Custom Functions » Script Parameter Interface

Overview

One of the common methods for passing parameters to scripts is via name-value pair or dictionary, which can make code easier to read and understand. This differs from many other languages which use ordinal parameters. We recommend name-value pairs as a best practice rather than other FileMaker methods analogous to ordinal parameters, such as return-delimited lists, due to a number of advantages:

- The name-value pairs are not sensitive to the order of the pairs. Ordinal parameters must exactly match what a script is expecting.
- Name-value pairs are more self-documenting. The names give clues to what the values represent or might be used for without necessarily having to read the script that will use them.
- With name-value pairs, optional parameters can be left unspecified, whereas care may be needed to leave gaps for unspecified optional parameters in delimited lists.

Script parameters are the most common application for these structures, but they are useful for other purposes as well.

This best practice suggests only the names of custom functions for working with name-value data structures and how the functions should behave when used together. The particulars of implementation are not specified. However, a reference implementation is available in the FileMakerStandards.org GitHub repository. A different implementation, even one using a different data serialization format, that still follows the behaviors described below would still match this best practice.

Core Functions

These functions implement the basics of adding data to and retrieving data from dictionary data structures.

(name ; value)

The # (name ; value) function creates a name-value pair. A dictionary data structure can be created by concatenating several calls to #() as if they were plain text. Name-value pairs can be nested.

```
# ( "name" ; "value" )
& # ( "outerName" ;
        # ( "innerName" ; "inner value" )
)
```

Named values can be over-written or effectively erased by concatenating a call to #() to the end of a dictionary using the same name and a different value. This works because the #Get and #Assign functions will always respect the *last* instance of a named value in a dictionary.

```
# ( "name" ; "value" )
& # ( "foo" ; "bar" )
& # ( "name" ; "new value" )
& # ( "foo" ; "" ) // Assigning empty value effectively deletes "foo"
```

This last-value-wins behavior can also be used to set default values for optional parameters.

```
# ( "optionalParameter" ; "default value" )
& Get ( ScriptParameter )
```

By placing the defaults before the actual parameters, any values set by the actual script parameter will override the defaults.

#Assign (parameters)

The #Assign function parses a dictionary into locally-scoped script variables. The name from each name-value pair is used as the variable name, and the value from each pair is used as that variable's value.

```
#Assign ( # ( "name" ; "value" ) ) // variable $name assigned "value"
```

The #Assign function returns True (1) if there was no error detected while assigning the values to variables, and returns False (0) otherwise. If there was an error detected, FileMaker's error code is assigned to the $#Assign.error variable.

#Get (parameters ; name)

The #Get function returns a named value from a dictionary.

```
#Get ( # ( "name" ; "value" ) ; "name" ) // = "value", $name is unaffected
```

Unlike the #Assign function, #Get will not modify any variables. This can be useful when a value only needs to be used in one calculation, or to assign a value to a variable with a different name.

Utility Functions

These functions are less fundamental to working with dictionary data structures than the core functions, but experience has demonstrated that this functionality can be indispensable in practical applications.

#AssignGlobal (parameters)

The #AssignGlobal function parses a dictionary into global variables.

```
#AssignGlobal ( # ( "name" ; "value" ) ) // variable $$name assigned "value"
```

The #AssignGlobal function returns True (1) if there was no error detected while assigning the values to variables, and returns False (0) otherwise. If there was an error detected, FileMaker's error code is assigned to the $#AssignGlobal.error variable.

#Filter (parameters ; filterParameters)

The #Filter function returns a dictionary containing only those name-value pairs where the name is included in the return-delimited list filterParameters.

```
#Assign ( #Filter (
        # ( "name" ; "value" )
        & # ( "otherName" ; "otherValue" )
        & # ( "foo" ; "bar" ) ;
        List ( "name" ; "otherName" )
) ) // $name and $other name are assigned "value" and "otherValue", respectively; $foo is unaffected
```

This function can prevent an "injection" of unexpected variables that might cause problems.

