

# Programming OpenOffice.org with Visual Basic

---

Version 0.8, May-06-2006

French translation is [here](#), but it may be outdated, please check version number and date.

## Public Documentation License Notice

The contents of this Documentation are subject to the Public Documentation License Version 1.0 (the "License"); you may only use this Documentation if you comply with the terms of this License. A copy of the License is available at <http://www.openoffice.org/licenses/PDL.html>.

The Original Documentation is "Programmer OpenOffice.org avec Visual Basic". The Initial Writer of the Original Documentation is Didier ALAIN Copyright (C)2004-2005. All Rights Reserved. (Initial Writer contact(s): [didier.alain@kalitech.fr](mailto:didier.alain@kalitech.fr)).

Contributor(s): Bernard Marcelly, Laurent Godard, Christophe Thibierge.

Portions created by \_\_\_\_\_ are Copyright (C)\_\_\_\_\_ [Insert year(s)]. All Rights Reserved. (Contributor contact(s): \_\_\_\_\_ [Insert hyperlink/alias]).

## Warning

This document is a work in progress : it may still contain some mistakes, misunderstandings or imprecisions

## Special credits



This document is based on work of [KaliTech](http://www.kalitech.fr) SARL (<http://www.kalitech.fr>), french editor of KaliSuite, quality management software.

## Contents

1. [Intented audience](#)
2. [First steps](#)
  1. [For the hurried ones](#)
  2. [Installation](#)
  3. [Short description of OOo API](#)
3. [OOo API and Visual Basic specificities](#)
  1. [OOP \(Object-Oriented Programming\) !](#)
  2. [From MS Office to OOo](#)
  3. [From API to programming language](#)
  4. [From API to Visual Basic](#)
  5. [Tips for "translation"](#)
  6. [Main issues for the Visual Basic programmer](#)
4. [Replacement VB functions for OOo Basic functions](#)
5. [Code samples](#)
6. [Useful Links](#)

## Revision marks

- Suppressed text : ~~the suppressed text~~
- Text being revised : the text being revised
- Added text : **the added text**

## Intended audience

The reader is supposed to have a minimal knowledge about Visual Basic programming concepts and practise.

## First steps

### For the hurried ones

1. Install OpenOffice.org (most recent version advisable)
2. In a Visual Basic or Visual Basic for Application module, copy and paste this procedure :

```
Sub firstOOoProc()  
  
Dim oSM                'Root object for accessing OpenOffice from VB  
Dim oDesk, oDoc As Object 'First objects from the API  
Dim arg()              'Ignore it for the moment !  
  
'Instantiate OOo : this line is mandatory with VB for OOo API  
Set oSM = CreateObject("com.sun.star.ServiceManager")  
'Create the first and most important service  
Set oDesk = oSM.CreateInstance("com.sun.star.frame.Desktop")  
  
'Create a new doc  
Set oDoc = oDesk.loadComponentFromURL("private:factory/swriter", "_blank", 0, arg())  
'Close the doc  
oDoc.Close (True)  
Set oDoc = Nothing  
  
'Open an existing doc (pay attention to the syntax for first argument)  
Set oDoc = oDesk.loadComponentFromURL("file:///c:/dev/ooo/test.doc", "_blank", 0, arg())  
'Save the doc  
Call oDoc.storeToURL("file:///c:/dev/ooo/test2.sxw", arg())  
'Close the doc  
oDoc.Close (True)  
Set oDoc = Nothing  
  
End Sub
```

3. Run the procedure in step by step mode (otherwise you won't see anything !)

## Installation

The setup process for the Visual Basic programmer is very simple :

- Download OOo latest stable release
- Install it
- Open your VB or VBA IDE...
- Program !

There are some details in the official API documentation about OOo and COM technologies :

(<http://api.openoffice.org/docs/DevelopersGuide/ProfUNO/ProfUNO.htm#1+4+4+Automation+Bridge>)

### "Requirements

*The Automation technology can only be used with OpenOffice.org on a Windows platform (Windows 95, 98, NT4, ME, 2000, XP). There are COM implementations on Macintosh OS and UNIX, but there has been no effort to support Automation on these platforms.*

*Using Automation involves creating objects in a COM-like fashion, that is, using functions like `CreateObject()` in VB or `CoCreateInstance()` in C. This requires the `OpenOffice.org` automation objects to be registered with the Windows system registry. This registration is carried out whenever an office is installed on the system. If the registration did not take place, for example because the binaries were just copied to a certain location, then Automation clients will not work correctly or not at all. Refer to 3.4.4 Professional UNO - UNO Language Bindings - Automation Bridge - The Service Manager Component for additional information."*

