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 (null). Change it with Option Base 1 (rarely needed).
- Below 1x = index.

Static arrays
- Size is known at write-time

<table>
<thead>
<tr>
<th>Dim T(1) As SomeType</th>
<th>Dim T(1 To 12) As SomeType</th>
</tr>
</thead>
<tbody>
<tr>
<td>Dim T(1, 2) As SomeType</td>
<td>Dim T(1 To 12, 1 To 31) As Integer</td>
</tr>
</tbody>
</table>

Dynamic arrays
- Size is known at run-time

<table>
<thead>
<tr>
<th>Dim MyArr() As SomeType</th>
<th>Dim MyArr(i) As SomeType</th>
</tr>
</thead>
<tbody>
<tr>
<td>Dim MyArr() As Variant</td>
<td>Dim MyArr(i, j) As SomeType</td>
</tr>
</tbody>
</table>

**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 items:
  - Dim MyArr As Variant
  - MyArr = Array(Array(1, 2, 3), Array(10, 20, 30), Array(7, 8, 9))
  - MyArr(8)(8) = 1; MyArr(2)(2) = 9, etc.

**Accessing An Array Item**

- By index:
  - Dim MyArr() As Integer
  - Dim MyArr(i) As Integer

- By Dim:
  - Dim MyArr As Variant
  - Dim MyArr = Array(Array(1, 2, 3), Array(10, 20, 30), Array(7, 8, 9))

**Array Functions And Instructions**

- Option Base 1 (instruction at module start – applies to the current module)
- Function 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, Nam1) 'here, variant array
- Redim (instruction) Re-dimensions an array
  - Dim T(1 To 12, 1 To 31) As Integer
  - 2 dimensions (reserves 12 and 31 items)
  - Dim T(1 To 12) As SomeType
  - 1 dimension of 12 items, indexed from 1 to 12.
- Erase MyArr (Instruction) Deletes an array contents.
- Erase MyArr
- LBound() Returns an array lower bound.
- LBound(MyArr) = 1
- UBound() Returns an array upper bound (same condition as LBound).
- UBound(MyArr) >= LBound(MyArr)
- Split() Creates an array (vector) by splitting a string on a delimiter.
- A.Split(",",";") = Split(",",";",",") + A(0) + C(0), ",",(0) + 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?
  - MyArr is an array type.
  - Is the array dimension defined?
- Not IsArray(MyArr)
- UBound(MyArr) >= LBound(MyArr)

**Browsing A Collection**

- For Each Val in MyArr
  - Print Val

**Browsing A 1-Dimension Array (vector)**

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

**Browsing A 2-Dimension Array**

- Dim MyArr(2, 4) As Integer
- For i = 1 To UBound(MyArr)
- Print MyArr(i, j)
- Next j

**Arrays And Spreadsheet Ranges**

- (see RefCard #3)

**Custom Types**

- Type Declaration
  - New data type.
  - Dim T As Type
  - Dim T(1 To 12) As SomeType
  - Size is known at write-time

- Declaring Custom Type Variables
  - Dim MyCustType As MyCustomType
  - Limitation: a custom type is only visible in the very module it is declared. Thus, the As MyCustType key is only possible within the same module where MyCustomType is declared.

- Using Custom Type Variables
  - With MyVar
    - MyVar.SomeMember = SomeValue
    - 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.
  - Without a key
    - oColl.Add(Item)

- Deleting An Item
  - With Index
    - oColl.Remove("TheKey")
    - Future read by index only.
  - Without Index
    - oColl.Remove(item, "TheKey")

- Replacing An Item
  - With key
    - oColl("TheKey") = SomeValue
    - Same key, new item value.

- 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.
  - (see below).

- Getting An Item
  - By key
    - Value = oColl("TheKey")
  - By index
    - oColl(Index)

- Replacing An Item
  - By key
    - oColl("TheKey") = SomeValue
  - By index
    - oColl(Index) = SomeValue

- Deleting An Item
  - By key
    - oColl.Remove("TheKey")
  - By index
    - oColl.Remove(Index)

**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 loopy!
- **Hint**
  - Assign then Redim Preserve:
    - Array = Array * both var. -> same data
    - Redim Preserve Array(2) 'now 2 different data sets
  - Only applies to simple types arrays (non object).

**Using Arrays With Subprograms**

- As A Sub Parameter
  - Sub with an array parameter
    - Sub UseArray(ByRef MyArr() As String)
    - UseArray(MyArray)

- As A Function Result
  - Function returning an array
    - Function GetArray() As Integer
    - GetArray = SomeIntegerArray()
    - Return from the function
      - Dim MyArray As Integer
      - MyArray = GetArray()

**Structured Data Types**

- Arrays And Spreadsheet Ranges
Creating Classes In Basic

Embryo of object oriented programming (OOP).
☞ Lockdown: 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 through 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 not to have this destructor within your classes: because of an implementation bug in VisualBasic, Class_Terminate is a security breach and, as such, is rejected 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 outside, thus declared as Public.

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

Writing (Let)
Public Property Let Name(ByVal pName As String)
name = pName
End Property

Writing (Set)
Public Property Set Sheet(ByVal 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 onFromFile 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 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 WithEvents...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.

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 oMyVar As Object
oMyVar = CreateMyClass()
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 [947].)

History

<table>
<thead>
<tr>
<th>Version</th>
<th>Date</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.0</td>
<td>04/20/2019</td>
<td>First version</td>
</tr>
<tr>
<td>1.01</td>
<td>11/03/2019</td>
<td>Fixes</td>
</tr>
</tbody>
</table>

License

This RefCard is placed under the CreativeCommons BY-SA v3 (fr) license

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