


LibreOffice RefCard

LibreOffice BASIC

Structured Data Types

v. 1.01 – 11/03/2019



Written with LibreOffice v. 5.3.3 – Platform : All

Arrays

Used for grouping related items. The items are indexed (Long).

Dimensions

An array may have several sizes/dimensions (60 max.).

Declaring Arrays

⚠ The index base is 0 (zero)! Change it with Option Base 1 (rarely needed).
Below: ix = index.

Static arrays	Size is known at write-time
1 dimension (« vector »)	
Dim T(ixMin To ixMax) As SomeType	Dim T(1 To 12) As SomeType 1 dimension of 12 memory places, indexed from 1 to 12.
or Dim T(ixMax) As SomeType	Dim T(9) : 1 dimension of 10 items, indexed from 0 to 9.
Several dimensions	
Dim T(ixMin1 To ixMax1, ixMin2 To ixMax2) As SomeType	Several dimensions (example with 2). Dim T(1 To 12, 1 To 31) As Integer 2 dimensions (reserves 12 and 31 items)
or Dim T(ixMax1, ixMax2, ...) As SomeType	Dim T(2,4) As String : 2 dimensions. 3 items for the 1 st , 5 for the 2 nd .

Dynamic arrays

Dynamic arrays	Size is known at run-time
Dim MyArr() As SomeType Dim MyArr As Variant	} Declares an unknown dimension array. } ReDim is necessary in the future.

☞ When declared as a Variant type, an array may home items of different types.

Nested Arrays/Jagged Arrays

Or array of arrays. Ex: used to access Calc range data values (.DataArray property).

An encompassing array (variant type) has arrays of data as items:

```
Dim MyArr As Variant
MyArr = Array(Array(1, 2, 3), Array(10, 20, 30), Array(7, 8, 9))
MyArr(0)(0) is 1; MyArr(2)(2) is 9, etc.
```

Accessing An Array Item

```
By index :
Dim MyArr(9) As Integer           Dim MyArr(11, 31) As Integer
MyArr(5) = 123                   MyArr(5, 28) = 123
```

Array Functions And Instructions

Option Base 1	(instruction at module start – applies to the current module) Forces array indices to start at 1 instead of 0.
IsArray()	Returns True if the variable is an array type. OK = IsArray(MyArr)
Array()	Returns an initialized array from discreet values. MyArr = Array("A", 2, Now()) 'here, variant array
ReDim	(instruction) Re-dimensions an array With data loss: ReDim MyArr(dimension) Without data loss: ReDim Preserve MyArr(dimension)
Erase	(Instruction) Deletes an array contents. Erase MyArr In case of a dynamic array, frees memory.
LBound()	Returns an array lower bound. Defaults to the 1 st dimension, otherwise specify : LBound(MyArr, 2)
UBound()	Returns an array upper bound (same condition as LBound). ☞ An array has no defined dimension if UBound(MyArr) = -1 and LBound(MyArr) = 0
Split()	Creates an array (vector) by splitting a string on a delimiter. A=Split("C:\file.txt", "/") → A(0)="C:", A(1)="file.txt"
Join()	Reverts the Split() operation: merges an array items (vector) to get a string. Join(A, " ") → "C: file.txt"

Checking An Array Validity

MyArr is an array variable. It can be manipulated as such if it passes the three tests:

- Does the variable exists? Not IsNull(MyArr)
- Is it an array? IsArray(MyArr)
- Is the array dimension defined? UBound(MyArr) >= LBound(MyArr)

Browsing A 1-Dimension Array (vector)

By Index

```
Dim MyArr(9) As Integer, i As Long
For i = LBound(MyArr) To UBound(MyArr)
    Print MyArr(i)
Next i
```

By Items

```
Dim Val As 'compatible type with the array items
For Each Val In MyArr
    Print Val
Next
```

Browsing A 2-Dimension Array

```
Dim MyArr(2, 4) '3 rows, 4 columns
Dim i As Long, j As Long
For i = LBound(MyArr) To UBound(MyArr)
    For j = LBound(MyArr, 2) To UBound(MyArr, 2)
        Print MyArr(i, j)
    Next j
Next i
```

Sorting Arrays

☞ No such predefined functionality (look for QuickSort on the web).

Copying Arrays

General Case A (For . . .Next) loop copying values from one array to the other. Might be looong!

