LibreOffice

Guide Calc
6.2

Chapitre 12
Macro Calc

Automatiser les tâches répétitives

Droits d’auteur

Contributeurs

Ce chapitre est une adaptation mise à jour de *LibreOffice.org 4.1 Calc Guide*.

Ont contribué à cette édition

Steve Fanning
Jean Hollis Weber

Ont contribué aux éditions précédentes

Andrew Pitonyak
Barbara Duprey
Jean Hollis Weber
Simon Brydon

Traduction

De cette édition

Traducteurs Jean-Luc Vandemeulebroucke
Relecteurs Philippe Clément

Des éditions précédentes

Traducteurs Christian Chenal
Relecteurs Philippe Clément

Retours

Veuillez adresser tout commentaire ou suggestion concernant ce document à la liste de diffusion de l’Équipe de Documentation : doc@fr.libreoffice.org

Remarque

tout ce que vous envoyez à la liste de diffusion, y compris votre adresse mail et toute autre information personnelle incluse dans le message, est archivé publiquement et ne peut pas être effacé.

Date de publication et version du logiciel

Publié en août 2020. Basé sur LibreOffice 6.2.

Utiliser LibreOffice sur un Mac

Sur Mac, certaines touches et certains éléments de menu sont différents de ceux utilisés sous Windows ou Linux. Le tableau ci-dessous donne quelques substitutions courantes pour les instructions de ce chapitre. Pour une liste plus détaillée, voyez l'Aide de l'application.

Windows ou Linux	***Équivalent Mac***	***Effet***
Sélection du menu **Outils > Options**	**LibreOffice > Préférences**	Accès aux options de configuration
Clic droit	*Control+clic* ou *clic droit* selon la configuration de l'ordinateur	Ouvre un menu contextuel
Ctrl (Control)	⌘ *(Command)*	Utilisé avec d'autres touches
F5	*Shift+⌘+F5*	Ouvre le navigateur
F11	*⌘+T*	Ouvre l'onglet *Styles et Formatage* du volet latéral

Table des matières

Introduction

Le chapitre 13, *Débuter avec les macros*, du Guide du débutant est une introduction aux fonctionnalités des macros disponibles dans LibreOffice. Le présent chapitre offre des informations supplémentaires sur l'utilisation des macros dans un classeur Calc.

Une macro est une séquence enregistrée de commandes ou de saisies au clavier qui sont conservées pour une utilisation ultérieure. Un exemple de macro simple serait une macro qui « saisit » votre adresse. Le langage de macro LibreOffice est très souple et permet l'automatisation de tâches simples ou complexes. Les macros sont particulièrement utiles pour répéter une suite d'actions exactement de la même façon, plusieurs fois au cours du temps et elles vous permettent d'ajouter à Calc des fonctionnalités qui n'y sont pas incorporées.

La façon la plus simple pour créer une macro consiste à enregistrer une série d'actions grâce à l'interface utilisateur de Calc. Celui-ci enregistre les macros en utilisant le langage de script LibreOffice Basic, qui est une variante du langage de programmation bien connu BASIC. Ces macros peuvent être éditées et enrichies après enregistrement grâce à l'Environnement de Développement Intégré (EDI) de LibreOffice Basic.

Les macros de Calc les plus puissantes sont créées en écrivant du code dans un des quatre langages de script possibles (LibreOffice Basic, BeanShell, JavaScript et Python). Ce chapitre offre un survol des fonctionnalités des macros dans Calc, essentiellement axé sur son langage de script par défaut, LibreOffice Basic. Certains exemples sont aussi proposés pour les langages de script BeanShell, JavaScript et Python mais une description complète des fonctionnalités de ces langages sort du cadre de ce document.

Utiliser l'enregistreur de macro

Le Chapitre 13, *Débuter avec les macros*, du Guide du Débutant apporte les connaissances de base pour comprendre les possibilités générales des macros de LibreOffice en utilisant l'enregistreur de macro. L'exemple, spécifique à Calc, qui va suivre comportera des explications moins détaillées. Les étapes suivantes créent une macro qui effectue un collage spécial avec une multiplication dans une plage de cellules d'un classeur.

Attention

Pour pouvoir utiliser l'enregistreur de macro, vous devez tout d'abord activer l'option *Activer l'enregistrement des macros* qui se trouve dans **Outils > Options > LibreOffice > Avancé**.

1) Ouvrez un nouveau classeur.

2) Saisissez des nombres dans une feuille comme sur la Figure 1.

	A	B	C
1	1	8	9
2	2	7	10
3	3	6	11

Figure 1 : Saisissez des nombres dans les cellules A1:C3.

3) Sélectionnez la cellule A3, qui contient le nombre 3, et copiez cette valeur dans le presse-papiers.

4) Sélectionnez la plage A1:C3.

5) Utilisez **Outils > Macros > Enregistrer une macro** pour démarrer l'enregistreur de macro. La boîte de dialogue *Enregistrer une macro* s'affiche avec un bouton **Terminer l'enregistrement** (Figure 2).

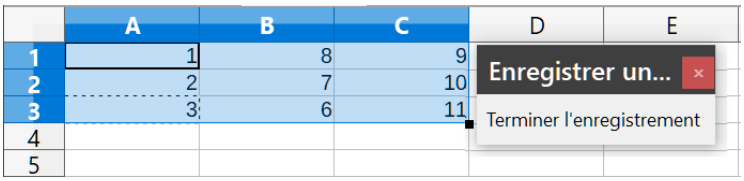

Figure 2 : La boîte de dialogue Enregistrer une macro avec le bouton Terminer l'enregistrement.

6) Utilisez **Édition > Collage spécial > Collage spécial** pour ouvrir la boîte de dialogue *Collage spécial* (Figure 3).

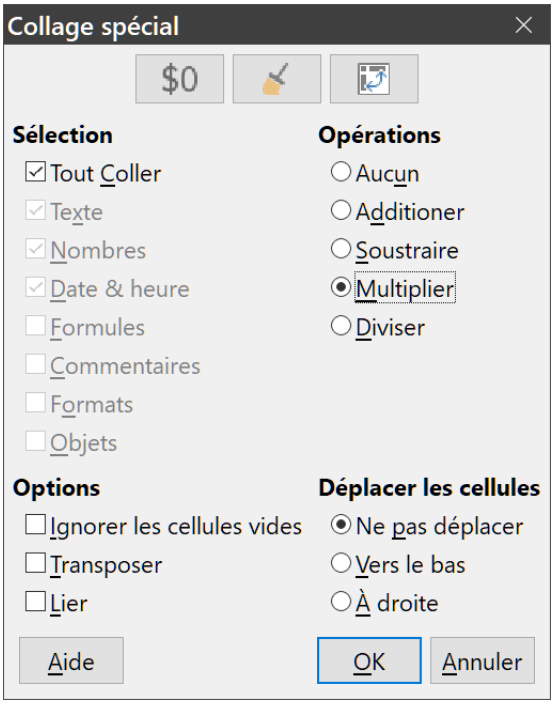

Figure 3 : La boîte de dialogue Collage spécial

7) Choisissez *Tout coller* dans la section **Sélection** et *Multiplier* dans la section **Opérations**, et cliquez sur **OK**. Les cellules sont alors multipliées par 3 (Figure 4).

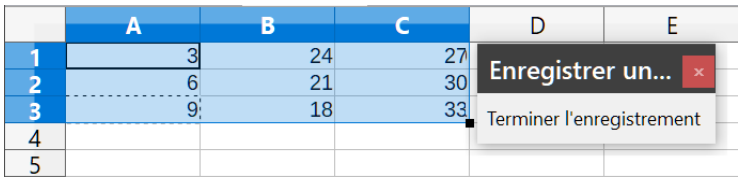

Figure 4 : Les cellules de la plage A1:C3 multipliées par 3.

8) Cliquez sur **Terminer l'enregistrement** pour arrêter l'enregistreur de macro. La boîte de dialogue *Macros LibreOffice Basic* s'ouvre (Figure 5).

Remarque

La section **Enregistrer la macro** dans de la boîte de dialogue *Macros LibreOffice Basic* affiche les macros existantes, hiérarchiquement structurées en conteneurs, bibliothèques, modules et macros comme il est décrit au chapitre 13 du Guide du débutant. La Figure 5 affiche les conteneurs *Mes macros*, *Macros LibreOffice* ainsi que ceux pour les fichiers ouverts *Budget football.ods* et *Journal des ventes.ods*, et celui du fichier sans nom créé à l'étape 1. Servez-vous de l'icône d'expansion ou de réduction à gauche de chaque nom de conteneur pour voir les bibliothèques, modules et macros qu'il contient.