## Short description of OOo API

*We describe here the main concepts in a very general way. You should read subsequent documentations in detail to get enough understanding of the concepts. I urge you to read one of the many ressources, most of them are of the best quality (see [Useful links](#) section below).*

OOo API is object-oriented, thus it is very different from MS Office components, which are more hierarchical :

- *Some services:*  
In OOo API, one never accesses directly to objects. One uses an intermediate layer : the `services`. `Services` group interfaces, methods and properties which permit the manipulation of objects. Thus, to make things clear, we could say that **object = service** !
- *Some interfaces :*  
Each `service` provides a set of `interfaces`. These `interfaces` export object methods and properties. But with Basic languages (OOo Basic as well as Visual Basic), `interfaces` are not invoked, because they are hidden for easyness. Unfortunately, that easyness becomes a difficulty when trying to understand API documentation, because `interface` is a central concept in it ! The same `interface` is common to many `services` : that's the of an Object-oriented API. As a consequence, when one is able to manipulate an `interface` on a `service`, one is able to manipulate that `interface` on many `services`.
- *Some properties :*  
`Services`, representing objects, have properties. Properties are defined by the (value, name) pair. Some properties are of `structure` data type, which is a problem in Visual Basic (see below).
- *Some methods :*  
`Interfaces` export a set of `methods`, which in turn give access to object properties.
- *Some collections and containers :*  
These concepts are nearly identical to those of the MS Office API. Types of collections are `named collection` (`XNameContainer interface`, `getByName()` method), `indexed collections` (`XIndexContainer interface`, `getByIndex()` method) and `iterative collections` (`XEnumeration interface`, `nextElement()` method).
- *Some modules :*  
`Modules` group `services`, `interfaces`, `types`, `enumerations` and `data structures`.  
Ex. : `text`, `sheet`, `document`, `style`, etc. (the root is always `com.sun.star`)
- *Some events :*  
...

## OOo API and Visual Basic specificities

### OOP (Object-Oriented Programming) !

What is very specific to the object-oriented programming world is the (modules-)services||interfaces-method layer.

`Interfaces` may be identical for many `services` to which they give access (see [this topic](#) in OOoforum, from DannyB). What makes an `interface` is that it always exports the same methods, whatever object it gives access to. But due to inheritance mechanism, it is often very difficult to find which methods is provided for one `service` : a `service` -> multiple `interfaces` -> multiple `methods`.

Whatever you do, you do it accross interfaces, bound with their services. Thus :

1. You access to the `service`
2. You access to `interfaces` (hidden in BASIC)
3. You access to methods, properties

## From MS Office to OOo

Historically, the first versions of StarOffice (up to version 5.2) used to launch the office in a specific desktop environment (graphical environments as KDE or gnome were not mature enough and lightweight X Window managers were widely used). After that, the user used to launch an office component, even in MS Windows (very heavy...). StarOffice integration in OSes GUI desktops has much evolved, but the Desktop object is still very important in OOo API. It offers *by itself* many services, thus *independently* of Office components.

This leads to a distinction between context-dependent services (services included in office components), and context-independent services, callable via the global `StarDesktop` service.

## From API to programming language

It is possible to program OOo API with different languages : C++, java, python, StarBasic, ECMAScript. On the other way, you can also make use of the COM technology from Microsoft : \*.NET (still in development), Visual C++, Visual Basic, VBScript, thanks to *the UNO-Automation bridge*. OOo API is abstract and independent from programming languages. Thus, one has to read and "translate" the API to the programming language : that's the main difficulty !

## From API to Visual Basic

1. An OOo object is created by the VB command `CreateObject`
2. You call other API functions from this root object

In concrete terms :

```
set oRootOOoObject = CreateObject(...)
                        ^VB command

set oOOoObject = oRootOOoObject.CreateInstance(...)
                        ^OOo API method : everything after this point is from the API
```

## Tips for "translation"

Terms used in the API documentation are often strange for the BVB programmer (basic Visual Basic programmer, like me !)...

- **Service = Object:**  
Objects, as explained below, are called `services`. The main reason is that one always accesses and manipulates objects through services, and never in a direct way (services is a abstraction layer between objects and programming languages). Additionnaly, API objects often represents abstract functions, that's the reason why service is a correct term.
- **Component = Document:**  
OpenOffice.org's documents, should they be texts (from Writer), spreadsheets (from Calc), presentations (from Impress), drawings (from Draw), support the `com.sun.star.lang.XComponent` interface. So when you see the `component` term in the API documentation, it refers to a document. Component also refers to an

Office Application (Writer, Calc and so on).

