Overview

- Development time: Coding 20% - Maintaining 80%

Entities Naming

Variables, constants, subs, and functions must be identified.

- Allowed chars: unaccented chars, numbers, underscore (_).
- An identifier can't start with a number nor contain a space.
- Do not use any Basic keyword to name an entity!

Easy to read names: CamelCase, Name_with_separators

Explicit names: IsCell(), SaveSpreadsheet()

Comments

- *(apostrophe)* or REM. What follows is a comment.
- *Comments are also important as code! They apply to the current line only.*

Code Indent

Indented code is easier to read. Indent each code level with Space / Tabulation.

Continuing An Instruction On The Next Line

Last two chars or the first line: _ (space + underscore).

Variables

- By default, variable declares are not mandatory but this is dangerous (typos lead to double declares).
- Adding Option Explicit on top of a module forces variable declaration.

Declaring Variables

Variable: a memory place. A variable can be modified at run-time.

Simple Variables

Dim MyVar As AType

Ex: Dim MyText As String

Arrays

- Array indices are zero-based.

Dim MyArray(NumDim) As AType

Number of dimensions: any.

Dim MyArray(2,4) : 2 dimensions. 3 items for the 1st, 5 for the 2nd (*base 0*).

Dim MyArray(S) As AType

Declares a 1-dim array with 10 items (base 0).

Dim MyArray() As AType

Declares an array of unknown dimension. Calling ReDim will be required.

Accessing Arrays Items

MyArray(1, 3) = Value

Sets value to item 1, 3.

Inf = LBound(MyArray())

Lower bound [for dimension n].

Sup = UBound(MyArray())

Upper bound [for dimension n].

ReDim
dimensioning

ReDim MyArray(NewDim)

With data loss.

ReDim Preserve MyNewDim

Without data loss.

Emptying

Erase MyArray() Use ReDim with data loss.

Text

- Strings concatenation (fusion) (*“* + *“* is possible; better not use because of its ambiguity).

Comparisons (return True or False)

- Strictly equal
- Strictly lower
- Lower or equal
- Greater or equal
- Mind to floating numbers comparisons!

Mathematical Operators

Addition

Subtraction

Multiplication

Division

Mod

Integer division

Modulo (remainder of integer division)

Text

Raising to the power

Operators

Not

Or

And

Xor

Exclusive or

Comparisons

Comparison operators return True or False

Booleans

Not

Null

Empty

Boolean variables:

Null:

Invalid contents. Null assignment possible.

Empty:

Uninitialized variable yet.

Functions

Empty

Null

And

Or

Xor

Comparisons

Strictly upper or equal

Strictly lower or equal

Integer division

Modulo

(remainder of integer division)

Creating/Setting Object Variables

Dim MyObject As New AClass

Set MyObject = AClass

Initialization differed to the 1st setting.

Initialization is immediate

Declaring...

- **gives visibility**

Declaring an object in the current subprogram or module.

- In the current subprogram.

- Persistent value between calls.

- In the current object.

- In all libraries.