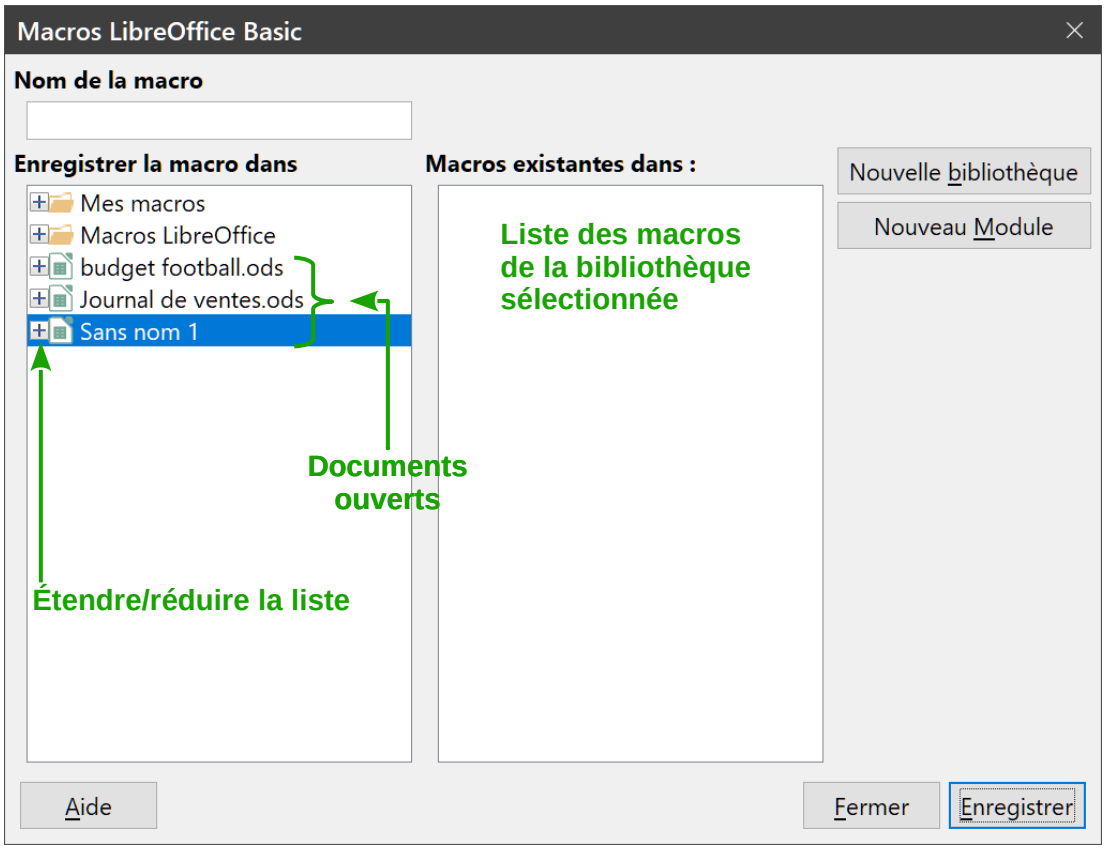

Figure 5 : La boîte de dialogue Macro LibreOffice Basic.

9) Sélectionnez le document en cours (voir Figure 5). Pour cet exemple, le document Calc en cours s'intitule *Sans nom 1*. Les documents comportent une bibliothèque appelée *Standard*. Cette bibliothèque n'est pas affichée tant que ce n'est pas nécessaire pour Calc, et donc à cet instant votre nouveau document ne montre pas de bibliothèque. Vous pouvez en créer une nouvelle, mais ce n'est pas indispensable.

10) Cliquez sur **Nouveau module**. Dans la boîte de dialogue *Nouveau module* (Figure 6), saisissez un nom pour le nouveau module ou laissez le nom par défaut.

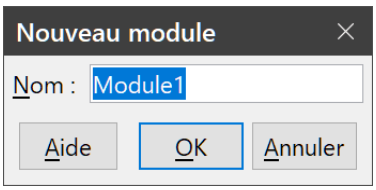

Figure 6 : La boîte de dialogue Nouveau module.

Remarque

Les noms de bibliothèques, modules, macros doivent respecter des règles strictes. Pour se limiter aux principales, les noms doivent :

- commencer par une lettre ;
- ne pas contenir d'espace ;
- ne pas contenir de caractères spéciaux, accents compris, hormis le _ (tiret bas) ;
- être constitués de lettres minuscules (a..z), majuscules (A..Z), chiffres (0..9) et de tiret bas (_).

11) Cliquez sur **OK** pour créer un nouveau module appelé *Module1*. Comme il n'existe aucune bibliothèque de macros dans notre document, Calc en crée automatiquement une nommée *Standard*.

12) Sélectionnez le *Module1* nouvellement créé, saisissez *CopieMultiplication* dans la zone **Nom de la macro** en haut à gauche, et cliquez sur **Enregistrer** (Figure 7).

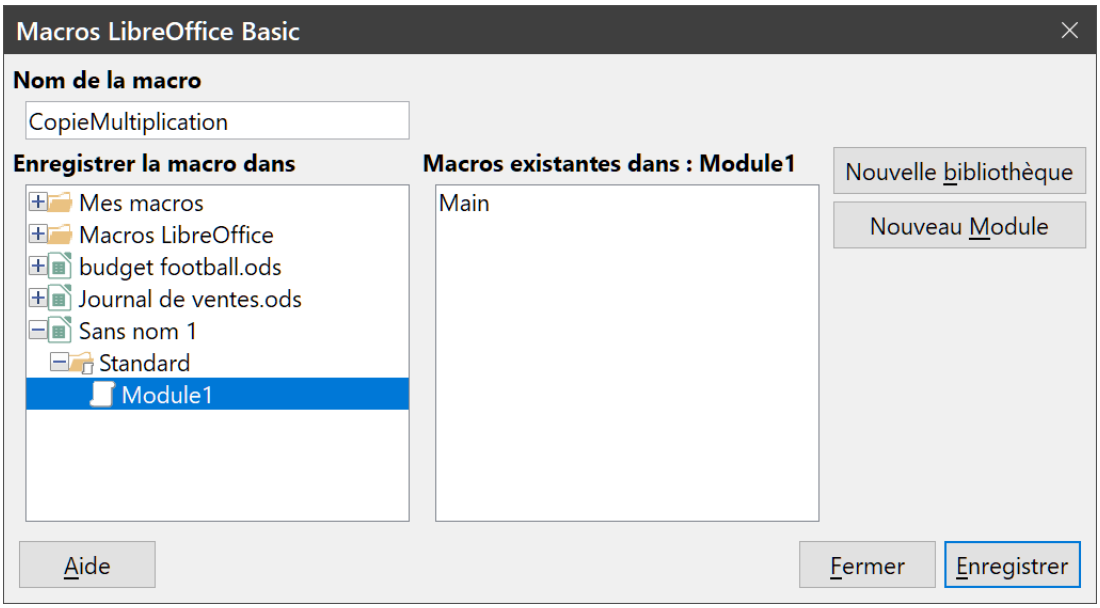

Figure 7 : Sélectionner le module et nommer la macro.

La macro créée est enregistrée dans *Module1* de la bibliothèque *Standard* du document *Sans nom 1*. Le Listing 1 vous montre le contenu de la macro.

Listing 1. Collage spécial avec multiplication

```
sub CopieMultiplication
rem ----------------------------------------------------------------------
rem define variables
dim document   as object
dim dispatcher as object
rem
----------------------------------------------------------------------------
rem get access to the document
document   = ThisComponent.CurrentController.Frame
```

```
dispatcher = createUnoService("com.sun.star.frame.DispatchHelper")

rem
----------------------------------------------------------------------
dim args1(5) as new com.sun.star.beans.PropertyValue
args1(0).Name = "Flags"
args1(0).Value = "A"
args1(1).Name = "FormulaCommand"
args1(1).Value = 3
args1(2).Name = "SkipEmptyCells"
args1(2).Value = false
args1(3).Name = "Transpose"
args1(3).Value = false
args1(4).Name = "AsLink"
args1(4).Value = false
args1(5).Name = "MoveMode"
args1(5).Value = 4

dispatcher.executeDispatch(document, ".uno:InsertContents", "", 0,
args1())


End Sub
```

Vous trouverez plus de détails sur l'enregistrement des macros dans le chapitre 13, *Débuter avec les Macros*, du Guide du débutant. Nous vous recommandons de le lire si vous ne l'avez déjà fait. Vous trouverez également plus de détails dans les paragraphes suivants, mais nous ne reviendrons plus sur l'enregistrement des macros.