- **UNO and OpenOffice.org API :**

UNO stands for Universal Network Objects. It is a low-level layer used for communication between objects in many type of environments (platforms and programming languages). UNO provides *bridges* and language bindings to do its work. As far as I understand for now, UNO is *almost* completely transparent for the Basic Programmer. OpenOffice.org API (called OOo API in the rest of the document) uses UNO mechanisms underneath. The Visual Basic programmer works with OOo API.

## Main issues for the Visual Basic programmer

Apart from the rudeness and complexity of the API itself, the UNO-Automation bridge have certain restrictions at the moment. These restrictions make the way a little bit more complicated, on certain aspects.

### Passing a structure as a function arguments from VB

(Thanks to Bernard Marcelly)

Some object properties in OOo API have the type `structure`. Structures are the equivalent of User-Defined Type (UDT) in Visual Basic. Because of the implementation of the UNO-Automation bridge, **it is not possible to pass an UDT as a method argument**. (cf <http://api.openoffice.org/docs/DevelopersGuide/ProfUNO/ProfUNO.htm#1+4+4+5+3+Usage+of+Types>). From Visual Basic, one has to use a particular method of OOo API (Bridge\_GetStruct) : it gives access to an OOo structure from Visual Basic. After that, one can give the result of the Bridge\_GetStruct call as argument in a method call.

You could for example use a helper function like this :

```
Function MakePropertyValue(cName, uValue) As Object

    Dim oPropertyValue As Object
    Dim oSM As Object

    Set oSM = CreateObject("com.sun.star.ServiceManager")
    Set oPropertyValue = oSM.Bridge_GetStruct("com.sun.star.beans.PropertyValue")
    oPropertyValue.Name = cName
    oPropertyValue.Value = uValue

    Set MakePropertyValue = oPropertyValue

End Function
```

You could use this function in a portion of code as below. For this example, we use the `loadComponentFromUrl` method. This method accepts as fourth argument a structure that permits opening parameters.

```
Sub openDoc()
    Rem
    Rem Load an existing writer document, with opening parameters
    Rem
    Dim oSM, ODesk as Object 'root object from OOo API
    Dim oDoc as Object      'The document to be opened
    Dim OpenPar(2) As Object 'a Visual Basic array, with 3 elements

    'Instantiate OOo : the first line is always required from Visual Basic for OOo
    Set oSM = CreateObject("com.sun.star.ServiceManager")
    Set oDesk = oSM.CreateInstance("com.sun.star.frame.Desktop")

    'We call the MakePropertyValue function, defined just before, to access the structure
    Set OpenPar(0) = MakePropertyValue("ReadOnly", True)
    Set OpenPar(1) = MakePropertyValue("Password", "secret")
    Set OpenPar(2) = MakePropertyValue("Hidden", False)

    'Now we can call the OOo loadComponentFromURL method, giving it as
    'fourth argument the result of our precedent MakePropertyValue call
    Set oDoc = oDesk.loadComponentFromURL("file:///c:/test.sxw", "_blank", 0, OpenPar)
    ...
End Sub
```

## Name conflicts between Visual Basic integrated methods and OOo API methods

If you write...

```
Call oDoc.Print
```

...The VB `Print` method will take the priority on OOo API `Print` method. It is easy to verify because VB IDE systematically uppercases the method name and suppresses the `call` keyword.

You should write :

```
CallByName oDoc, "print", vbMethod, arg 'arg is an array of arguments (cf MakePropertyValue)
```

## OOo Basic functions

OOo Basic has some (usefull) integrated functions or instructions, that Visual Basic doesn't have (and the opposite is true, of course ! see the VB 6 `replace` - also usefull - function !). Sometimes, it is difficult to know if you call an OOo Basic function or an OOo API function, but you can always verify it with the OOo Basic Online Help, in the 'other commands' section. Don't get fooled by this. Here is a non-exhaustive list :

- ConvertToUrl ([see replacement functions](#))
- HasUnoInterfaces (although see [here](#))
- CreateUnoStruct (an ingrated function for MakePropertyValue, described before, see [replacement functions](#))
- CreateUnoService ([see replacement functions](#))
- CreateUnoDialog
- CreateUnoListener
- CreateUnoValue
- ThisComponent
- Dbg\_Methods
- Dbg\_Properties

to be written, as suggested by DannyB, VB replacement functions

## Event handling

to be written...

