Data structures in MBS Plugin

Let us show you today the various data structures we have in MBS FileMaker Plugin and how to use them.

- QuickList
- Dictionary
- Matrix
- JSON References
- SQL Result

In general you can keep huge amounts of data in memory to query them anywhere with a quick lookup in one of the data structures. Such a lookup may avoid needing extra relationships and can be faster than a database lookup.

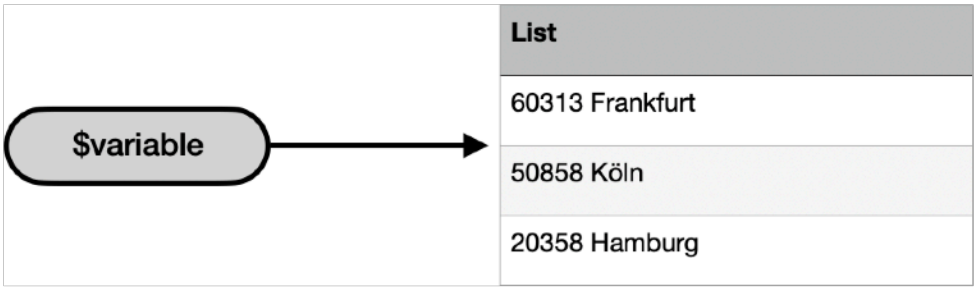
List

Normally in FileMaker you may have a variable containing a list of values. You append something with a calculation like this:

Set Variable [$list; $list & $value & ¶]

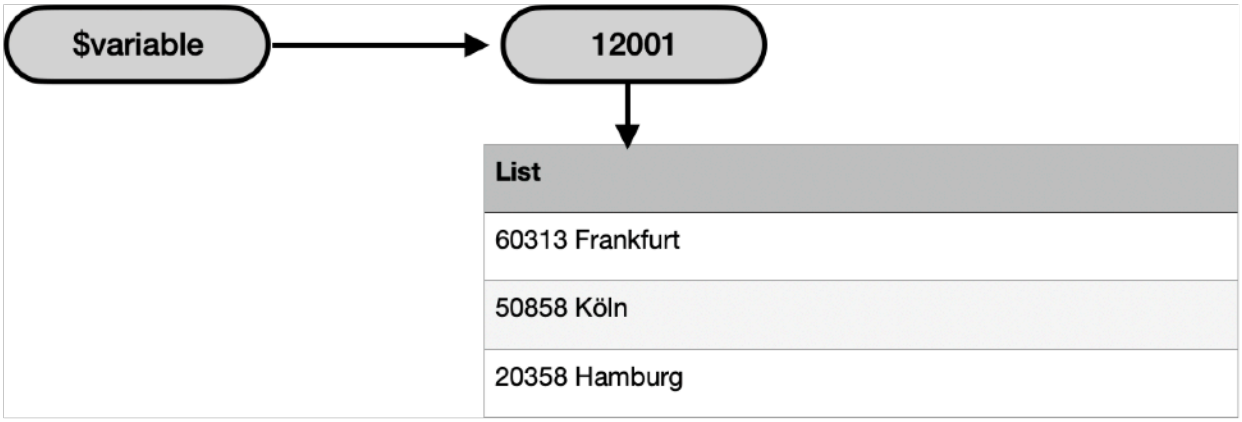
When you pass it around, you pass the whole list and every append copies the whole list.

The variable contains the list directly:



QuickList

To improve a list, we add a little indirection. The plugin manages the list for you and in the variable is just the identifier for the list, usually a number.