Écrire vos propres fonctions

Créer une fonction macro

Calc peut appeler des macros en tant que fonctions Calc. Suivez les étapes suivantes pour créer une macro simple :

1) Créez un nouveau document Calc appelé par exemple *CalcTestMacros.ods* et laissez-le ouvert dans Calc.

2) Utilisez **Outils > Macros > Gérer les macros > LibreOffice Basic** pour ouvrir la boîte de dialogue *Macros LibreOffice Basic* (Figure 8). Notez que la disposition de cette boîte de dialogue n'est, dans ce cas, pas la même que celle que Calc affiche quand l'utilisateur clique sur le bouton **Terminer l'enregistrement** de la boîte de dialogue *Enregistrer une macro* (voir Figure 5).

   La zone **Macro de** liste les conteneurs de bibliothèques de macros disponibles, dont les documents LibreOffice ouverts. *Mes macros* contient les macros que vous avez écrites ou ajoutées à LibreOffice. *Macros LibreOffice* contient les macros fournies avec LibreOffice ou installées au travers des extensions, qui ne peuvent être que consultées.

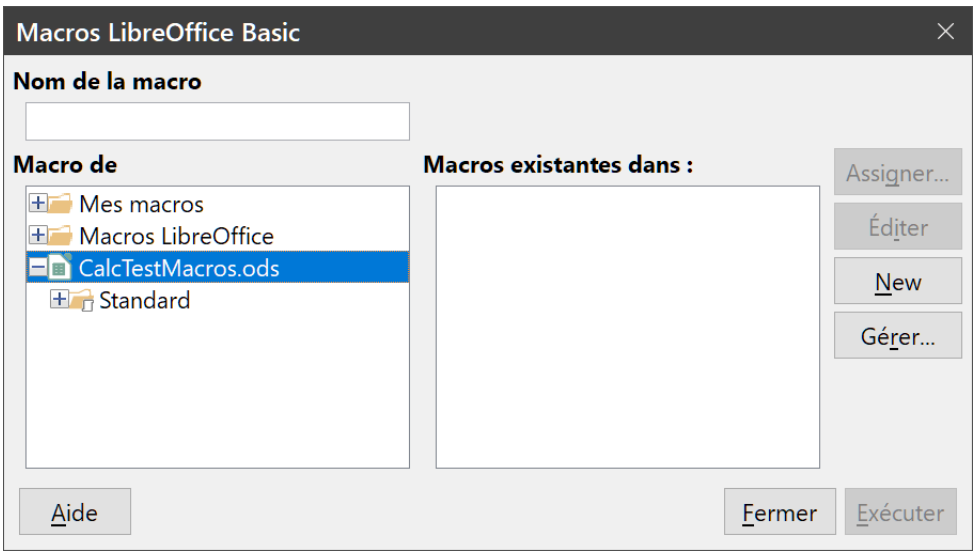

Figure 8 : La boîte de dialogue Macros LibreOffice Basic.

3) Cliquez sur **Gérer** pour ouvrir la boîte de dialogue *Gestionnaire de macros de LibreOffice Basic* (Figure 9). Dans l'onglet Bibliothèques, sélectionnez le document qui va contenir la macro.

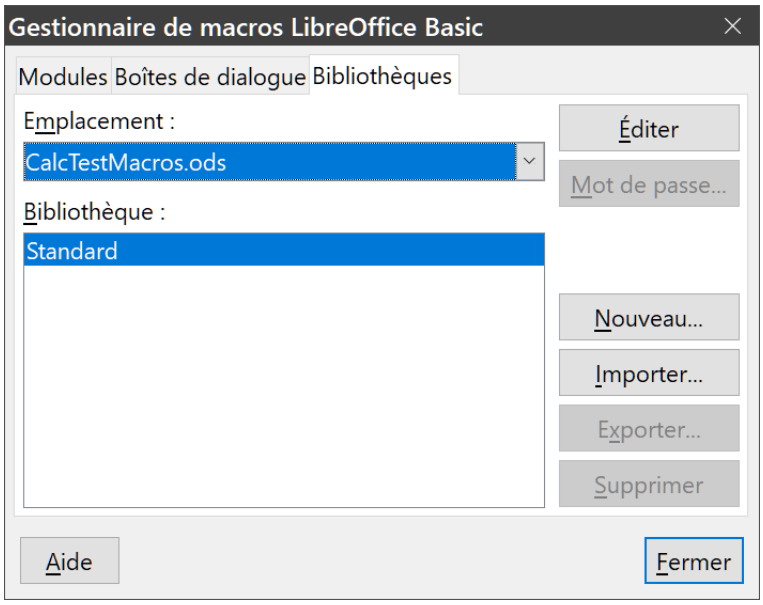

Figure 9 : La boîte de dialogue Gestionnaire de macros LibreOffice Basic.

Cliquez sur l'onglet *Bibliothèques* et, dans la liste déroulante *Emplacement*, sélectionnez le document courant. La section *Bibliothèque* se met à jour et affiche le nom de la bibliothèque *Standard* vide.

4) Cliquez sur **Nouveau** pour ouvrir la boîte de dialogue *Nouvelle bibliothèque*.

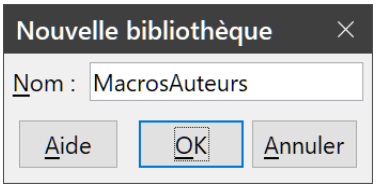

Figure 10 : La boîte de dialogue Nouvelle bibliothèque.

5) Saisissez un nom de bibliothèque significatif (comme `MacrosAuteurs`) et cliquez sur **OK** pour créer la bibliothèque. Le nouveau nom de bibliothèque est affiché dans la liste des bibliothèques, mais la boîte de dialogue peut n'afficher qu'une partie du nom.

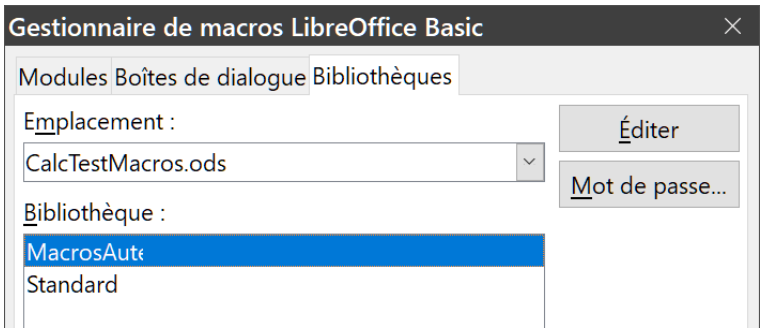

Figure 11 : La bibliothèque est affichée.

6) Sélectionnez *MacrosAuteurs* et cliquez sur **Éditer** pour éditer la bibliothèque. Calc crée automatiquement un module appelé *Module1* et une macro appelée *Main*. Calc ouvre l'Environnement de Développement Intégré (EDI) de LibreOffice Basic visible sur la Figure 12.

