Custom Functions » Arrays

An Array, or ordered list of values, is a common data structure in programming languages. This page is dedicated to the discussion of a standard format for array's within FileMaker, along with a set of custom functions used to create/access data in this format.

Dan Smith's Proposal

I plan to use the same method of encoding each values in an array, as used by the # (name ; value) function. The only difference is the name will be omitted, which will result in a return-delimited list of encoded values.

#List (value)

Encode a value in a manner that preserves the data type and escapes all returns/new lines, then append a trailing return.

```
#List ( "value1" )
& #List (
        #List ( "subValue" )
        &  #List ( "subValue2" )
)
```

will produce:

```
"value1"
"\"subValue\"¶\"subValue2\"¶"
```

#ListGet (listOfValues ; valueNumber)

Retrieve a value from a list created with the *#List (value)* function. Preserve the data type and all special characters of the original value.

```
#ListGet ( #List ( "value1" ) & #List ( "value2" ) ; 1 ) = "value1"
#ListGet ( #List ( "" ) & #List ( "value2" ) ; 2 ) = "value2"
#ListGet ( #List ( "value1" ) & #List ( "value2" ) ; 3 ) = ""
#ListGet ( "This is not a valid #List string" ; 1 ) = ""
```

Other Array Operations

Since *#List (value)* produces a return-delimited list, all of FileMaker's list-related functions can be used to manipulate an array:

```
Let ( [
        ~array =
                #List ( "value1" )
                & #List ( "value2" )
                & #List ( Get ( CurrentDate ) )
                & #List ( 123456 )
        ;
        ~arrayCount = ValueCount ( ~array ) ;
        ~dwindlingArray = RightValues ( ~array ; ~arrayCount - 1 )
] ;
        ~arrayCount = 4
        and
        ~dwindlingArray =
                #List ( "value2" )
                & #List ( Get ( CurrentDate ) )
                & #List ( 123456 )
) = True
```

Jeremy Bante's Proposal

I propose an alternate technique for handling array data that strictly follows the Let notation format. Let notation is defined by FileMaker's built-in behaviors — if FileMaker variables support something, so does Let notation. This means that Let notation already supports value names using FileMaker's repeating variable syntax. Most of the custom functions supporting this proposal are merely syntactic sugar for behaviors we can already implement with the existing parameter interface functions. Any solution already using Let notation does not need any extensions to be compatible with such arrays. However, this format is less well suited for manipulation with FileMaker's built-in functions for manipulating return-delimited lists.

#Array (name ; index ; value)

The #Array function is syntactic sugar for the # (name ; value) function using repeating variable names. As with the # function, the data type of the encoded value will be preserved.

```
#Array ( "name" ; 1 ; "value 1" )
& #Array ( "name" ; 2 ; "value 2" )
& #Array ( "name" ; $n ; "value N" )
=
# ( "name" ; "value 1" )
& # ( "name[2]" ; "value 2" )
& # ( "name[" & $n & "]" ; "value N" )
```

Values created with the #Array function are accepted by the #Assign function, and would be set to locally scoped variables.

```
Let ( [
        ~dictionary =
                #Array ( "name" ; 1 ; "value 1" )
                & #Array ( "name" ; 2 ; "value 2" )
                & #Array ( "name" ; 3 ; "value 3" ) ;
        ! = #Assign ( ~dictionary )
] ;
        $name = "value 1"
        and $name[2] = "value 2"
        and $name[3] = "value 3"
)
```

#ArrayGet (parameters ; name ; index)

The #ArrayGet function is syntactic sugar for the #Get function using repeating variable names.

```
#ArrayGet ( $parameters ; "name" ; $i ) = #Get ( $parameters ; "name[" & $i & "]" )
```

#ArrayFromList (name ; valueList)

The #ArrayFromList function is a convenience function for creating a repeating variable array in Let notation from an existing return-delimited list.

```
Let ( [
        ~list = List ( "value 1" ; "value 2" ; "value 3" ) ;
        ~dictionary = #ArrayFromList ( "name" ; ~list ) ;
        ! = #Assign ( ~dictionary )
] ;
        $name = "value 1"
        and $name[2] = "value 2"
        and $name[3] = "value 3"
)
```