La personnalisation du ruban sous Excel 2007

Par SilkyRoad

Date de publication : 21 avril 2008

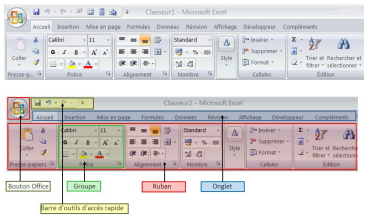
Dernière mise à jour : 7 août 2008

Ce tutoriel est une présentation des options qui vous permettront de modifier la présentation du ruban, sous Excel 2007.

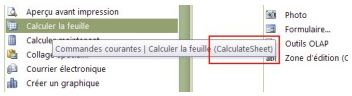
Commentez

I - Introduction

Une des principales nouveautés sous Excel 2007, est le remplacement des menus et des barres d'outils par un ruban constitué d'onglets qui regroupent les fonctionnalités de l'application.



Cette nouvelle interface utilisateur semblera peut être déroutante pour les possesseurs d'anciennes versions d'Excel. L'adaptation se fait toutefois rapidement. Passez un peu de temps à repérer **dans quels onglets sont rangés vos anciennes barres d'outils** et familiarisez vous avec la logique du tableur, le ruban proposant des groupes de contrôles, arrangés dans des onglets et regroupés par thème. Si vous avez besoin d'aide, profitez des info-bulles qui détaillent les fonctionnalités lorsque le curseur de la souris passe sur les menus.
Il est toujours possible d'utiliser les barres d'outils et les menus personnalisés de vos anciens classeurs, et en créer de nouveaux par VBA, mais ils seront accessibles uniquement depuis l'onglet **Complément**.
Pour modifier le ruban, vous devez enregistrer vos classeurs au format .xlsm (format xml prenant en charge les macros) et y ajouter un fichier xml (généralement nommé customUI.xml par convention) qui contient les instructions de personnalisation. Que faut-il indiquer dans le fichier xml ? C'est l'objet de ce tutoriel.
Vous pourrez ainsi ajouter vos propres menus/fonctions dans les classeurs, mais également regrouper dans un seul onglet toutes les fonctions d'Excel que vous utilisez le plus souvent, masquer le ruban ou des onglets prédéfinis, désactiver des transactions d'Excel, renommer les commandes de l'application à votre goût.
Nota 1:
Utilisez le format .xlam pour créer un complément (macro complémentaire) et modifier le ruban dès l'ouverture d'Excel.
Cet article ne décrit pas en détail l'Open XML et les nouveaux formats de fichier disponibles dans Microsoft Office 2007. Consultez le tutoriel d' **Olivier Lebeau** si vous souhaitez obtenir plus d'information sur ces nouvelles extensions: **Le XML dans Microsoft Office (OpenXML)**.
Nota 2:
Bien qu'il soit possible de modifier manuellement le fichier xml de personnalisation, les exemples de ce tutoriel sont majoritairement basés sur l'utilitaire "customUI Editor" présenté dans le chapitre II-B.
Pour réutiliser les contrôles prédéfinis du Ruban, vous devez connaître leur identificateur (idMso/imageMso).
Cliquez sur le bouton "Office".
Cliquez sur le bouton "Options Excel".
Utilisez le menu "Personnaliser".
Le volet permet d'accéder à tous les contrôles (visibles et masqués).
Les identificateurs apparaissent dans une info-bulle lorsque vous passez le curseur sur les icônes.



Microsoft met également à votre disposition la liste des identificateurs de contrôles prédéfinis :
2007 Office System Document: Lists of Control IDs

II - Les outils pour modifier le ruban

II-A - Manuellement

Avant d'aller plus loin, il est important de connaitre les outils qui permettent de modifier le ruban.
Il n'existe malheureusement pas pour l'instant de concepteur pour gérer de manière intuitive le Ruban.
L'interface utilisateur reposant sur le langage **xml**, vous pouvez créer les fichiers de personnalisation depuis un simple éditeur de texte (par exemple le Bloc-notes).
Le mode opératoire consiste à créer un fichier xml qui va contenir les paramètres de personnalisation, puis à insérer ce fichier dans le classeur Excel préalablement sauvegardé au format xlsm ou xlam.
Pour démarrer, vous devez créer le fichier xml de personnalisation:

Ouvrez le Bloc-notes et copiez les balises suivantes dans le fichier.