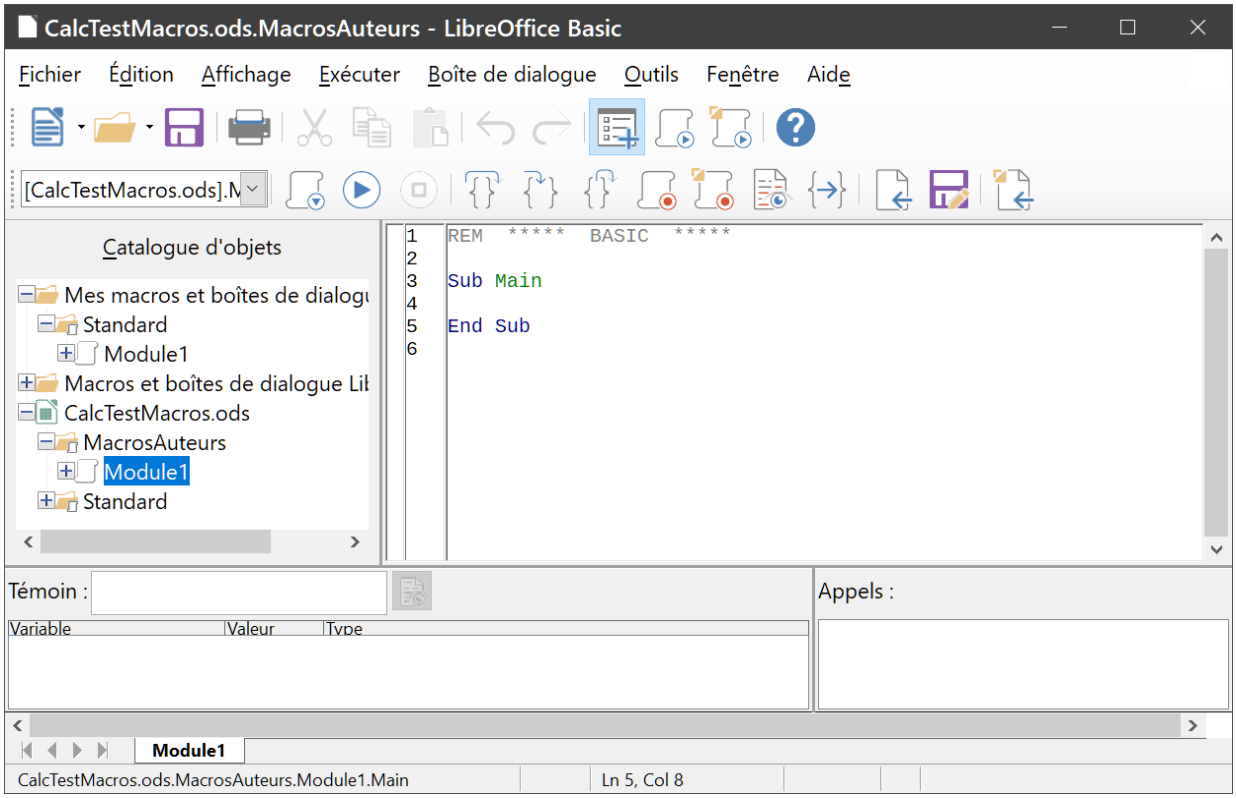

Figure 12 : L'environnement de développement intégré Basic.

La Figure 12 montre la configuration par défaut de l'EDI de LibreOffice Basic qui comporte :

– une barre de menu ;

– trois barres d'outils (*Langue*, *Macro* et *Standard*). La barre d'outils *Macro* propose divers boutons qui permettent d'éditer et de tester les programmes ;

– le Catalogue d'objets où il est possible de sélectionner le conteneur, la bibliothèque, le module et la macro désirés ;

- la fenêtre d'édition où vous pouvez saisir et modifier le code du programme en LibreOffice Basic ;
- la fenêtre de suivi (située à gauche, sous le catalogue d'objets et la fenêtre d'édition) qui affiche le contenu des variables et des tableaux pendant un processus pas à pas ;
- la fenêtre de la pile d'appels (située à droite, sous le catalogue d'objets et la fenêtre d'édition) qui offre des informations sur la pile d'appels des procédures et fonctions pendant l'exécution d'un programme ;
- un ensemble d'onglets ;
- une barre d'état.

L'EDI de LibreOffice Basic offre de puissantes fonctionnalités pour le développement et le débogage des macros de LibreOffice. Leur description complète sort du cadre de ce document.

7) Modifiez le code pour qu'il soit celui du Listing 2. Vous écrivez la fonction *NombreCinq*, qui retourne le nombre 5.

Conseil

L'instruction `Option Explicit` rend obligatoire la déclaration de toutes les variables avant leur utilisation. Si cette instruction est omise et les variables non déclarées, les variables seront automatiquement définies avec le type `Variant` pour leur première utilisation.

Attention

L'instruction `Option Explicit` doit être écrite au début du module, avant toute instruction. Elle ne peut être précédée que de commentaires (ligne commençant une apostrophe ou par REM), comme dans l'exemple du Listing 2.

8) Enregistrez le *Module1* ainsi modifié.

Listing 2. Fonction qui retourne le nombre 5

```
REM  *****  BASIC  *****
Option Explicit

Sub Main

End Sub

Function NombreCinq()
  NombreCinq = 5
End Function
```

Utiliser une macro en tant que fonction

Dans le document CalcTestMacros.ods, saisissez la formule `=NombreCinq()` (Figure 13). Calc retrouve la macro et l'appelle.

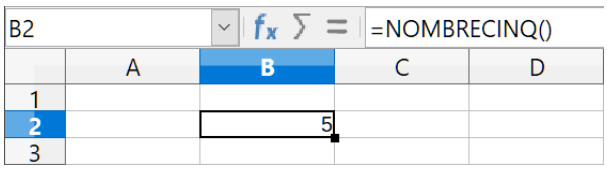

Figure 13 : Utilisation de la macro NombreCinq() en tant que fonction.

Remarque

Les noms de fonctions ne sont pas sensibles à la casse. Dans la Figure 13, vous pouvez saisir `=NombreCinq()` et Calc affichera `=NOMBRECINQ()`.

Avertissement de sécurité concernant les macros

Enregistrez le document Calc, fermez-le et ouvrez-le à nouveau. En fonctions de vos paramètres dans **Outils > Options > LibreOffice > Sécurité** section **Sécurité des macros**, Calc va afficher le message d'avertissement de la Figure 14 ou celui de la Figure 15.

Si l'avertissement de la Figure 14 s'affiche, vous devrez cliquer sur **Activer les macros**, ou Calc ne permettra pas qu'une macro soit exécutée à l'intérieur du document. Si vous ne vous attendiez pas à ce que le document contienne une macro, il est plus sûr de cliquer sur **Désactiver les macros**, pour le cas où la macro contiendrait du code malveillant.

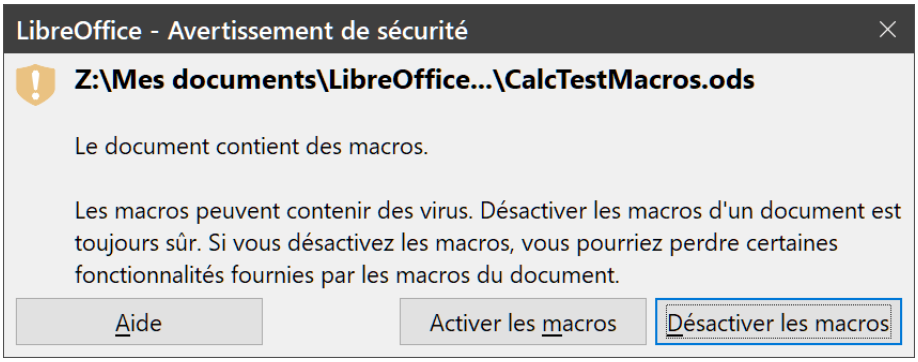

Figure 14 : Avertissement de sécurité moyenne si le document contient des macros.

Si l'avertissement de la Figure 15 s'affiche, Calc n'exécutera aucune macro dans le document et vous ne pouvez que cliquer sur **OK** pour refermer la boîte de dialogue.

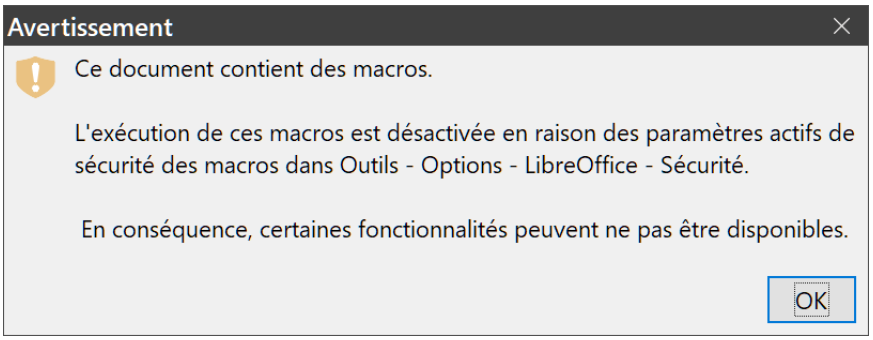

Figure 15 : Avertissement de sécurité élevée si le document contient des macros.

