Sort with evaluate in FileMaker

We got a few new functions to sort lists: List.SortWithEvaluate, QuickList.SortWithEvaluate and JSON.SortWithEvaluate.

The idea is to pass an expression to do the comparison for the sort operation. This way you can use functions like for example GetAsNumber() to convert from text to number.

If you run the function with simply comparing leftValue and rightValue like this:

MBS("List.SortWithEvaluate"; "1¶22¶3"; "leftValue < rightValue")
You get the list back with 1, 22 and 3. But if you use GetAsNumber(), you define the order for numbers like this:
MBS("List.SortWithEvaluate"; "1¶22¶3"; "GetAsNumber(leftValue) < GetAsNumber(rightValue)")

The list is now returned as 1, 3 and 22.

This function can handle various things including sorting by date with GetAsDate(), sorting only text after first 5 characters with Middle() function or do various other operations as needed.

Finally we applied the same technique to JSON with our new JSON.SortWithEvaluate function. You can sort array values here and do the JSON comparison in the expression by using built-in JSON functions in FileMaker or our MBS JSON functions. Here an example sorting a JSON array:

MBS("JSON.SortWithEvaluate"; "[2,9,3,1]"; "MBS(\"JSON.GetValue\"; leftJSON) < MBS(\"JSON.GetValue\"; rightJSON)")

Next an example to sort objects by a value of a key within each object. For example if you have records with a name as one of the fields, you can now sort by the field name:

```
MBS( "JSON.SortWithEvaluate"; "[
    {\"name\": \"Joe\"},
    {\"name\": \"Anna\"},
    {\"name\": \"Tim\"},
    {\"name\": \"John\"}
    ]"; "JSONGetElement ( leftJSON; \"name\") < JSONGetElement
( rightJSON; \"name\")" )
```

The expression can combine multiple comparisons to look e.g. on first and last name. The following expression would look first if last name is the same and then compare first name:

```
If (JSONGetElement ( leftJSON; \"lastName\") = JSONGetElement
( rightJSON; \"lastName\");
    JSONGetElement ( leftJSON; \"firstName\") < JSONGetElement
( rightJSON; \"firstName\");
    JSONGetElement ( leftJSON; \"lastName\") < JSONGetElement
( rightJSON; \"lastName\") )
```

If you do the standard sorting, our List.Sort, QuickList.Sort and JSON.Sort functions will be much faster. But this is for maximum flexibility. If needed we could put in some optimization of course for the future.