# Data Serialization (Let Notation)

Let notation is proposed as a best practice for data serialization (Wikipedia) within FileMaker. Let notation is a text format for representing dictionary (name-value pair) data structures in FileMaker. It is most commonly used for passing script parameters and results, but developers have found other applications as well. Similarly to JSON, it is based on a subset of FileMaker calculation syntax — a subset of the Let function, specifically. This makes it easy to parse in FileMaker, human-readable and easy to learn for FileMaker developers, and therefore a solid basis for a common and consistent interface for message passing (Wikipedia) and managing data within scripting and other operations.

The Let function can be used to set variables in a FileMaker calculation which have a larger scope than the Let function or the calculation itself.

```
Set Variable [$!; Value:Let ( [ $foo = "bar" ] ; "" )]
Set Variable [$result; Value:$foo]
# variable $result contains the value "bar"
```

When combined with the Evaluate function, this gives developers an easy way to encode and retrieve series of name-value pairs.

```
Set Variable [$dictionary; Value:"$name = \"value\" ;¶$foo = \"bar\" ;¶"]
Set Variable [$calculation; Value: "Let ( [ " & $dictionary & " $! = $! ] ; \"\" )"]
Set Variable [$error; Value:EvaluationError ( Evaluate ( $calculation ) )]
# local variable $name now holds value "value"
# local variable $foo now holds value "bar"
```

If the $error variable contains a value other than 0 after the last two script steps above are executed exactly as written, then the $dictionary variable did not contain valid Let notation. However, if $error does contain 0, that doesn't necessarily indicate that $dictionary did contain best-practice Let notation. There are additional recommendations that facilitate parsing a dictionary and encourage programming best practices.

## Implementation Recommendations

Name-value pairs begin with a locally-scoped variable name and no leading whitespace.

**KEY POINT:** While it is certainly possible to set a global variable (prefixed with "$$", as in "$$VARIABLE") with FileMaker's Let function, this is discouraged in Let notation. The ultimate purpose of Let notation is to encode data, not to set variables; and it is the prerogative of the software receiving and parsing the data to decide what to do with it.

(The #Assign custom functions provided on this site go so far as to coerce global names to local variables.) However, the Evaluate-Let code pattern described above is efficient enough to be worth accommodating. Calculation-scoped variables can exist as part of a Let notation dictionary without interfering with other syntax, but this is only bloat, as Let notation parsers should not retrieve these values. This ensures consistent behavior between parsers using the simple Evaluate-Let pattern and parsers that extract values one at a time.

Each name-value pair ends with a semi-colon and carriage return (";¶" or Char ( 1300059 )), and there should be no other carriage return character anywhere in the name-value pair. Text containing carriage returns can be encoded using FileMaker's Quote function. This facilitates parsing, such as to retrieve a particular named value without setting variables as with the Evaluate-Let construction. Note that the final name-value pair in a dictionary should also end with ";¶", which makes it easy to add pairs to a dictionary using basic text concatenation. Also, if the last pair did not end with at least a semi-colon, the Evaluate-Let construction above would fail.

```
Set Variable [$dictionary; Value:"$name = \"value\" ;¶"]
Set Variable [$dictionary[2]; Value:"$foo = \"bar\" ;¶"]
Set Variable [$dictionary[3]; Value:$dictionary & $dictionary[2]]
```

When the same name appears more than once in the same dictionary, the last occurrence is used when retrieving the value. This is the native behavior of the Evaluate-Let pattern, so other parsing methods should adopt the same behavior to be consistent. This can be used to over-write or erase values from a dictionary by concatenating new name-value pairs to the end, or to define overridable default values for script parameters.

```
"$overwriteMe = \"old text\" ;¶"
& "$overwriteMe = \"new text\" ;¶"

& "$eraseMe = \"still here!\" ;¶"
& "$eraseMe = \"\" ;¶"

& "$parameter = \"default value\" ;¶"
& Get ( ScriptParameter )
```

Named values may be expressed as a calculation. For example, this can be used to ensure that the data type of a value is faithfully reproduced after parsing.

```
"$text = \"text\" ;¶"
& "$number = 1.234e-5 ;¶"
& "$timestamp = Timestamp ( Date ( 3 ; 25 ; 2013 ) ; Time ( 22 ; 23 ; 48 ) ) ;¶"
```

Referencing other values from a dictionary in such expressions is discouraged, since this can lead to inconsistent results between parsers based on Evaluate-Let and parsers based on text manipulation functions. Environmental status functions (Get functions, Location functions, etc.) are also discouraged, as such usage is inconsistent with Let notation's purpose of encoding data rather than executable instructions.