Xml

```
<customUI xmlns="http://schemas.microsoft.com/office/2006/01/customui">
    <ribbon startFromScratch="false">

        <tabs>
            <tab id="OngletPerso" label="OngletPerso" visible="true">
            </tab>
       </tabs>

    </ribbon>
</customUI>
```

Sauvegardez le fichier au format xml, à l'emplacement de votre choix sur votre PC. Par convention, ce fichier doit être nommé customUI.xml.
Ensuite, fermez le fichier.
Ouvrez un nouveau classeur Excel et enregistrez-le sous l'extension .xlsm (format openXML avec macros).
Refermez le classeur.
Recherchez le classeur sur votre PC et ajoutez lui l'extension .zip.
Ensuite dézippez le fichier.
Ouvrez le dossier dézippé.
Ajoutez un répertoire nommé customUI dans le répertoire dézippé.
Faites glisser le fichier customUI.xml dans ce dossier.
Vous allez maintenant créer une relation entre le classeur et le fichier xml:
Démarrez le Bloc-notes.
Utilisez le menu Fichier/Ouvrir.
Ouvrez le fichier .rels du fichier Zip, depuis le bloc-notes :
Dans le champ "Fichier de type", sélectionnez l'option "Tous les fichiers".
Recherchez et ouvrez le fichier .rels dans le dossier dézippé (dans le sous répertoire _rels).
Ajoutez-y la balise ci-dessous.

Xml

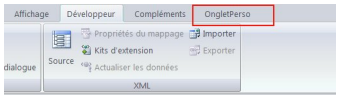
```
<Relationship Id="rId10" Type="http://schemas.microsoft.com/office/2006/relationships/ui/
extensibility"
Target="/customUI/customUI.xml"/>
```

Vous allez obtenir un fichier de ce style :

Xml

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<Relationships xmlns="http://schemas.openxmlformats.org/package/2006/relationships">

    <Relationship Id="rId10" Type="http://schemas.microsoft.com/office/2006/relationships/ui/
extensibility"
        Target="/customUI/customUI.xml"/>

    <Relationship Id="rId3" Type="http://schemas.openxmlformats.org/officeDocument/2006/
relationships/extended-properties"
        Target="docProps/app.xml"/>

    <Relationship Id="rId2" Type="http://schemas.openxmlformats.org/package/2006/relationships/
metadata/core-properties"
        Target="docProps/core.xml"/>

    <Relationship Id="rId1" Type="http://schemas.openxmlformats.org/officeDocument/2006/
relationships/officeDocument"
        Target="xl/workbook.xml"/>

</Relationships>
```

Sauvegardez et refermez le fichier .rels.
Rezippez le dossier complet.
Supprimez l'extension .zip et renommez le classeur si nécessaire.
Ouvrez le classeur. Vous constatez qu'un nouvel onglet nommé "OngletPerso" est ajouté dans le ruban.



L'onglet est vide car il s'agit ici d'un simple exemple de principe. Vous découvrirez de nombreuses démos dans les chapitres suivants afin d'ajouter des contrôles et compléter la personnalisation.

II-B - L'utilitaire Custom UI Editor

La méthode manuelle n'est pas toujours aisée à mettre en oeuvre.
Heureusement, il existe un utilitaire pour faciliter cette personnalisation : **Custom UI Editor Tool**.
Cette application permet de créer et modifier les fichiers customUI.xml, et les fichiers .rels sont automatiquement mis à jour sans que vous ayez besoin de les ouvrir manuellement.

Nota:
Vous devez disposer du pack de redistribution .Net framework 2.
(Un lien de téléchargement est automatiquement proposé lors de l'installation de l'utilitaire de personnalisation).
Par exemple, après avoir installé **Custom UI Editor Tool**, créez un nouveau classeur et enregistrez le au format xlsm (prenant en charge les macros).
Fermez le classeur (le fichier xml ne pourra pas être sauvegardé si le classeur est ouvert).
Lancez **Custom UI Editor**.
Utilisez le menu File/Open.
Recherchez votre classeur Excel sur le PC.
Cliquez sur le bouton **Ouvrir**.
Une page blanche s'affiche. Si votre classeur contenait déjà un ruban personnalisé, le contenu xml s'afficherait à l'écran.
Collez le code xml suivant dans la fenêtre de l'éditeur:

Xml

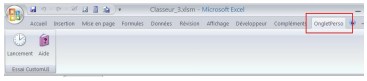
```
<customUI xmlns="http://schemas.microsoft.com/office/2006/01/customui">
<!-- Indiquez True pour masquer tous les autres onglets standards-->
<ribbon startFromScratch="false">