## Pre-defined constants

OOo API predefined constants are not accessible from Visual Basic (although, as [DannyB said](#), I would be very happy if someone could prove I am wrong). One solution is to map the constant value with a VB constant declaration, for example :

```
Const com_sun_star_text_ControlCharacter_PARAGRAPH_BREAK = 0
```

to be written : an exhaustive constant declaration list

## Replacement VB functions for OOo Basic functions

*Warning ! These functions are not fully tested and are in beta state. Any comment would be very appreciated ([didier.alain@kalitech.fr](mailto:didier.alain@kalitech.fr))*

## ConvertToUrl

*see below for a more complete function*

```
'
'Converts a Ms Windows local pathname in URL (RFC 1738)
'Todo : UNC pathnames, more character conversions
'
Public Function ConvertToUrl(strFile) As String
    strFile = Replace(strFile, "\", "/")
    strFile = Replace(strFile, ":", "|")
    strFile = Replace(strFile, " ", "%20")
    strFile = "file:/// " + strFile
    ConvertToUrl = strFile
End Function
```

## MakePropertyValue

Original from DannyB.

```
'
'Creates a sequence of com.sun.star.beans.PropertyValue s
'
Public Function MakePropertyValue(cName, uValue) As Object
Dim oStruct, oServiceManager As Object
    Set oServiceManager = CreateObject("com.sun.star.ServiceManager")
    Set oStruct = oServiceManager.Bridge_GetStruct("com.sun.star.beans.PropertyValue")
    oStruct.Name = cName
    oStruct.Value = uValue
    Set MakePropertyValue = oStruct
End Function
```

## CreateUnoService

```
'
'A simple shortcut to create a service
'
Public Function CreateUnoService(strServiceName) As Object
Dim oServiceManager As Object
    Set oServiceManager = CreateObject("com.sun.star.ServiceManager")
    Set CreateUnoService = oServiceManager.CreateInstance(strServiceName)
End Function
```

## Code samples

### FileName2URL (Martin Wildam, [mwildam\\_at\\_may.co.at](mailto:mwildam_at_may.co.at))

```
Public Function FileName2URL(ByVal pFileName As String, _
                             Optional ByVal pConvertBackslashesToSlashes As Boolean = True) _
    As String

    Dim s As String
    Dim z As String
```

```

Dim j As Long
Dim x As Integer

On Error Resume Next
s = ""
For j = 1 To Len(pFileName)
    z = Mid(pFileName, j, 1)
    x = Asc(z)
    Select Case x
        Case 9
            z = "%09"
        Case 13
            z = "%0d"
        Case 10
            z = "%0a"
        'Case 32
        ' z = "+"
        Case 32 To 35, 37 To 41, 43, 44, 59 To 63, 91, 93, 94, 96, Is >= 123
            z = "%" & Hex(x)
        Case 92
            If pConvertBackslashesToSlashes Then z = "/"
    End Select
    s = s & z
Next
s = "file:////" & s
FileName2URL = s
End Function

```

## Simple bookmark demo

```

Sub bk_demo()
'
' Bookmark demo
'
' You need an existing writer doc (c:\tmp\testdoc.odt),
' execute in step by step mode to follow
' result in the doc (use Writer navigator to see bookmarks)
'
' Note : this sample code works with writer documents (sxw or odt formats)
' as well as with MsOffice Word documents
'

Dim oSM, oDesk, oDoc, oTxt, oBk, oRng As Object 'OOo objects
Dim strBkName As String
Dim arg()

Set oSM = CreateObject("com.sun.star.ServiceManager")
Set oDesk = oSM.CreateInstance("com.sun.star.frame.Desktop")
Set oDoc = oDesk.loadComponentFromURL("file:///C:/tmp/testdoc.odt", "_blank", 0, arg())
Set oTxt = oDoc.Text

'Inserts a bookmark at the beginning of the text
Set oBk = oDoc.CreateInstance("com.sun.star.text.Bookmark")
oBk.Name = "New_Bookmark_Beginning"
Set oRng = oTxt.getStart()
Call oTxt.insertTextContent(oRng, oBk, True)

'Inserts a bookmark at the end of the text
Set oBk = oDoc.CreateInstance("com.sun.star.text.Bookmark")
oBk.Name = "New_Bookmark_End"
Set oRng = oTxt.getEnd()
Call oTxt.insertTextContent(oRng, oBk, True)

'Inserts some text at an existing bookmark, requested by name
strBkName = "New_Bookmark_End"
oDoc.getBookmarks().getByName("New_Bookmark_End").getAnchor.setString ("**Inserted at '" & strBk

'Deletes an existing bookmark, requested by name
strBkName = "New_Bookmark_End"
Set oBk = oDoc.getBookmarks().getByName(strBkName)
Set oTxt = oDoc.Text
Call oTxt.removeTextContent(oBk)