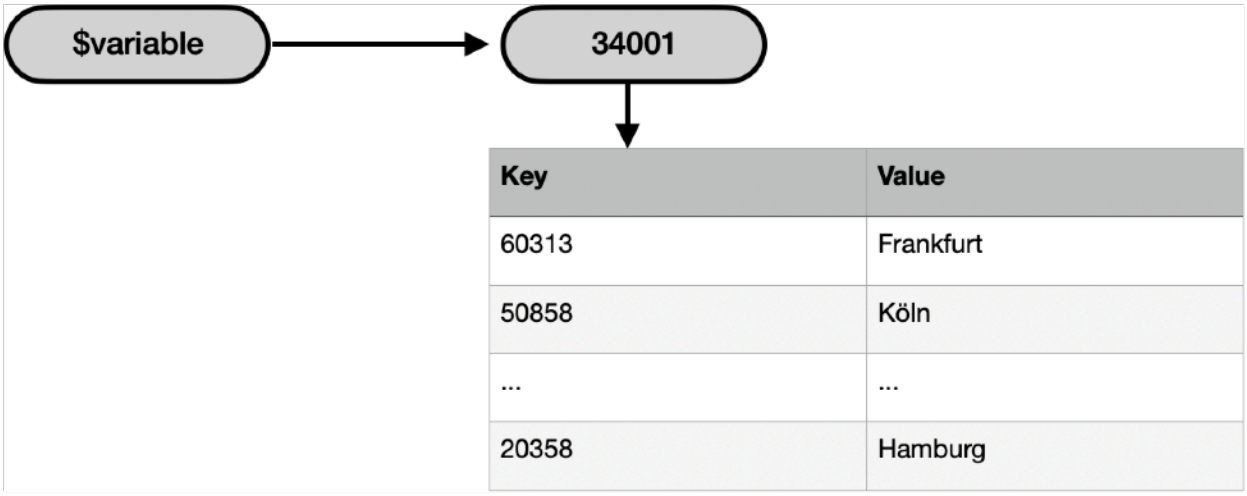
You create such a list with QuickList.New function. Then you can add values using QuickList.AddValue and when you need a value, check the QuickList.Count function and the QuickList.GetValue function. If you use these functions, you can speed up any operation using GetValue() in a loop.

If you need a quick way to look for the index of a value, please check the QuickList.IndexOfFirstValueMatching function. To fill a list, you can pass an existing list to QuickList.New function or fill it using SQL with QuickList.AddSQL function.

Dictionary

The dictionary stores values for keys. It uses an index and thus is a great way to quickly lookup values for keys. You can store for example zip codes and city names in a dictionary or the display names for UUID values.

You can store various values in the dictionary. Like you can put in artikel IDs as keys and then put in a list of values as value. This way you can for example have the price in there with description: "1234556" -> "Rose¶1.99€¶5,10,50". Then when you lookup the article, you get the list. This list then contains for example the bulk price quantities "5,10,50". The same (or second) dictionary then has an entry "1234556-10" to get the bulk price for 10 times the product: "Rose¶1.89¶". By storing this lookup tables in memory, you can access them anywhere very quickly without database access.



Key	Value
60313	Frankfurt
50858	Köln
...	...
20358	Hamburg

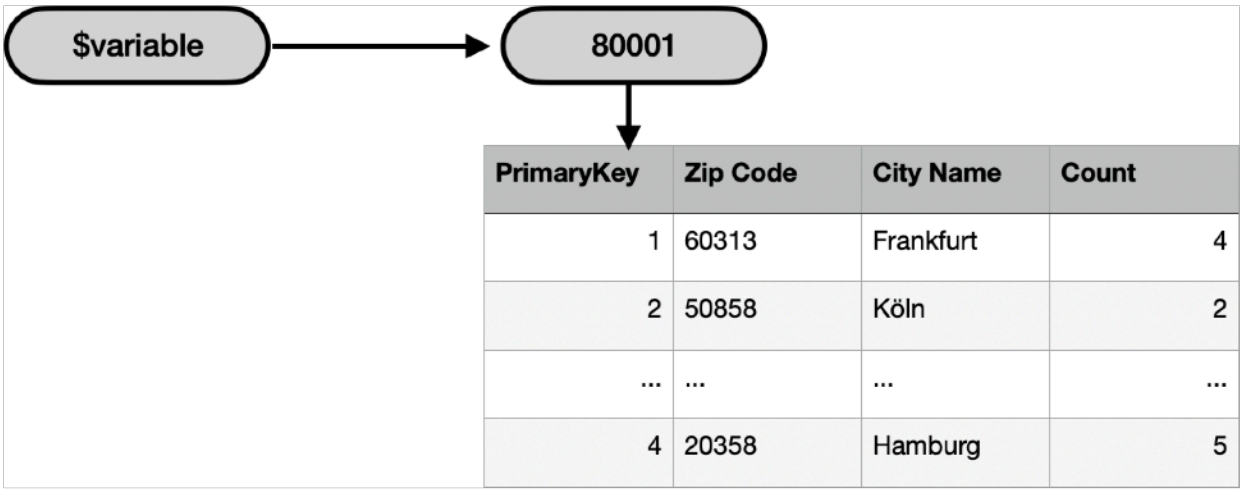
You create a dictionary with the Dictionary.Create function. Use Dictionary.SetValueForKey function to add one value or use Dictionary.SetList to put in a list of values with a delimiter. Using Dictionary.AddSQL you can fill the dictionary using SQL by loading two fields from a record. The SQL may do calculations like concat multiple fields to make one value.