<tabs>
  <!-- Crée un onglet personnalisé: -->
  <!-- L'onglet va se positionner automatiquement à la fin du ruban. -->
  <!-- Utilisez insertAfterMso="NomOngletPrédéfini" pour préciser l'emplacement de l'onglet -->
  <tab id="OngletPerso" label="OngletPerso" visible="true">

    <!-- Crée un groupe -->
    <group id="Essai" label="Essai CustomUI">

      <!-- Crée un bouton: -->
      <!--onAction="ProcLancement" définit la macro qui va être déclenchée lorsque vous allez
 cliquer sur le bouton -->

      <!--imageMso="StartAfterPrevious" définit une image de la galerie Office qui va
 s'afficher sur le bouton. -->
 <!--(consultez la FAQ Excel "Comment retrouver l'ID de chaque contrôle du ruban ?" pour plus de
 détails). -->
      <!-- Nota: il est aussi possible
 d'ajouter des images externes pour personnaliser les boutons -->
      <button id="btLance01" label="Lancement" screentip="Déclenche la procédure."
      onAction="ProcLancement"
```

Xml
```
        supertip="Utilisez ce bouton pour Lancer la macro."
        size="large" imageMso="StartAfterPrevious" />

        <!-- Crée un deuxième bouton -->
        <button id="btAide01" label="Aide" screentip="Consultez l'aide."
        onAction="OuvertureAide" size="large"
        supertip="Consultez les meilleurs cours et tutoriels Office."
        imageMso="FunctionsLogicalInsertGallery"
        tag="http://office.developpez.com/" />

      </group>
    </tab>

</tabs>
</ribbon>
</customUI>
```

Cliquez sur le bouton **Generate CallBacks**. Cette action va définir une macro pour chaque paramétrage d'attribut (onAction dans notre exemple) spécifiée dans votre fichier xml. Consultez le chapitre consacré aux fonctions CallBacks pour plus de détails.

Vba
```
'Callback for btLance01 onAction
Sub ProcLancement(control as IRibbonControl)
End Sub

'Callback for btAide01 onAction
Sub OuvertureAide(control as IRibbonControl)
End Sub
```

Faites un Copier (Ctrl+C) des CallBacks pour les placer dans le presse papiers. Vous ajouterez ces macros ultérieurement dans un module standard de votre classeur.
Revenez sur l'onglet **CustomUI**.
Enregistrez les modifications xml.
Ouvrez votre classeur Excel en ignorant les éventuels messages d'erreurs.
Vous constatez qu'un nouvel onglet apparait dans le ruban.



Pour terminer, vous allez ouvrir l'éditeur de macros (Alt+F11).
Insérez un module standard.
Collez les CallBacks précédemment créés.
Il ne vous reste plus qu'à rédiger vos macros. Celles ci seront associées à l'évènement Clic sur les boutons du ruban.
Par exemple:

Vba
```
Option Explicit

Private Declare Function ShellExecute Lib "shell32.dll" Alias "ShellExecuteA" _
    (ByVal hWnd As Long, ByVal lpOperation As String, ByVal lpfile As String, _
    ByVal lpParameters As String, ByVal lpDirectory As String, ByVal nShowCmd As Long) As Long


'Callback for btLance01 onAction
Sub ProcLancement(control As IRibbonControl)
    MsgBox "Opération 'Personnaliser le ruban' réussie!"
End Sub


'Callback for btAide01 onAction
'Va ouvrir le lien indiqué dans le Tag du bouton (http://office.developpez.com/)
Sub OuvertureAide(control As IRibbonControl)
    ShellExecute 0&, vbNullString, control.Tag, vbNullString, vbNullString, 0
```