```



```

'Cleaning all objects
Call oDoc.Close(False) 'Close without saving
Set oBk = Nothing
Set oRng = Nothing
Set oTxt = Nothing
Set oDoc = Nothing
Set oDesk = Nothing
Set oSM = Nothing

End Sub

```

### Simple Save as PDF demo

```

Sub SaveAsPDF_demo()
'
' Save an existing writer document as PDF
'
Dim oSM, oDesk, oDoc As Object 'OOo objects
Dim OpenParam(1) As Object 'Parameters to open the doc
Dim SaveParam(1) As Object 'Parameters to save the doc

Set oSM = CreateObject("com.sun.star.ServiceManager")
Set oDesk = oSM.CreateInstance("com.sun.star.frame.Desktop")

Set OpenParam(0) = MakePropertyValue("Hidden", True) ' Open the file hidden
Set oDoc = oDesk.loadComponentFromURL("file:///C:/tmp/testdoc.odt", "_blank", 0, OpenParam())

Set SaveParam(0) = MakePropertyValue("FilterName", "writer_pdf_Export")
Call oDoc.storeToURL("file:///C:/tmp/testdoc.pdf", SaveParam())

Set oDesk = Nothing
Set oSM = Nothing

End Sub

```

### Search And Replace demo (Jon Winchester, [jon\\_at\\_logicworks.cc](http://jon_at_logicworks.cc))

```

Private Sub Search_Replace()
Dim oSM 'Root object for accessing OpenOffice from VB
Dim oDesk, oDoc As Object 'First objects from the API
Dim arg() 'Ignore it for the moment !
Dim mmerge As Object
'Instantiate OOo : this line is mandatory with VB for OOo API
Set oSM = CreateObject("com.sun.star.ServiceManager")

'Create the first and most important service
Set oDesk = oSM.CreateInstance("com.sun.star.frame.Desktop")
'Create a new doc
Set oDoc = oDesk.loadComponentFromURL("file:///c:/temp/oo2.sxw", "_blank", 0, arg())
' jon code
Dim objText As Object, objCursor As Object
Set objText = oDoc.GetText
Set objCursor = objText.createTextCursor

' replace all
Dim oSrch As Object
'Set oSrch = oDoc.createSearchDescriptor
Set oSrch = oDoc.createReplaceDescriptor
oSrch.setSearchString ("[name]")
oSrch.setReplaceString ("Smith, Robert James")
Debug.Print oDoc.replaceAll(oSrch)

oSrch.setSearchString ("[city]")
oSrch.setReplaceString ("West Palm Beach")

```

```

Debug.Print oDoc.replaceAll(oSrch)

oSrch.setSearchString ("[state]")
oSrch.setReplaceString ("Florida")
Debug.Print oDoc.replaceAll(oSrch)

oSrch.setSearchString ("[zip]")
oSrch.setReplaceString ("33411")
Debug.Print oDoc.replaceAll(oSrch)

Dim arg1(0) As Object
Set arg1(0) = MakePropertyValue("", 0)
CallByName oDoc, "print", VbMethod, arg1 'arg is an array of arguments (cf MakePropertyValue)

'Close the doc
oDoc.Close (True)
Set oDoc = Nothing
End Sub

```

## Useful links

- [Porting Excel/VBA to Calc/StarBasic](#) : Although the perfect opposite of mine (migrating from VB to OOo Basic), some parts are really very usefull, since they compare the two languages. See also the part named "Understanding the OpenOffice Object Model" : it's far more precise and explanatory than my "Short description of OOo API".
- [StarOffice 7 Software Basic Programmer's guide](#) also in other languages, see [OOo API external ressources](#)
- [API reference](#)
- [Developer's Guide](#)
  - Especially [chapter 3.4.4 "Automation Bridge"](#)
- Many threads in OOoForum, most of them from DannyB :
  - [Using COM for OOo with different languages](#)
  - [Open and Create documents, various prog. languages](#)
  - [MakePropertyValue function](#) (the original one, from which I took the `MakePropertyValue()` function)
  - [A Service may support multiple Interfaces...](#)