Type

- **Persistent value between programs!**

Declaring Types

- **Specifies the value set a variable can carry or a function return.**

- **Predefined Types**

<table>
<thead>
<tr>
<th>Type name</th>
<th>Description</th>
<th>Initialized to</th>
</tr>
</thead>
<tbody>
<tr>
<td>Boolean</td>
<td>Logical values True / False.</td>
<td>False</td>
</tr>
<tr>
<td>Byte</td>
<td>Integer numbers (8 bits), from 0 to 255.</td>
<td>0</td>
</tr>
<tr>
<td>Currency</td>
<td>Currency numbers (4 decimals).</td>
<td>0.0000</td>
</tr>
<tr>
<td>Date</td>
<td>Dates and hours. In fact, doubles.</td>
<td>0.0</td>
</tr>
<tr>
<td>Double</td>
<td>Floating numbers (64 bits).</td>
<td>0.0</td>
</tr>
<tr>
<td>Integer</td>
<td>Integer numbers (16 bits), -32768 to +32767.</td>
<td>0</td>
</tr>
<tr>
<td>Long</td>
<td>32 bits integers, -2^31 to 2^31 - 1.</td>
<td>0</td>
</tr>
<tr>
<td>Object</td>
<td>Objects. Allow to manipulate LibreOffice API objects.</td>
<td>Null</td>
</tr>
<tr>
<td>Single</td>
<td>Floating numbers (32 bits).</td>
<td>0.0</td>
</tr>
<tr>
<td>String</td>
<td>Text (0 to 65 545 characters).</td>
<td>“”</td>
</tr>
<tr>
<td>Variant</td>
<td>Any type, incl. object.</td>
<td>Empty</td>
</tr>
</tbody>
</table>

- **Every time a type is unspecified, Variant is implicit.**

LibOBasic_2_Overview.Flat_A4_EN_v120.cdt
Ex:

```
All commands start with
```

**Knowing UNO Menus Commands**

- Ex: MySub(ByRef AParam as Long, ByVal OtherParam As String, ...
  Optional ByRef SomeParam As String)
  
  **ByRef**
  _By reference_ (default). The parameter points to the argument passed to the caller.
  _Any modification of a ByRef item is propagated to the caller at return time._

- Ex: Loading From A Global Container
  
  **ByVal**
  _By value_. The parameter is a _copy_ of the argument passed by the caller.
  _Value modifications are local to the called and not propagated to the caller._

- Ex: Loading From The Local Container (document)
  
  **Optional parameter.**
  _Giving a default value to an optional parameter:_
  ```
  If IsMissing(SomeParam) Then SomeParam = SomeValue
  The identifier is always available in the subprogram.
  ```

**Control Structures**

**Loops**

Repeat a sequence of instructions.
- **Premature exit possible using Exit For or Exit Do according to situation.**

For ... Next

For each counter value ...
- **You must know the counter bounds.**
  - By default, increment step is 1.
  - _Never_ set the counter in the loop instructions!

For Each ... Next

For each item ...
- **The number of items is unknown.**
  - _Item must be of a compatible type._

Do While ... Loop

Do While _Condition_
  - _Loop is evaluated first._
  - _Infinite loops (condition never met)!_
       
  or...

While _Condition_

  - _Older syntax, for compatibility only. Doesn't support Exit._
  - _Do not use!_

Do Loop ... Until

Do _Instructions_ Loop Until _Condition_
  - _Infinite loops (condition never met)!

**Conditional Tests**

A branch that allows to take action according to a given situation.

If (alone)

- **If Condition Then SomeInstruction**
  - _If True_ then...;

If Then Else

If Condition Then
  - _SomeInstructionsThen_
Else
  - _SomeInstructionsElse_
End If

If ElseIf Then ElseEnd If

If Condition Then
  - _SomeInstructionsThen_
ElseIf ConditionThen
  - _SomeInstructionsThen2_
Else
  - _SomeInstructionsElse2_
End If

Select

Select Case SomeVariable
  - _Case Value0_ _‘Instructions for Value0 only’_
  - _Case Value1, Value2_ _‘Instructions for Value1 or Value2’_
  - _Case Value2 To Value4_ _‘Instructions for Value3..Value4’_
  - _Case Else_ _‘Instructions for other situations’_
End Select

**Loading A Code Library**

For readability and maintainability, organize your code in several libraries (RefCard #1).

- The standard code library is the only loaded library at document opening. Others must be explicitly loaded to gain access to their content.
- _Library names are case sensitive!_

**Loading From The Local Container (document)**

_[Checking existence](https://example.com)_

```
```

**Loading From A Global Container**


Mind to identifiers collisions between libraries! You may qualify names using:
- `container.library.modulename.propertyname` (see Examples).
- `Exp:GlobalScope,Tools,Strings,ClearMultiDimArray(MyArray, 3)`

**Calling A Command Associated With A LibreOffice Menu**

101

Use the Dispatcher, and pass it the wanted UNO menu command.

