(x * x + y * y) * R;  
    return d;  
}
```

We have a function with four input parameters. That's the latitudes and longitudes of the coordinates. For each of these values we call the converting function and save the returned value in the variable again. Because these variables are defined as parameters of the function, we don't need to declare them again. All other variables x, y and d are declared with the keyword "var" in front of the variable name. The variable R defines the radius of the earth. Variables can be assigned with calculated values similar to FileMaker. When using variables we must take care of the fact that JavaScript is a case-sensitive language. A and a would be two different variables!

After we finish the script we want to look at the possibilities of using this in FileMaker. As we want to work with a JavaScript snippet, we create a new JavaScript environment with the "[JS.New](#)" function in the MBS Plugin. As a result we get the reference number that we save in the variable \$js. Then we want to evaluate our JavaScript snippet with the help of the "[JS.EvaluateToString](#)" function. In the parameters of the function we set \$js and our JavaScript.

```
Set Variable [ $r ; Value: MBS( "JS.EvaluateToString"; $JS;
"function DegToRad( deg )
{
    return deg * Math.PI / 180;
}

function distance( lat1, lon1, lat2, lon2 ) {
    var R = 6371; // The earth's radius in km

    lat1 = DegToRad(lat1);
    lat2 = DegToRad(lat2);
    lon1 = DegToRad(lon1);
    lon2 = DegToRad(lon2);

    var x = (lon2 - lon1) * Math.cos((lat1+lat2) / 2);
    var y = (lat2 - lat1);
    var d = Math.sqrt(x*x + y*y) * R;
    return d;
}

distance( 50.73743, 7.0982068, 50.61, 7.2025 )" ) ]
```

Then we have the result in the variable \$r. The result is a string. If we want the result as a JSON object we call the "[JS.Evaluate](#)" function in place of the "[JS.EvaluateToString](#)" function with the same parameters. In the end we release the JavaScript environment.

But it would be very annoying if we have to change the string of the JavaScript for each dataset to work with different values. We don't have to, because we can set and read variables in the JavaScript environment with the MBS Plugin functions.

Before we call one of the evaluation functions we set the variables like that:

```
Set Variable [ $r ; Value: MBS( "JS.SetGlobalPropertyValue";  
$js; "lat1"; 50.73743 ) ]  
Set Variable [ $r ; Value: MBS( "JS.SetGlobalPropertyValue";  
$js; "lon1"; 7.0982068 ) ]  
Set Variable [ $r ; Value: MBS( "JS.SetGlobalPropertyValue";  
$js; "lat2"; 50.61 ) ]  
Set Variable [ $r ; Value: MBS( "JS.SetGlobalPropertyValue";  
$js; "lon2"; 7.2025 ) ]
```

Then we Change the last line of our Javascript to:

```
distance( lat1, lon1, lat2, lon2 )
```

If we want to avoid to set this huge part of script in every evaluation function, we can add the DegToRad function and the distance function to the environment of the JavaScript. Because of this we only need to call the distance function with the right parameters. That is easier for debugging, too.

```
Set Variable [ $js ; Value: MBS( "JS.New" ) ]  
Set Variable [ $r ; Value: MBS( "JS.AddFunction"; $js;  
"DegToRad"; "function( deg ) { return deg * Math.PI /  
180; }" ) ]  
Set Variable [ $r ; Value: MBS( "JS.AddFunction"; $js;  
"distance"; "function( lat1, lon1, lat2, lon2 )  
{  
    lat1 = DegToRad(lat1);  
    lat2 = DegToRad(lat2);  
    lon1 = DegToRad(lon1);  
    lon2 = DegToRad(lon2);  
    var R = 6371;  
    var x = (lon2-lon1) * Math.cos((lat1+lat2)/2);  
    var y = (lat2-lat1); var d = Math.sqrt(x*x + y*y) * R;  
    return d;  
}" ) ]
```

Go to Record/Request/Page [[First](#)]
[Loop](#)

```
    Set Variable [ $lat1 ; Value: MBS( "JSON.CreateNumber";  
GetAsNumber(distance::lat1)) ]  
    Set Variable [ $lon1 ; Value: MBS( "JSON.CreateNumber";  
GetAsNumber(distance::lon1)) ]  
    Set Variable [ $lat2 ; Value: MBS( "JSON.CreateNumber";  
GetAsNumber(distance::lat2)) ]  
    Set Variable [ $lon2 ; Value: MBS( "JSON.CreateNumber";  
GetAsNumber(distance::lon2)) ]  
    Set Variable [ $res ; Value: MBS( "JS.CallFunction"; $js;  
"distance"; $lat1; $lon1; $lat2; $lon2 ) ]  
    Set Field [ distance::distance; $res ]  
    Go to Record/Request/Page [ Next ; Exit after last: On ]  
End Loop  
Set Variable [ $r ; Value: MBS( "JS.Free"; $js ) ]
```

I hope that you like our new [JavaScript](#) functions. I wish you lots of fun with it. If you have any questions, please do not hesitate to contact us.
by Stefanie Juchmes