```vba
End Sub
```

Cliquez sur les boutons de l'onglet "OngletPerso" pour visualiser le résultat.
L'utilitaire customUI Editor permet également de gérer l'ajout d'images ou d'îcones personnels par le bouton "Insert Icons".
Remarque :
Pour réutiliser les exemples de cet article tout en conservant l'indentation, copiez les codes xml dans le bloc-notes et ensuite collez les dans la fenêtre de l'utilitaire customUI Editor.

III - Personnaliser le ruban

III-A - Généralités

Comme cela a déjà été partiellement évoqué dans le chapitre précédent, les fichiers customUI.xml contiennent toutes les informations permettant la personnalisation du ruban. Ensuite, les classeurs doivent être enregistrés aux nouveaux formats Office **OpenXML** (xlsm et xlam). Quand le classeur s'ouvre, l'application Excel lit le contenu du fichier xml et applique les personnalisations qui y sont définies. Lorsqu'il s'agit d'un fichier .xlsm, les personnalisations sont accessibles uniquement quand le classeur est actif. Dans un complément .xlam, les personnalisations sont accessibles en permanence, pour l'ensemble des classeurs ouverts dans l'application. Les personnalisations sont automatiquement supprimées lorsque le classeur (xlsm ou xlam) est refermé.
Cette partie du tutoriel montre :

- Comment doivent être structurés les fichiers customUI.xml.
- A quoi correspondent les différentes balises.

Pour qu'un fichier xml soit valide, le code doit respecter le schéma standard Office.
Excel propose un outil d'aide pour être informé des erreurs dans le fichier de personnalisation xml:
Utilisez le bouton Office
Bouton "Options Excel"
Menu "Options avancées"
Cochez l'option "Afficher les erreurs du complément d'interface utilisateur", dans la zone "Général".
Désormais, lorsque vous ouvrez le classeur, et si le fichier de personnalisation contient une erreur, un message s'affiche en précisant les numéros de ligne et de colonne posant soucis, ainsi qu'une description courte.
Il est important de noter que le contenu des fichiers xml est sensible à la casse. Vous devez être très attentif aux majuscules et aux minuscules dans l'utilisation des mots clefs (écriture des attributs, balises ...).
Chaque fichier customUI.xml commence systématiquement par l'élément :

```xml
<customUI xmlns=?http://schemas.microsoft.com/office/2006/01/customui?>
```

Et se termine par la balise de fermeture:

```xml
</customUI>
```

customUI est l'élément de base (root) pour le contenu du fichier xml.
Deux fonctions Callbacks peuvent être exécutées lors du chargement du ruban :
onLoad qui indique la fonction à exécuter lors du chargement du ruban.
loadImage qui spécifie la procédure de chargement des images.
Consultez le chapitre "La mise à jour dynamique du ruban" pour en savoir plus sur les paramètres additionnels qui permettent l'actualisation des contrôles.
Le ruban est défini par la balise **ribbon**: Cette balise est l'élément principal pour toutes les modifications du ruban. Vos paramètres de personnalisation doivent être insérés entre ces balises.

Xml

```xml
<customUI xmlns="http://schemas.microsoft.com/office/2006/01/customui">
   <ribbon>

   </ribbon>
</customUI>
```

L'application offre la possibilité de masquer tous les onglets prédéfinis en ajoutant l'attribut **startFromScratch** et en lui appliquant la valeur "true".

Xml

```xml
<customUI xmlns="http://schemas.microsoft.com/office/2006/01/customui">
    <!-- Indiquez True pour masquer tous les autres onglets standards-->
    <ribbon startFromScratch="true">

    </ribbon>
</customUI>
```

Ainsi, tous les onglets standards sont cachés et seules les options "Nouveau", "Ouvrir", "Enregistrer", "Documents récents", "Options Excel" apparaissent depuis le bouton "Office". Ces éléments restants sont également personnalisables (par la balise "officeMenu" qui sert à modifier les composants du bouton "Office" et à l'attribut "visible").
Plus généralement, tous les contrôles prédéfinis peuvent être réutilisés, modifiés, exécutés, masqués dans vos projets. Reportez vous au chapitre III-F-11 pour plus d'informations.
Utilisez les tabulations pour indenter votre code et faciliter la lisibilité de l'ensemble. Vous pouvez également ajouter des commentaires dans le fichier xml:

Xml