Hint Assign then ReDim Preserve:
Array2 = Array1 'both var. -> same data
ReDim Preserve Array2(ADim) '-> now 2 different data sets
☞ Only applies to simple types arrays (non object).

Using Arrays With Subprograms

As A Sub Parameter

```
Sub UseArray(ByRef MyArr() As String)
    Dim MyArray(9) As String
    UseArray(MyArray)
End Sub
```

As A Function Result

```
Function GetArray() As Integer()
    GetArray = SomeIntegerArray()
End Function
Return from the function
Dim MyArray As Integer
MyArray = GetArray()
```

Arrays And Spreadsheet Ranges

(see RefCard #3)

Custom Types

Allow to aggregate several values (members) within a unique data type. Data manipulation, parameter passing and function returns are simplified.

⚠ Members may be of any type, simple or custom, but not array.

Type Declaration

```
Type MyCustomType
    SomeMember As SimpleType
    OtherMember As OtherSimpleType
End Type
Type MyEvent
    Name As String
    DateTime As Date
End Type
```

Declaring Custom Type Variables

Dim SomeVar As MyCustomType
⚠ **Limitation:** a custom type is only visible in the very module it is declared. Thus, the As MyCustType is only possible within the same module where MyCustType is declared. See Factory/Accessor functions.

Using Custom Type Variables

```
Assigning MyVar.SomeMember = SomeValue
Reading SomeVar = MyVar.OtherMember
```

The With Keyword

```
Shortens items references With MyVar
.SomeMember = SomeValue
End With
☞ Note the dot presence.
```

Collections

Structure for fast access to indexed data using keys.

A collection is handled as an Object type.

Stored items can be of any type, incl. Object.

⚠ The key is of String type. In a collection, every key is unique.

Declaring A Collection

```
Dim oColl As New Collection
```

Adding An Item

```
With a key
oColl.Add(Item, "TheKey")
☞ Future read by key or by index.
☞ The key is not case-sensitive.
☞ The insertion position may be specified with Before/After :
oColl.Add(Item, "TheKey", After:="Key0")
oColl.Add(Item)
☞ Future read by index only.
```

Checking An Item Exists

Try reading the key and handle the possible error. (see below).

Getting An Item

```
By key Value = oColl("TheKey")
By index Value = oColl.Item(Index)
```

Replacing An Item

Same key, new item value.
☞ Delete then Add.

Deleting An Item

```
By key oColl.Remove("TheKey")
By index oColl.Remove(Index)
```

Deleting All Items

```
ReDim oColl As New Collection
```

Counting Items

```
NumItems = oColl.Count
```

Checking An Item Existence

Try reading the key and handle the possible error.

```
Function ExistsItem(ByRef pColl As Object, pKey As String) As Boolean
    Dim Item As Variant, Exists As Boolean
    On Local Error Goto ErrHandler
    Exists = False
    Item = pColl(pKey) 'if pKey not found -> ErrHandler:
    Exists = True
    ErrHandler:
        'do nothing
    ExistsItem = Exists
End Function
```

Browsing A Collection

You may get all collection items by browsing them.

☞ It is not possible to browse by keys.

By Items Index

```
For i = 1 To oColl.Count
    Value = oColl.Item(i) 'reading the data
Next i
```

By Items Direct Access

```
Dim AnItem As 'type compatible with the collection items
For Each AnItem In oColl
    'do smthg with AnItem (data)
Next
```

Creating Classes In Basic

Embryo of object oriented programming (OOP).

☞ **Limitations: no inheritance (use delegation), no polymorphism!**

A LibreOffice Basic class might thus be seen as a glorified custom type to which we add some behavior (functions and subprograms).

Vocabulary

Class module	Code module that contains the class declares.
Class	A type that allows to create (instanciate) object variables.
Events	Two events may be intercepted: object creation and destruction .
Member	A variable that is internal to a class (not meant to be used outside).
Property	Reflects an object state .
Method	Realizes some action on/with the object.
Instance	The object created from a class type.

Specifying A Class

A class specifications (members, events, properties, methods) are all written within a single dedicated code module. In LibreOffice Basic, this module only differs from standard modules by its initial options.

☞ **Hint: use a naming convention for class modules.**

Initial Options

A class module should start with the options :

```
Option Explicit 'as usual
Option Compatible
Option ClassModule
```

Member Variables

They are internal to the class, thus declared as Private.