**Knowing UNO Menus Commands**

- UNO menu commands: see the menubar.xml files in the LibreOffice installation directory (OS specific), under share/config/soffice.cfg/modules. Subdir `menubar` of the wanted module (eg: `global/menubar/menubar.xml`, etc).

All commands start with _uno:
- `Ex: ".uno:Open" (File > Open), ".uno:OptionsTreeDialog" (Tools > Options), etc._

**Program Skeleton**

```
Dim Frame As Variant
Dim Dispatch As Object
Dim Args() As Variant
'contents depends from context
Dim UnoCmd As String
Frame = ThisComponent.CurrentController.Frame
UnoCmd = "UNO command to run (above)
Dispatch.executeDispatch(Frame, UnoCmd, "", 0, Args())
```

```
where UnoCmd is the command found in the files above.
```

**Examples**

(only modified parts are shown)

**Ex1. Calling Print Preview**

```
Dim Args() As Variant
 LibExists = BasicLibraries.hasByName("MyLib")
 Frame  = ThisComponent.CurrentController.Frame
 Dim UnoCmd As String
 Dim Args() As Variant  'contents depends from context
 Dim Dispatch As Object
 Dim Frame As Variant
```

```
Dim UnoCmd As String
UNOCmd = "Uno command to run (above)
Dispatch.executeDispatch(Frame, UnoCmd, "", 0, Args())
```

**Ex2. Showing/ HIDing The Sidebar**

```
Dim Args() As Variant
 LibExists = BasicLibraries.hasByName("MyLib")
 Frame  = ThisComponent.CurrentController.Frame
 Dim UnoCmd As String
 Dim Args() As Variant  'contents depends from context
 Dim Dispatch As Object
 Dim Frame As Variant
```

```
Dim UnoCmd As String
UNOCmd = "Uno command to run (above)
Dispatch.executeDispatch(Frame, UnoCmd, "", 0, Args())
```

**Error Management**

In Basic, error management is available using:

- On Error _Xxx_: instructions for error interception;
- Err, Erl and Error: functions to get information about the last error met.

**Error Information Functions**

```
Err
The error code.
```

```
On Error Goto 0
Activates error interception. If an error occurs, the execution continues to MyLabel.
```

```
On Error Goto MyLabel
Activates error interception. If an error occurs, the execution continues to MyLabel.
```

```
On Error Resume Next
In the program body, you must define the label MyLabel! (beware the semicolon character).
```

```
On Error Goto 0
Cancels error interception.
```

```
On Error Resume Next
The line number where the error occurred.
```

```
Error
The message that describes the error.
```

```
Erl
The line number where the error occurred.
```

```
MyError = Err
You may create custom errors by setting a value to Err:
```

```
Err = 1234 generates error 1234.
```

**Different Ways Of Running A Macro**

- **Method**
  - LibreOffice
  - Document Type
  - Current Document

- **Using a toolbar button**
  - 
  - 

- **Using a menu**
  - 
  - 

- **Using a shortcut**
  - 
  - 

- **Through an event**
  - 
  - 

**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 greatness of our sight or the nature of our body, but because we are carried aloft and elevated by the magnitude of the giants (Bernard de Chartres [attr.])

**History**

<table>
<thead>
<tr>
<th>Version</th>
<th>Date</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.0</td>
<td>02/11/2018</td>
<td>Minor updates</td>
</tr>
</tbody>
</table>

**License**

This refcard is placed under the CreativeCommons [BY-SA v4 (intl)](https://creativecommons.org/licenses/by-sa/4.0/) license. More information: [https://creativecommons.org/licenses/by-sa/4.0/](https://creativecommons.org/licenses/by-sa/4.0/)