Si vous choisissez de désactiver les macros, alors, quand le document se charge, Calc ne peut plus trouver la fonction et indique une erreur dans les cellules concernées en affichant `#NOM ?` dans celles-ci.

Bibliothèques chargées / déchargées

Quand un document est créé et enregistré, il contient automatiquement une bibliothèque appelée *Standard*. Quand un classeur est ouvert, Calc n'ouvre pas toutes les bibliothèques de macros qu'il peut trouver dans les conteneurs disponibles, car ce serait gâcher des ressources. À la place, Calc ne charge automatiquement, quand le document est ouvert, que cette bibliothèque *Standard* au sein du conteneur *Mes macros* ainsi la bibliothèque *Standard* propre au document. Aucune autre bibliothèque n'est chargée automatiquement.

Calc ne comporte pas de fonction appelée *NombreCinq()*, et il va donc rechercher cette fonction dans toutes les bibliothèques de macros ouvertes et visibles. Calc va explorer les bibliothèques dans *Macros LibreOffice*, *Mes macros* et la bibliothèque *Standard* du document Calc (voir Figure 5). La fonction *NombreCinq()* est stockée dans la bibliothèque *MacrosAuteurs*, qui n'est pas chargée automatiquement à l'ouverture du document, ce qui explique que la fonction ne soit pas trouvée et qu'une erreur d'exécution apparaisse dans la cellule où elle est appelée (Figure 16)

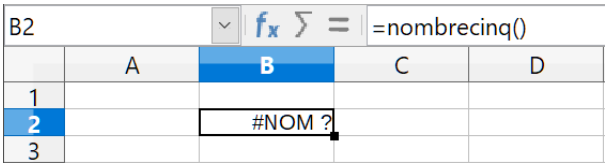

Figure 16 : La fonction macro n'est pas accessible.

Utilisez **Outils > Macros > Gérer les macros > LibreOffice Basic** pour ouvrir la boîte de dialogue *Macros LibreOffice Basic*, développez *CalcTestMacros* et trouvez *MacrosAuteurs*. (Figure 17). Les icônes d'une bibliothèque chargée (*Standard*, par exemple) n'ont pas la même apparence que celles d'une bibliothèque qui n'est pas chargée (*MacrosAuteurs*, par exemple).

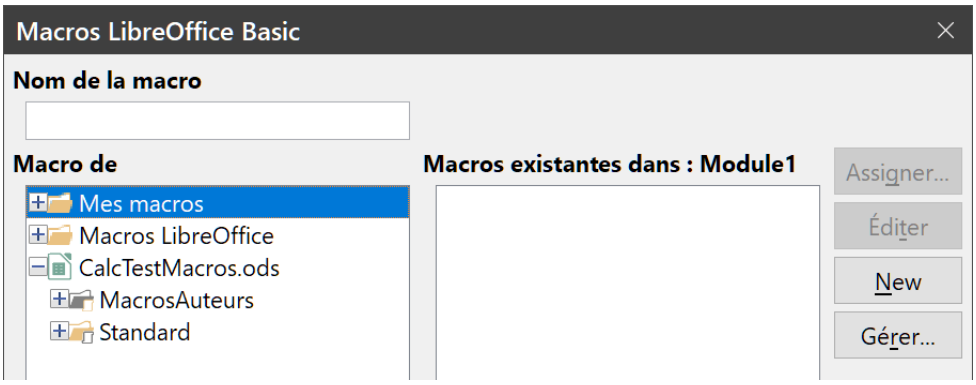

Figure 17 : les icônes des bibliothèques chargées et non chargées sont différentes.

Cliquez sur le symbole d'extension (habituellement le signe + ou un triangle ▶) devant *MacrosAuteurs* pour charger la bibliothèque. La couleur de l'icône change pour indiquer que la bibliothèque est désormais chargée. Cliquez sur **Fermer** pour fermer la boîte de dialogue.

Malheureusement, les cellules qui contiennent `=NombreCinq()` sont toujours en erreur. Calc ne recalcule pas les cellules en erreur, à moins que vous ne les modifiiez ou que vous les changiez d'une façon ou d'une autre. La meilleure solution est de stocker les macros utilisées en tant que fonctions dans la bibliothèque *Standard*. Si la macro a une taille importante ou s'il y a beaucoup de macros, un relais avec le nom voulu peut être stocké dans la bibliothèque *Standard*. La macro relais charge la bibliothèque contenant l'implémentation, puis appelle cette implémentation. Cette méthode est illustrée ci-dessous.

1) Utilisez **Outils > Macros > Gérer les macros > LibreOffice Basic** pour ouvrir la boîte de dialogue *Macros LibreOffice Basic*. Sélectionnez la macro *NombreCinq* et cliquez sur **Éditer** pour ouvrir la macro en édition.

2) Modifiez le nom `NombreCinq` en `NombreCinq_Implementation` (Listing 3).

Listing 3. Nom changé de NombreCinq en NombreCinq_Implementation

```
Function NombreCinq_Implementation() as integer
  NombreCinq = 5
End Function
```

3) Dans l'EDI Basic, cliquez sur le bouton **Sélectionner une macro** pour ouvrir la boîte de dialogue *Macros LibreOffice Basic*.

4) Sélectionnez la bibliothèque *Standard* du document *CalcTestMacros* et cliquez sur **Nouveau** (**New**) pour créer un nouveau module. Saisissez un nom significatif comme `FonctionsCalc` et cliquez sur **OK**. LibreOffice crée automatiquement une macro appelée *Main* et ouvre le module pour édition.

5) Créez une macro dans la bibliothèque *Standard* qui appelle l'implémentation de la fonction (voir Listing 4). La nouvelle macro charge la bibliothèque *MacrosAuteurs* si ce n'est pas déjà fait, et ensuite appelle l'implémentation de la fonction.

Listing 4. Macro NombreCinq de la bibliothèque Standard

```
Function NombreCinq()
  If Not BasicLibraries.isLibraryLoaded("MacrosAuteurs") Then
    BasicLibraries.LoadLibrary("MacrosAuteurs")
  End If
  NombreCinq = NombreCinq_Implementation()
End Function
```

6) Enregistrez, fermez et ouvrez à nouveau le document Calc. Cette fois, la fonction *NombreCinq()* fonctionne.

Passer des arguments à une macro

Pour illustrer une fonction qui accepte des arguments, voici une macro qui calcule la somme des nombres positifs de son argument, en ignorant ceux qui sont inférieurs à zéro (Listing 5).

Listing 5. SommePositif calcule la somme des nombres positifs de son argument

```
Function SommePositif(Optional x)
  Dim LaSomme As Double
  Dim iLig As Integer
  Dim iCol As Integer

  LaSomme = 0.0
  If NOT IsMissing(x) Then
    If NOT IsArray(x) Then
      If x > 0 Then LaSomme = x
    Else
```

```
      For iRow = LBound(x, 1) To UBound(x, 1)
        For iCol = LBound(x, 2) To UBound(x, 2)
          If x(iRow, iCol) > 0 Then LaSomme = LaSomme + x(iRow, iCol)
        Next
      Next
    End If
  End If
  SommePositif = LaSomme
End Function
```

La macro du Listing 5 permet d'illustrer des techniques importantes :

1) L'argument x est facultatif (`optionnal`). Si un argument n'est pas facultatif et qu'une fonction est appelée sans lui, LibreOffice affiche un message d'avertissement à chaque appel de la macro. Si la fonction est appelée plusieurs fois, l'erreur sera alors affichée plusieurs fois.

2) `IsMissing` vérifie qu'un argument a été passé avant de l'utiliser.

3) `IsArray` vérifie si l'argument est une valeur simple ou un tableau de valeurs. Par exemple, `=SommePositif(7)` ou `=SommePositif(A4)`. Dans le premier cas, le nombre 7 est passé en tant qu'argument, et dans le second cas, la valeur de la cellule A4 est passée à la fonction. Dans les deux cas, `IsArray` renvoie `False`.

4) Si une plage est passée à la fonction, elle est passée en tant que tableau de valeurs à deux dimensions. Par exemple, `=SommePositif(A2:B5)`. `LBound` et `UBound` servent à déterminer les limites du tableau utilisé. Bien que la limite inférieure du tableau soit 1, il est plus prudent d'utiliser `LBound` si jamais la fonction change dans le futur. C'est par ailleurs une bonne habitude à prendre, car le fait que la limite soit 1 est ici une exception. Par défaut la limite inférieure des matrices est zéro.