```xml
<!-- Mon commentaire -->
```

III-B - Les attributs

Le ruban, les onglets, les groupes et tous les contrôles possèdent des attributs qui permettent de paramétrer leur apparence et leur contenu. Le tableau suivant récapitule la liste des attributs, le type de donnée, une description et à quels contrôles ils peuvent être appliqués.

Attribut	Valeur	Description	S'applique à
boxStyle	horizontal ou vertical	Regroupement vertical ou horizontal de contrôles.	box
columns	donnée de type numérique	Définit le nombre de colonnes dans la galerie.	gallery
description	donnée de type texte 4096 caractères maximum	Renvoie une description pour le contrôle.	button,splitButton, toggleButton, checkBox, gallery, menu, dynamicMenu
enabled	true ou false	Autorise ou non l'utilisation du contrôle.	tous les contrôles
id	donnée de type texte 1024 caractères maximum	Identificateur unique pour chaque contrôle personnel dans le fichier de personnalisation. La chaîne de caractères	tab, group, box, button, buttonGroup, checkBox, comboBox, dropDown, dynamicMenu, editBox, gallery, lableControl, menu, menuSeparator,

		ne doit pas contenir d'espace.	separator, splitButton, toggleButton
idMso	donnée de type texte 1024 caractères maximum	Identificateur des contrôles prédéfinis.	tous les contrôles
idQ	donnée de type texte 1024 caractères maximum	Identificateur qualifié permettant de partager un ou plusieurs éléments entres différents classeurs (Dans chaque classeur, le fichier xml de personnalisation doit contenir un espace de nom identique).	tab, group, box, button, buttonGroup, checkBox, comboBox, dropDown, dynamicMenu, editBox, gallery, lableControl, menu, menuSeparator, separator, splitButton, toggleButton
image	donnée de type texte 1024 caractères maximum	Définit l'image personnelle qui va être associée au contrôle.	button, comboBox, dropDown, dynamicMenu, editBox, gallery, menu, splitButton, toggleButton
imageMso	donnée de type texte 1024 caractères maximum	Identificateur (id) d'un contrôle prédéfini dont l'image va être réutilisé pour un autre contrôle.	button, comboBox, dropDown, dynamicMenu, editBox, gallery, menu, splitButton, toggleButton
insertAfterMso	donnée de type texte 1024 caractères maximum	Insère le contrôle après l'identificateur prédéfini que vous spécifiez.	tab, group, box, button, buttonGroup, checkBox, comboBox, dropDown, dynamicMenu, editBox, gallery, lableControl, menu, menuSeparator, separator, splitButton, toggleButton
insertAfterQ	donnée de type texte 1024 caractères maximum	Insère le contrôle après l'identificateur qualifié que vous spécifiez.	tab, group, box, button, buttonGroup, checkBox, comboBox, dropDown, dynamicMenu, editBox, gallery, lableControl, menu, menuSeparator, separator, splitButton, toggleButton
insertBeforeMso	donnée de type texte 1024 caractères maximum	Insère le contrôle avant l'identificateur prédéfini que vous spécifiez.	tab, group, box, button, buttonGroup, checkBox, comboBox, dropDown, dynamicMenu, editBox, gallery, lableControl, menu, menuSeparator, separator, splitButton, toggleButton
insertBeforeQ	donnée de type texte 1024 caractères maximum	Insère le contrôle avant l'identificateur qualifié que vous spécifiez.	tab, group, box, button, buttonGroup, checkBox, comboBox, dropDown, dynamicMenu, editBox, gallery, lableControl, menu, menuSeparator,

			separator, splitButton, toggleButton
invalidateContent OnDrop	true ou false	Actualise automatiquement le contenu du contrôle lorsque celui-ci est sélectionné.	comboBox, gallery, dynamicMenu
itemHeight	donnée de type numérique 4096 (pixels) maximum	Définit la hauteur d'un élément de la galerie.	gallery
itemSize	large ou normal	Définit la taille des éléments dans les menus.	menu
itemWidth	donnée de type numérique 4096 (pixels) maximum	Définit la largeur d'un élément de la galerie	gallery
keytip	donnée de type texte 3 caractères maximum	Définit un raccourci clavier pour le contrôle (Un raccourci clavier est disponibles après l'utilisation de la touche ALT.	Les onglets, les groupes et tous les contrôles
label	donnée de type texte 1024 caractères maximum	Définit le titre du contrôle.	Les onglets, les groupes et tous les contrôles
maxLength	donnée de type numérique	Définit le nombre de caractères maximum qui peut être utilisé dans un champ de saisie.	editBox, comboBox
rows	donnée de type numérique	Définit le nombre de lignes dans la galerie.	gallery
screentip	donnée de type texte 1024 caractères maximum	Définit une petite info-bulle d'information pour le contrôle.	group, label, dialogBoxLauncher, button, splitButton, toggleButton, checkBox, editBox, comboBox, dropDown, gallery, menu, dynamicMenu
showImage	true ou false	Définit si l'image du contrôle sera affichée.	button, toggleButton, editBox, combobox, dropDown, gallery, menu, dynamicMenu
showItemImage	true ou false	Définit si l'image d'un élément du contrôle sera affichée.	comboBox, dropDown, gallery
showItemLabel	true ou false	Définit si l'étiquette d'un élément du contrôle sera affichée.	comboBox, dropDown, gallery
showLabel	true ou false	Définit si l'étiquette du contrôle sera affichée.	button, comboBox, dropDown, dynamicMenu, editBox, gallery, labelControl,

			menu, splitButton, toggleButton
size	large ou normal	Définit la taille du contrôle.	tous les contôles
sizeString	donnée de type texte 1024 caractères maximum	Chaîne de caractères permettant d'ajuster la largeur du contrôle.	editBox, comboBox, gallery
startFromScratch	true ou false	La valeur "true" masque les onglets prédéfinis du ruban et quelques éléments du bouton "Office". La valeur "false" (donnée pas défaut) affiche les onglets prédéfinis.	ruban
supertip	donnée de type texte	Définit une info-bulle complémentaire pour le contrôle. Celui-ci s'affichera à la suite des informations indiquées dans l'attribut "screentip". Utilisez la chaîne de caractères
pour créer un retour à la ligne.	group, label, dialogBoxLauncher, button, splitButton, toggleButton, checkBox, editBox, comboBox, dropDown, gallery, menu, dynamicMenu
tag	donnée de type texte 1024 caractères maximum	Information personnelle complémentaire pour le contrôle.	tous les contrôles
title	donnée de type texte	Définit un titre.	menuSeparator
visible	true ou false	Définit si le contrôle est visible.	onglet, groupe et tous les contrôles
xmlns	donnée de type texte	Définit l'espace de nom (nameSpace) et permet d'identifier qu'il s'agit d'un fichier de ruban (RibbonX) Office: http://schemas.microsoft.com/office/2006/01/customui	customUI

III-C - Les onglets

Les onglets sont les éléments de base dans le ruban. La balise **tabs** représente l'ensemble des onglets prédéfinis et personnalisés. La balise **tab** définit un onglet particulier (prédéfini ou personnalisé), identifiable par son **id**. Comme pour tout type de contrôle, il existe 3 types d'attributs identificateurs (**id**):

* id: Pour un élément personnalisé.
* idMso: Pour un élément prédéfini (standard).
* idQ: Pour un élément partagé entre plusieurs classeurs.

Remarque:
Chaque identificateur **id** doit impérativement être unique dans le fichier xml.
Cet exemple masque l'onglet prédéfini "Révision" en donnant la valeur "false" à l'attribut "visible" et ajoute un nouvel onglet nommé "OngletPerso".

Xml