If you like to query one value, you can use the Dictionary.ValueForKey function or alternatively the Dictionary.Lookup function. The last one returns a default value instead of an error if the key doesn't exist.

Values in the dictionary are stored in their original data type and avoid the conversion to text. Use Dictionary.ValueTypeForKey or FM.DataType to check what data type a value has.

Matrix

While a dictionary has only one value for a key, a matrix is a 2D array which a matrix of values. It can basically store a whole table in memory.



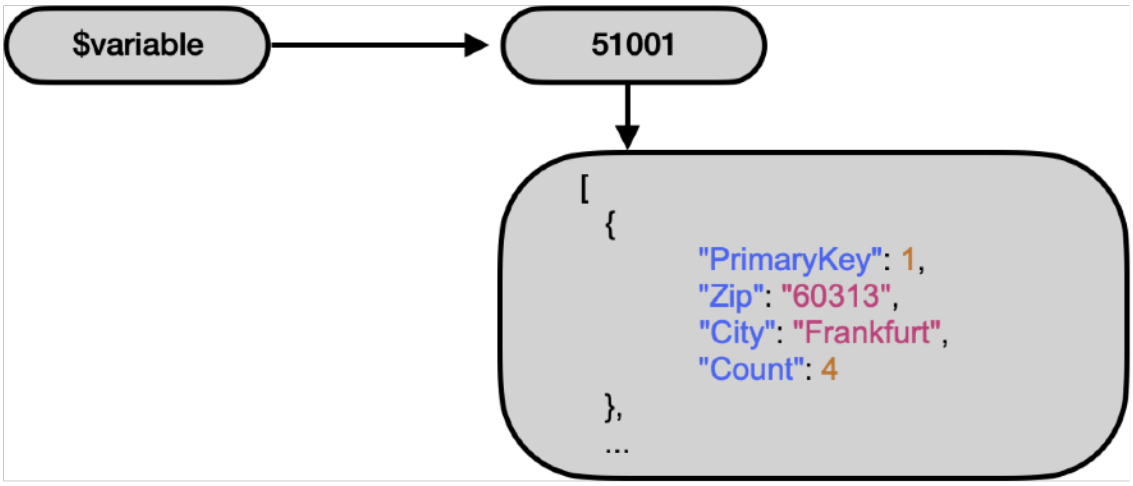
PrimaryKey	Zip Code	City Name	Count
1	60313	Frankfurt	4
2	50858	Köln	2
...	...	...	...
4	20358	Hamburg	5

Your variable contains the reference number and our plugin manages the memory for the matrix. Create it using Matrix.New function and add values as needed. You may fill a matrix quickly using Matrix.NewWithSQL function. Then at any time, you can lookup a value with Matrix.GetValue or Matrix.Lookup functions. Same as the dictionary, we store the values in their original data types without converting them to text.

To quickly find a value in one column, we'll add Matrix.IndexOfFirstValueMatching function for next plugin version.

JSON References

You may use JSON already, but with our plugin you can use JSON references. Instead of passing JSON around, we pass just the identifier and the plugin keeps the JSON in memory. This avoids parsing the same JSON text over and over again.



You usually may start with JSON.Parse to parse some JSON. Or start with a blank array with JSON.CreateArrayRef function. Then you get an identifier and to add something you pass in the reference and the value to add to JSON.AddStringToArray function.

At any time you may pick something from such a JSON with our normal JSON functions. They all allow you to pass a JSON reference number instead of a JSON text.

SQL Result References

If you use FM.SQL.Execute function, we store the result of the query in memory and give you a reference number. That is similar to matrix and you can convert from SQL result reference to a Matrix with FM.SQL.ToMatrix function.

The records are stored in their original data type and you can retrieve them using functions like FM.SQL.Field to get a value.

Please don't hesitate to contact us if you have questions.