Conseil

La macro du Listing 5 est prudente et vérifie si l'argument est une valeur simple ou un tableau. Par contre, elle ne vérifie pas si chaque valeur est numérique. Vous pouvez être aussi prudent que vous le voulez : plus vous vérifierez de choses, plus la macro sera robuste, mais plus son exécution sera lente.

Passer un ou deux arguments est tout aussi facile : ajoutez un autre argument à la définition de la fonction (Listing 6). Lorsque vous appelez dans une cellule de classeur une fonction avec deux arguments, séparez-les avec un point-virgule, par exemple, `=TestMax(3;-4)`.

Listing 6. TestMax retourne le plus grand des deux arguments

```
Function TestMax(x, y)
  If x >= y Then
    TestMax = x
  Else
    TestMax = y
  End If
End Function
```

Arguments passés en tant que valeurs

Les arguments passés à une macro de Calc sont toujours des valeurs, numériques ou textuelles (jamais des objets). L'argument ne permet pas de savoir quelles cellules sont utilisées, si c'est le cas. Par exemple, `=SommePositif(A3)` passe la valeur de la cellule A3, et la macro n'a pas le moyen de savoir que la cellule A3 est utilisée. Si vous devez savoir quelles cellules sont référencées plutôt que la valeur de ces cellules, passez la plage en tant que chaîne de caractères, analysez cette chaîne et obtenez les valeurs des cellules référencées.

Astuce

Passer l'adresse de la cellule ou de la plage en tant que chaîne de caractères ("A3" par exemple) rend impossible l'adaptation relative des formules lors de la recopie dans le tableur. La solution est d'utiliser les fonctions FEUILLE et CELLULE permettant respectivement d'obtenir le numéro de feuille et l'adresse de la cellule. Exemple : `=ESTRESULTPOSITIF(A1;FEUILLE();CELLULE("adresse";A1))`

Écrire des macros qui se comportent comme les fonctions internes

Bien que Calc retrouve et appelle les macros comme les fonctions normales, elles ne se comportent pas réellement comme les fonctions internes. Par exemple, les macros n'apparaissent pas dans les listes de fonctions. Il est possible d'écrire des fonctions qui se comportent comme les fonctions ordinaires en écrivant une extension. Il s'agit cependant là d'un sujet élaboré réservé aux programmeurs expérimentés et qui sort du cadre de ce guide.

Accéder directement aux cellules

Vous pouvez accéder directement aux objets internes de LibreOffice pour manipuler un document Calc. Par exemple, la macro du Listing 7 additionne les valeurs de la cellule A2 de toutes les feuilles dans le document en cours. `ThisComponent` est initialisé par le langage LibreOffice Basic quand la macro démarre et référence le document en cours. Un document Calc contient des feuilles : `ThisComponent.getSheets()` retourne l'ensemble de ces feuilles. Utilisez `getCellByPosition(col, row)` pour retourner une cellule se trouvant à une ligne et une colonne déterminées.

Listing 7. Somme de la cellule A2 de toutes les feuilles

```
Function SommeFeuilles()
  Dim LaSomme As Double
  Dim i As integer
  Dim oFeuilles
  Dim oFeuille
  Dim oCellule

  oFeuilles = ThisComponent.getSheets()
  For i = 0 To oFeuilles.getCount() - 1
    oFeuille = oFeuilles.getByIndex(i)
    oCellule = oFeuille.getCellByPosition(0, 1) ' Cellule A2
    LaSomme = LaSomme + oCellule.getValue()
  Next
```

```
  SommeFeuilles = LaSomme
End Function
```

Astuce

Un objet de type cellule offre les méthodes `getValue()`, `getString()` et `getFormula()` pour obtenir la valeur numérique, la valeur chaîne de caractères et la formule utilisées dans une cellule. Utilisez les méthodes « set » correspondantes pour fixer les valeurs voulues.

Utilisez `oFeuille.getCellRangeByName("A2")` pour obtenir une plage de cellules à partir de son nom. Si une seule cellule est référencée, un objet cellule est alors renvoyé. Si une plage de cellules est fournie, alors toute la plage de cellules est renvoyée (voir Listing 8). Notez qu'avec une plage de cellules, la fonction renvoie les données sous forme d'un tableau de tableaux, ce qui est plus lourd que de traiter un tableau à deux dimensions, comme dans le Listing 5.

Listing 8. Somme de la plage A2:C5 de toutes les feuilles

```
Function SommeFeuilles()
  Dim LaSomme As Double
  Dim iLig As Integer, iCol As Integer, i As Integer
  Dim oFeuilles, oFeuille, oCellules
  Dim oLig(), oLigs()

  oFeuilles = ThisComponent.getSheets()
  For i = 0 To oFeuilles.getCount() - 1
    oFeuille = oFeuilles.getByIndex(i)
    oCellules = oFeuille.getCellRangeByName("A2:C5")
    REM getDataArray() retourne les données comme variant,
    REM donc nombres et chaînes de caractères sont retournés.
    REM getData() retourne les données de type Double,
    REM seuls des nombres sont retournés.
    oLigs() = oCellules.getData()
    For iLig = LBound(oLigs()) To UBound(oLigs())
      oLig() = oLigs(iLig)
      For iCol = LBound(oLig()) To UBound(oLig())
        LaSomme = LaSomme + oLig(iCol)
      Next
    Next
  Next
  SommeFeuilles = LaSomme
End Function
```

Attention

Quand une macro est appelée en tant que fonction Calc, elle ne peut modifier que la valeur de la cellule appelante, c'est-à-dire celle qui contient la fonction. Aucune autre cellule ne pourra être modifiée.

Tri

Vous voulez trier les données de la Figure 18. Le tri se fera tout d'abord selon la colonne B en ordre décroissant, puis selon la colonne A en ordre croissant.

	A	B	C
1	1	5	Un
2	4	1	Deux
3	3	1	Trois
4	7	8	Quatre
5	4	2	Cinq

	A	B	C
1	7	8	Quatre
2	1	5	Un
3	4	2	Cinq
4	3	1	Trois
5	4	1	Deux

Figure 18 : Tri de la colonne B en ordre décroissant et de la colonne A en ordre croissant.

L'exemple du Listing 9 montre comment effectuer un tri sur deux colonnes.

Le tri s'applique à une plage de données. Avec un objet *Plage*, nous disposons de la méthode *Sort*. Cette méthode requiert en argument tous les paramètres de tri, sous la forme d'un « tableau de propriétés ».

L'appel à cette méthode est quant à lui très simple : `oPlage.Sort(oDescTri())`.

La difficulté est ici de créer le tableau des paramètres du tri (variable `oDescTri` dans l'exemple ci-dessus). Nous aurons des paramètres généraux (exemple : la zone de tri comprend-elle une ligne de titre), et des paramètres pour chaque clé de tri (colonne à utiliser, tri croissant ou décroissant).

Ceci correspond très exactement à ce qui est renseigné via la commande **Données > Trier** respectivement dans les onglets *Options* et *Critères de tri*.

Listing 9. Tri de la plage A1:C5 dans la Feuille1

```
Sub TriPlage
  Dim oFeuille  ' Feuille qui contient les données à trier.
  Dim oPlage    ' Plage de données à trier.

  REM Un tableau de champs de tri détermine les colonnes à
  REM trier. Il contient autant d'éléments que de colonnes à trier,
  REM numérotés à partir de 0 : ici 0 et 1.
  REM Pour ne trier qu'une colonne, faites :
  REM Dim oChampsTri(0) As New com.sun.star.table.TableSortField
  Dim oChampsTri(1) As New com.sun.star.table.TableSortField

  REM Le descripteur de tri est un tableau de propriétés.
  REM La propriété primaire contient les champs de tri.
  Dim oDescTri(1) As New com.sun.star.beans.PropertyValue

  REM Retourne la feuille appelée "Feuille1"
  oFeuille = ThisComponent.Sheets.getByName("Feuille1")
  REM Retourne la plage à trier
  oPlage = oFeuille.getCellRangeByName("A1:C5")

  REM Sélectionne la plage à trier. Aurait comme
  REM effet de mettre en surbrillance les données triées.
```

```
  'ThisComponent.getCurrentController.select(oPlage)

  REM Les colonnes sont numérotées à partir de 0,
  REM la colonne A est 0, B est 1, etc.
  REM Tri de la colonne B (colonne 1) décroissant.
  oChampsTri(0).Field = 1
  oChampsTri(0).SortAscending = False

  REM Si la colonne B a deux cellules de même valeur,
  REM trier selon la colonne A croissant.
  oChampsTri(1).Field = 0
  oChampsTri(1).SortAscending = True

  REM Paramétrer le descripteur de tri.
  oDescTri(0).Name = "SortFields"
  oDescTri(0).Value = oChampsTri()

  REM Trier la plage.
  oPlage.Sort(oDescTri())
End Sub
```

Aperçu des macros BeanShell, JavaScript et Python

Introduction

Comme de nombreux programmeurs peuvent ne pas être familiers de LibreOffice Basic, Calc permet d'écrire des macros dans trois autres langages qui sont peut-être plus connus. Ce sont BeanShell, JavaScript et Python.