```
<customUI xmlns="http://schemas.microsoft.com/office/2006/01/customui">
```

Xml
```
<ribbon startFromScratch="false">

    <tabs>
        <!-- Masque l'onglet prédéfini Révision -->
        <tab idMso="TabReview" visible="false"/>

        <!-- Ajoute un nouvel onglet nommé OngletPerso -->
        <tab id="OngletPerso" label="Mes fonctions préférées" visible="true">
        </tab>
    </tabs>

</ribbon>
</customUI>
```

Vous pouvez également spécifier la position d'un onglet personnalisé par rapport aux onglets standards, grâce aux attributs "insertAfterMso" et "insertBeforeMso":

Xml
```
<!-- Va positionner l'onglet OngletPerso après l'onglet Accueil (TabHome) -->
<tab id="OngletPerso" label="Mes fonctions préférées" insertAfterMso="TabHome" visible="true">
```

Voici la liste des onglets standards dans Excel2007:
TabHome: onglet **Accueil**
TabInsert: onglet **Insertion**
TabPageLayoutExcel: onglet **Mise en page**
TabFormulas: onglet **Formules**
TabData: onglet **Données**
TabReview: onglet **Révision**
TabView: onglet **Affichage**
TabDeveloper: onglet **Développeur**
TabAddIns: onglet **Compléments**
Si l'intitulé des onglets prédéfinis ne vous convient pas, ceux-ci peuvent être modifiés depuis un fichier de personnalisation.
Cet exemple remplace la description de l'onglet "Développeur" (TabDeveloper) par la chaîne "Gestion macros et XML".

Xml
```
<customUI xmlns="http://schemas.microsoft.com/office/2006/01/customui">
   <ribbon startFromScratch="false">
      <tabs>
        <tab idMso="TabDeveloper" label="Gestion macros et XML" />
      </tabs>
   </ribbon>
</customUI>
```

Si vous avez créé plusieurs compléments **.xlam** dans Excel 2007 et que chacun d'entre eux contient un onglet personnel qui s'affiche dans le ruban, vous pouvez être intéressé de les regrouper dans un même onglet, lorsqu'ils sont ouverts simultanément.
Pour réaliser cette action, le fichier CustomUI.xml de chaque complément doit contenir certaines informations identiques:

Un nom identique (xmls:A="Mes macros complémentaires")
Un qualificateur identique (idQ="A:CibleAddIn")
L'attribut idQ (qualified ID) permet de spécifier un élément commun à plusieurs classeurs.

Xml
```
<customUI xmlns="http://schemas.microsoft.com/office/2006/01/customui"
   xmlns:A="Mes macros complémentaires">
```

Xml
```
<ribbon startFromScratch="false">
  <tabs>