☞ **A class members should never be called from the outside through the instance but only using properties created for that purpose.**

```
Private mName As String
Private mSheet As Object
```

Events

These are two internal subprograms, thus declared as Private.

```

Constructor Private Sub Class_Initialize()
                To initialize the object being created.
Destructor Private Sub Class_Terminate()
                To cleanup internal items of an object being destroyed.
                ☞ Security breach. It is highly advised to not have this destructor
                within your classes: because of an implementation bug in VisualBa-
                sic, Class_Terminate() is a security breach and, as such, is re-
                jected by antivirus (see CVE-2018-8174).
```

☞ **Limitation:** these subprograms can't receive parameters.

Properties

Property = object **state**.

They are meant to be visible from the outer, thus declared as Public.

```

Reading (Get) (any data type, incl. object)
                 Public Property Get Name() As String
                 Name = mName
                 End Property
```

```

Writing (Let) (all data types, except objects)
                 Public Property Let Name(ByRef pName As String)
                 mName = pName
                 End Property
```

```

Writing (Set) (objects only)
                 Public Property Set Sheet(ByRef pSheet As Object)
                 Set mSheet = pSheet
                 End Property
```

☞ **A property may be prematurely exited using the Exit Property instruction.**

A class may contain read-only or read-write properties. Write both Get and Let/Set properties whenever necessary.

Properties may access the class members, properties and methods.

Methods

Method = **action** on/from the object.

These are Sub and Function specific to the class. They may be internal (Private) or visible (Public). They are written just like standard Sub and Function, introduced with the Public or Private keyword. They have full access to the class members and properties.

Using Classes In Basic

Declaring/Creating An Object

```

Immediate instantiation      Set MyObject = MyClass
Differed instantiation      { Dim MyObject As New MyClass
(at 1st call)                 { MyObject = New MyClass
```

The class constructor is called at the time of the object instantiation.

Declaring an object As New MyClass is not possible out of the library in which the MyClass module exists. You'll have to declare the object As Object.

☞ **Limitation:** a class is only visible *in the very library* where the class module exists. See Factory/Accessor functions.

Accessing An Object Properties And Methods

Object items access syntax: object.property or object.method

The With...End With syntax may be used, like for custom types.

Freeing An Object

When an object is not useful anymore, you may destroy it: Set oMyObject = Nothing
The class destructor is called at that time.

☞ **This instruction is not strictly necessary in Sub or Function, as local variables are destroyed at exit but the destruction time is not under your control then.**

The Set oMyObject = Nothing instruction:
– Shows the intention.
– Ensures the controlled object destruction time.

Factory/Accessor Functions

Custom Types And Classes Visibility Question

• A **custom type** is only visible in the **module** where it is declared.

• A **class** is only visible in the **library** where it is declared.

A workaround is to prepare a **factory** function (aka accessor) to create such variables.

Such function may be called from any other module or library.

	☞ Use	☞ Visibility	☞ Factory created in	☞ Declared
Cust. Type	As MyType	Same module	Same module	As Variant
Class	As MyClass	Same library	Same library, other module	As Object

Creating The Factory Function

☞ **The factory function can also be used for variable initialization.**

Custom Types

```

Function CreateMyCustomType() As MyCustomType
    Dim oVar As MyCustomType
    CreateMyCustomType = oVar
End Function
```

Classes

```

Function CreateMyClass() As MyClass
    Dim oVar As New MyClass
    CreateMyClass = oVar
End Function
```

Using The Factory Function

Custom Types

```

Dim MyVar As Variant
MyVar = CreateMyCustomType()
```

Classes

```

Dim oMyVar As Object
oMyVar = CreateMyClass()
```

Credits

Author : Jean-François Nifenecker – jean-francois.nifenecker@laposte.net

We are like dwarves perched on the shoulders of giants, and thus we are able to see more and farther than the latter. And this is not at all because of the acuteness of our sight or the stature of our body, but because we are carried aloft and elevated by the magnitude of the giants. (Bernard of Chartres [attr.])

History

Version	Date	Comments
1.0	04/20/2019	First version
1.01	11/03/2019	Fixes

License

This RefCard is placed under the
Creative Commons BY-SA v3 (fr) license
Information

<https://creativecommons.org/licenses/by-sa/3.0/fr/>