Le langage de script principal des macros de Calc est LibreOffice Basic et l'installation standard offre un environnement de développement intégré (EDI) puissant ainsi que plus d'options pour ce langage.

Les macros sont organisées de la même manière pour les quatre langages de script. Le conteneur *LibreOffice Macros* contient toutes les macros fournies à l'installation de LibreOffice. La bibliothèque *Mes Macros* contient vos macros qui sont disponibles dans n'importe lequel de vos documents LibreOffice. Chaque document peut aussi contenir vos macros qui ne sont disponibles dans aucun autre document.

Quand vous utilisez la fonction **Enregistreur de macro**, Calc crée la macro en LibreOffice Basic. Pour utiliser les autres langages de script disponibles, vous devez écrire le code vous-même.

Quand vous sélectionnez **Outils > Macros > Exécuter la macro** dans la barre de menu, Calc ouvre la boîte de dialogue *Sélecteur de macro* qui vous permet de choisir et d'exécuter n'importe quelle macro disponible, quel que soit le langage qui a servi à la coder (Figure 19).

Quand vous sélectionnez **Outils > Macros > Éditer la macro** dans la barre de menu, Calc ouvre l'EDI de LibreOffice Basic qui vous permet de choisir et de modifier toute macro LibreOffice Basic disponible, mais pas celles écrites dans d'autres langages.

Le modèle de composants utilisé dans LibreOffice est connu sous le nom Universal Network Objects (UNO). Les macros de LibreOffice, écrites avec n'importe quel langage de script, utilisent à l'exécution une interface de programmation d'application (Application Programming Interface : API) de UNO. L'interface XSCRIPTCONTEXT est transmise aux scripts des macros dans les quatre langages et offre un moyen d'accéder aux diverses interfaces dont vous pouvez avoir besoin pour effectuer une action sur un document.

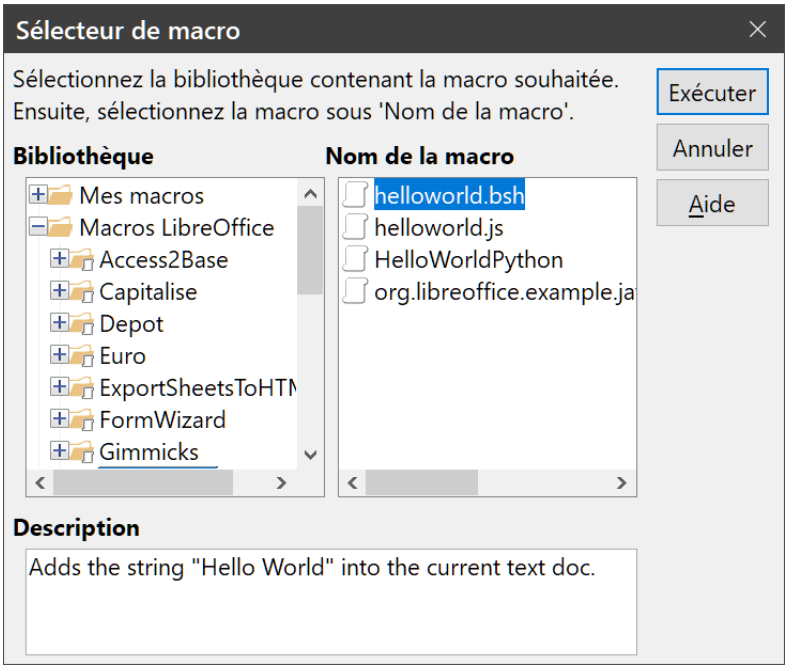

Figure 19 : La boîte de dialogue Sélecteur de macro.

Macros BeanShell

BeanShell est un langage de script qui ressemble à Java sorti en 1999.

Quand vous sélectionnez **Outils > Macros > Gérer les macros > BeanShell** dans la barre de menu, Calc ouvre la boîte de dialogue *Macros BeanShell* (Figure 20).

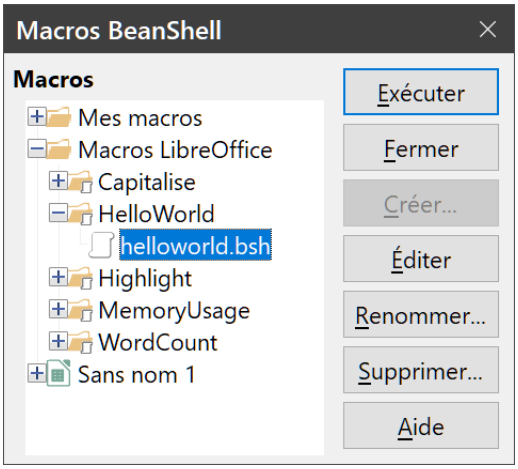

Figure 20 : La boîte de dialogue Macros BeanShell.

Cliquez sur le bouton Éditer de la boîte de dialogue Macros BeanShell pour ouvrir la boîte de dialogue BeanShell Debug Window (Figure 21).

Le Listing 10 est un exemple de macro BeanShell qui insère le texte « Hello World (in BeanShell) » dans la cellule A1 du classeur Calc actif.

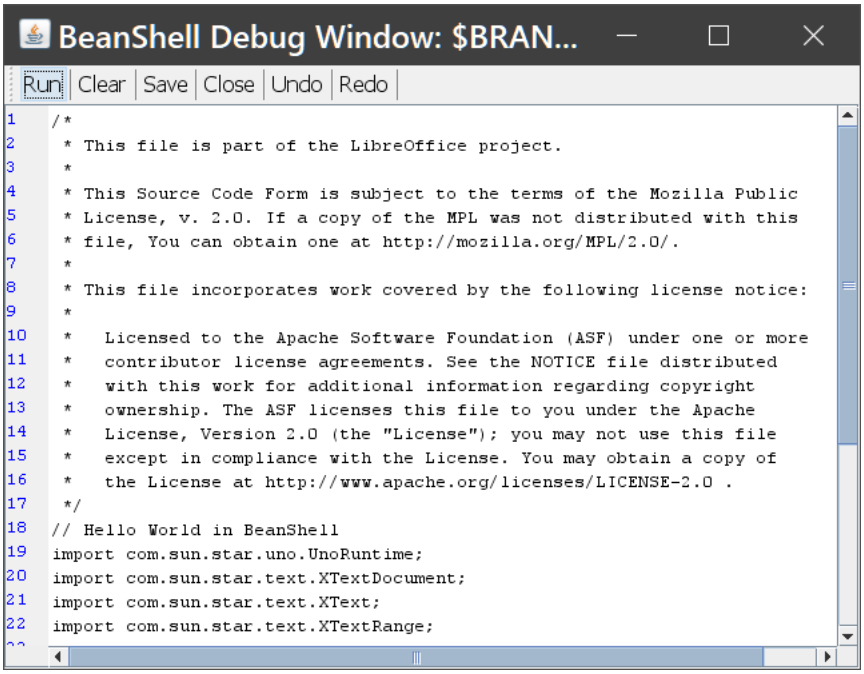

Figure 21 : la boîte de dialogue BeanShell Debug Window.