    <tab idQ="A:CibleAddIn" label="Mes macros complémentaires" visible="true">



    </tab>

  </tabs>
</ribbon>
</customUI>
```

Vous pouvez ensuite ajouter les groupes et vos options personnelles dans chaque classeur. Par exemple:

Xml
```
<customUI xmlns="http://schemas.microsoft.com/office/2006/01/customui"
   xmlns:A="Mes macros complémentaires">

<ribbon startFromScratch="false">
  <tabs>

    <tab idQ="A:CibleAddIn" label="Mes macros complémentaires" visible="true">
      <group id="Projet01" label="Projet 01">

        <button id="btnLance01" label="Lancement 01"
         onAction="ProcLancement01"
         size="normal" imageMso="Repeat" />

     </group>
    </tab>

  </tabs>
</ribbon>
</customUI>
```

Désormais lorsque vous ouvrez un complément contenant l'attribut (idQ) commun, un nouveau groupe est ajouté dans l'onglet "Mes macros complémentaires".

III-D - Les groupes

Chaque groupe est défini par la balise **group**. Ils apparaissent à l'intérieur l'onglet, dans le même ordre que vous les avez définis dans le fichier de personnalisation.

Xml
```
<customUI xmlns="http://schemas.microsoft.com/office/2006/01/customui">
<ribbon startFromScratch="false">

    <tabs>
        <!-- Ajoute un nouvel onglet nommé OngletPerso -->
        <tab id="OngletPerso" label="Mes fonctions préférées" visible="true">

            <!-- Crée un groupe -->
            <group id="Gr01" label="Groupe 01">
            </group>

            <!-- Crée un deuxième groupe -->
            <group id="Gr02" label="Groupe 02">
            </group>

        </tab>
    </tabs>
```

```Xml
</ribbon>
</customUI>
```

Vous pouvez également récupérer un groupe prédéfini et l'insérer dans un autre onglet.
Ce code affiche le groupe "bibliothèque de fonctions" dans un onglet personnalisé:

```Xml
<customUI xmlns="http://schemas.microsoft.com/office/2006/01/customui">
<ribbon startFromScratch="false">

    <tabs>
        <!-- Ajoute un nouvel onglet nommé OngletPerso -->
        <tab id="OngletPerso" label="Mes fonctions préférées" visible="true">

            <!-- Ajoute le groupe standard 'bibliothèque de fonctions' -->
            <group idMso="GroupFunctionLibrary" />

        </tab>
    </tabs>

</ribbon>
</customUI>
```

Tous les autres cas de figure sont réalisables dans vos projets:
* Ajouter un groupe personnel dans un onglet prédéfini.
* Afficher un groupe prédéfini dans un autre onglet prédéfini.
* Masquer un groupe prédéfini.

III-F - Les contrôles

Une fois définis l'onglet et le groupe (qu'ils soient de type standards ou personnalisés) dans le fichier customUI.xml, vous allez pouvoir paramétrer les contrôles. Une large panoplie d'objets, décrits dans les sous chapitres suivants, est mise à votre disposition.

III-F-1 - labelControl

Le **labelControl** est une zone d'étiquette sans action, utilisée pour afficher des informations ou pour par exemple servir d'en-tête lorsque vous créez des séparateurs dans le ruban.