#GetNameList (parameters)

The #GetNameList function returns a list of names from all name/value pairs in parameters. This is useful when you don't know what names exist, and you want to iterate through all the name/value pairs that exist.

```
#GetNameList (
        # ( "name" ; "value" )
        & # ( "foo" ; "bar" );
)
= List ( "name" ; "foo" )
```

#Remove (parameters ; removeParameters)

The #Remove function returns a dictionary containing only those name-value pairs where the name is not included in the return-delimited list removeParameters. This is complementary to the #Filter function.

```
#Assign ( #Remove (
        # ( "name" ; "value" )
        & # ( "otherName" ; "otherValue" )
        & # ( "foo" ; "bar" ) ;
        List ( "name" ; "otherName" )
) ) // $foo is assigned "bar"; $name and $otherName are unaffected
```

ScriptOptionalParameterList (scriptNameToParse)

The ScriptOptionalParameterList function parses a script name, returning a return-delimited list of optional parameters for that script, in the order they appear in the script name. This function assumes that the script name conforms to the FileMakerStandards.org naming convention for scripts.

```
ScriptOptionalParameterList ( "Script Name ( required1 ; required2 ; { optional1 ; optional2 } )" )
= List ( "optional1" ; "optional2" )
```

This is useful to generate the argument used by the #Filter function to restrict variable assignment to parameters actually accepted by a script.

```
#Assign ( #Filter (
        Get ( ScriptParameter ) ;
        ScriptRequiredParameterList ( "" ) & ScriptOptionalParameterList ( "" )
) )
```

When the scriptNameToParse parameter is empty, the function will use the current script name.

```
ScriptOptionalParameterList ( "" ) = ScriptOptionalParameterList ( Get ( ScriptName ) )
```

ScriptRequiredParameterList (scriptNameToParse)

The ScriptRequiredParameterList function parses a script name, returning a return-delimited list of parameters required for that script, in the order they appear in the script name. This function assumes that the script name conforms to the FileMakerStandards.org naming convention for scripts. This is useful to generate the argument used by the VerifyVariablesNotEmpty function to validate that all required parameters have values. When the scriptNameToParse parameter is empty, the function will use the current script name.

```
ScriptRequiredParameterList ( "Script Name ( required1 ; required2 ; { optional1 ; optional2 } )" )
= List ( "required1" ; "required2" )
```

VerifyVariablesNotEmpty (nameList)

The VerifyVariablesNotEmpty function returns True (1) if each of the parameters in nameList has been assigned to a non-empty variable of the same name. The function returns False (0) if any variable defined by nameList is empty. This is useful for verifying that each parameter required by a script has been successfully assigned before proceeding.

```
VerifyVariablesNotEmpty ( ScriptRequiredParameterList ( "" ) )
```

Array functions

A standard format for array's has not yet been agreed upon, so a page has been created for it in the Proposals section: Custom Functions » Arrays

Legacy functions

Deprecated

The functions below this point have been deprecated in favor of the more efficient functions above. Use of these functions is optional, but not recommended.

These functions are documented for backwards compatibility. These functions are all equivalent to simple combinations of other functions, which can make the functionality more self-describing, especially for developers who may be unfamiliar with them.

#AssignScriptParameters

The #AssignScriptParameters function will assign all named values in the script parameter to local script variables of the same name. If any parameters indicated as required by the script name are empty, the function returns False (0); the function returns True (1) otherwise.

```
If [not #AssignScriptParameters]
        Exit Script [Result:# ( "error" ; 10 ) // Requested data is missing]
End If
```

A combination of the #Assign, VariablesNotEmpty, and ScriptRequiredParameterList functions is the preferred way to replicate this behavior:

```
Set Variable [$~; Value:#Assign ( Get ( ScriptParameter ) )]
If [not VerifyVariablesNotEmpty ( ScriptRequiredParameterList ( "" ) )]
        Exit Script [Result:# ( "error" ; 10 ) // Requested data is missing]
End If
```

This approach enables greater flexibility in defining what variables are required and how those variables are assigned.

#AssignScriptResults

This function is exactly equivalent to this calculation:

```
#Assign ( Get ( ScriptResult ) )
```

#GetScriptParameter (name)

This function is exactly equivalent to this calculation:

```
#Get ( Get ( ScriptParameter ) ; name )
```

#GetScriptResult (name)

This function is exactly equivalent to this calculation:

```
#Get ( Get ( ScriptResult ) ; name )
```