Listing 10 Exemple de macro BeanShell

```
// Hello World in BeanShell
import com.sun.star.uno.UnoRuntime;
import com.sun.star.text.XTextDocument;
import com.sun.star.text.XText;
import com.sun.star.text.XTextRange;

// get the document from the scripting context which is made available
to all
// scripts
oDoc = XSCRIPTCONTEXT.getDocument();
//get the XTextDocument interface
xTextDoc = (XTextDocument)
UnoRuntime.queryInterface(XTextDocument.class,oDoc);
//get the XText interface
xText = xTextDoc.getText();
// get an (empty) XTextRange at the end of the document
xTextRange = xText.getEnd();
// set the string
xTextRange.setString( "Hello World (in BeanShell)" );
```

Macros JavaScript

JavaScript est un langage de script de haut niveau réalisé en 1995.

Quand vous sélectionnez **Outils > Macros > Gérer les macros > JavaScript** dans la barre de menu, Calc ouvre la boîte de dialogue *Macros JavaScript* (Figure 22).

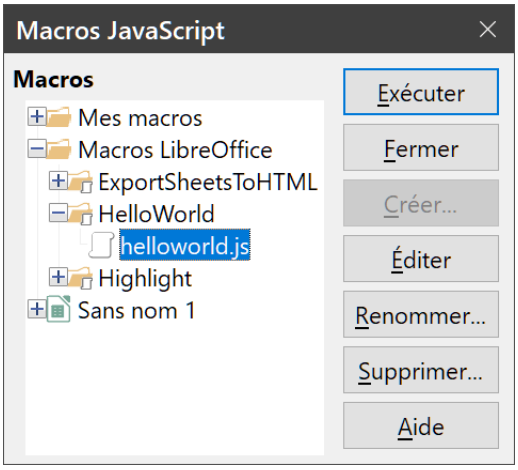

Figure 22 : La boîte de dialogue Macros JavaScript.

Cliquez sur le bouton **Éditer** de la boîte de dialogue *Macros JavaScript* pour ouvrir le débogueur JavaScript Rhino (Rhino JavaScript Debugger) (Figure 23). Vous pouvez trouver sur le site web de Mozilla, à l'adresse https://developer.mozilla.org/en-US/docs/Mozilla/Projects/Rhino/Debugger, des instructions détaillées sur l'utilisation de cet outil.

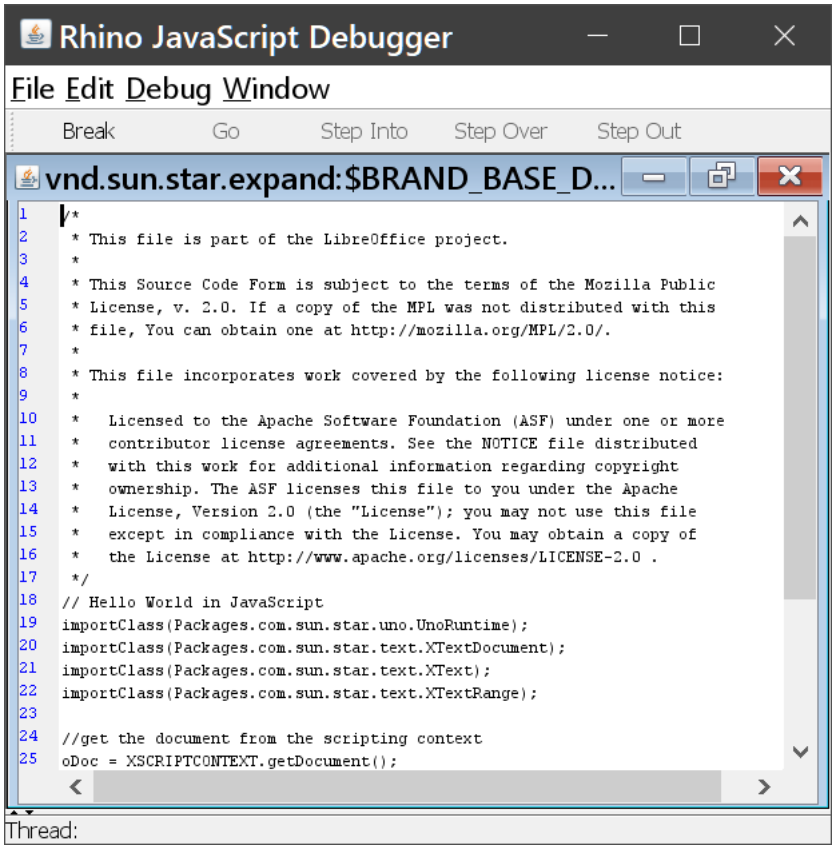

Figure 23 : Le débogueur JavaScript Rhino.

Le Listing 11 propose un exemple de macro en JavaScript qui insère le texte « Hello World (in JavaScript) » dans la cellule A1 de la première feuille d'un classeur Calc.

Listing 11 : exemple de macro en JavaScript.

```
// Hello World in JavaScript
importClass(Packages.com.sun.star.uno.UnoRuntime);
importClass(Packages.com.sun.star.text.XTextDocument);
importClass(Packages.com.sun.star.text.XText);
importClass(Packages.com.sun.star.text.XTextRange);

//get the document from the scripting context
oDoc = XSCRIPTCONTEXT.getDocument();
//get the XTextDocument interface
xTextDoc = UnoRuntime.queryInterface(XTextDocument,oDoc);
//get the XText interface
xText = xTextDoc.getText();
//get an (empty) XTextRange interface at the end of the text
xTextRange = xText.getEnd();
//set the text in the XTextRange
xTextRange.setString( "Hello World (in JavaScript)" );
```

Macros Python

Python est un langage de programmation de haut niveau à usage général qui est sorti en 1991.

Quand vous sélectionnez **Outils > Macros > Gérer les macros > Python** dans la barre de menu, Calc ouvre la boîte de dialogue *Macros Python* (Figure 24).

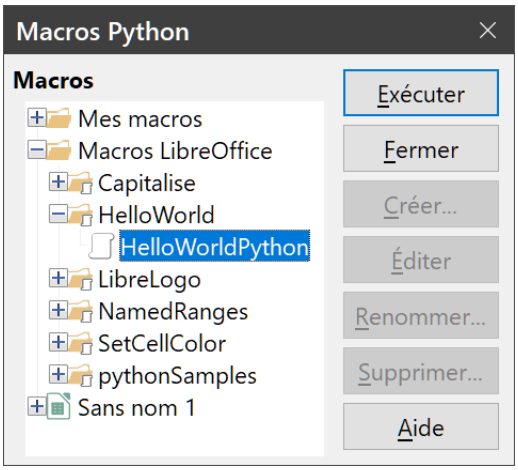

Figure 24 : La boîte de dialogue Macros Python.

Aucune fonction d'édition et de débogage des scripts Python n'est actuellement intégrée à l'interface utilisateur standard de LibreOffice. Vous pouvez cependant les modifier avec votre éditeur de texte préféré ou un EDI externe. L'extension Alternative Python Script Organizer (APS0) (https://extensions.libreoffice.org/extensions/apso-alternative-script-organizer-for-python) facilite l'édition des scripts Python, en particulier ceux qui sont incorporés à un document. Elle vous permet de configurer votre éditeur de code source préféré, de lancer l'environnement Python intégré et de déboguer les scripts Python. Si vous souhaitez plus d'informations, recherchez le terme « Python » dans le système d'aide de LibreOffice et visitez la section *Designing & Developing Python Applications* du wiki de *The Document Foundation* à l'adresse https://wiki.documentfoundation.org/Macros/Python_Design_Guide/fr.

Le Listing 12 est un exemple de macro en Python qui place le texte « Hello World from Python » dans la cellule A1 de la première feuille d'un classeur Calc.

Listing 12 : exemple de macro en Python

```
import uno

def HelloWorld():
    doc = XSCRIPTCONTEXT.getDocument()
    cell = doc.Sheets[0]['A1']
    cell.setString('Hello World from Python')
    return
```

Conclusion

Ce chapitre constitue un bref aperçu de la façon de créer des bibliothèques et des modules, en utilisant l'enregistreur de macro, en utilisant les macros en tant que fonctions Calc, ou en écrivant vos propres macros sans l'enregistreur. Chaque sujet mériterait au moins un chapitre, et l'écriture de vos propres macros pour Calc pourrait faire l'objet d'un ou plusieurs livres. Vous pouvez également vous reporter à la page du wiki correspondante pour des exemples supplémentaires : https://wiki.documentfoundation.org/Macros/fr. En d'autres termes, ceci est juste le début de ce que vous pouvez découvrir !