```Xml
<customUI xmlns="http://schemas.microsoft.com/office/2006/01/customui">
    <ribbon startFromScratch="false">
        <tabs>
            <tab id="OngletPerso" label="Exemples de labelControl">
                <group id="GR1" label="Test">
                    <labelControl id="LC01" label=" Mon texte 01" />

     <separator id="Sep01"/>

     <labelControl id="LC02" label="Montexte 02" />
                </group>
            </tab>
        </tabs>
    </ribbon>
</customUI>
```

III-F-2 - comboBox

Le contrôle **comboBox** (zones de liste déroulante) permet de choisir une donnée dans une liste de choix, mais également d'écrire une valeur manuellement dans le champ de saisie.
Le contenu de la liste peut être défini en statique, comme dans l'exemple ci-dessous, ou dynamiquement par VBA.
Chaque **item** correspond à un élément de la liste déroulante :

Xml

```
<customUI xmlns="http://schemas.microsoft.com/office/2006/01/customui">
    <ribbon startFromScratch="false">
        <tabs>
            <tab id="OngletPerso" label="Exemples ComboBox">
                <group id="GR1" label="Test">
   <comboBox id="CB1" label="Coefficient" onChange="ChangeCB1" >
         <item id="it1" label="1" imageMso="MacroPlay" />
         <item id="it2" label="1,25" imageMso="CellsInsertDialog" />
         <item id="it3" label="1,5" imageMso="FileOpen" />
         <item id="it4" label="1,8" imageMso="Spelling" />
   </comboBox>
                </group>
            </tab>
        </tabs>
    </ribbon>
</customUI>
```

La fonction CallBack associée à l'attribut onChange, à placer dans un module standard du classeur:

Vba

```
'Callback for CB1 onChange
Sub ChangeCB1(control As IRibbonControl, text As String)
    MsgBox text
End Sub
```

Le code xml suivant gère l'alimentation de la liste déroulante par VBA. Il permet d'afficher les données contenues dans colonne A de la Feuil1, dans le comboBox.
Le rappel **getItemCount** définit le nombre d'items à afficher dans le contrôle.
L'attribut **invalidateContentOnDrop** actualise automatiquement le contenu du contrôle lorsque celui-ci est sélectionné.

Xml

```
<customUI xmlns="http://schemas.microsoft.com/office/2006/01/customui" >
<ribbon startFromScratch="false">
<tabs>

<tab id="OngletPerso" label="OngletPerso" visible="true">
   <group id="Projet01" label="Projet 01">

      <!-- getItemCount="NbItemCombo" va définir le nombre d'items dans la combobox. -->
      <!-- getItemLabel="ComboLabel" permet d'alimenter la combobox. -->
      <!-- invalidateContentOnDrop="true" permet la mise à jour automatique du contrôle. -->

      <comboBox id="Combo1" label="Choix : " getItemCount="NbItemCombo" getItemLabel="ComboLabel"
         invalidateContentOnDrop="true" />
   </group>
</tab>

</tabs>
</ribbon>
</customUI>
```

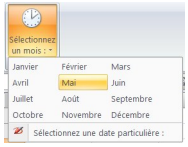
Les fonctions de rappel associées au code xml, à placer dans un module standard du classeur :

vba

```
'Callback for Combo1 getItemCount
Sub NbItemCombo(control As IRibbonControl, ByRef returnedVal)
    'Définit le nombre d'éléments dans la combobox

    'Récupère le nombre de données dans la colonne A.
    returnedVal = Feuil1.Range("A65536").End(xlUp).Row
End Sub


'Callback for Combo1 getItemLabel
'Index est défini par la valeur saisie dans la fonction de rappel NbItemCombo
Sub ComboLabel(control As IRibbonControl, index As Integer, ByRef returnedVal)

    If Feuil1.Range("A65536").End(xlUp).Row = 0 Then Exit Sub

    'Alimente le Combobox à partir des données de la plage de cellules
    returnedVal = Feuil1.Cells(index + 1, 1)
End Sub
```

III-F-3 - gallery

Le contrôle **gallery** est un tableau qui contient d'autres types de contrôles. Vous pouvez définir le nombre de lignes et de colonnes dans votre grille, ainsi que les dimensions.



Cet exemple permet de choisir un nom de mois dans le tableau ou de sélectionner une date particulière. La sélection s'affiche dans la cellule active. La partie de la procédure qui affiche le calendrier pour définir une date particulière nécessite de disposer de l'ocx MSCOMCT2.ocx.

Xml

```
<customUI xmlns="http://schemas.microsoft.com/office/2006/01/customui" onLoad="objRuban">

  <ribbon>
    <tabs>
      <tab id="Planning" label="Gestion planning" >
        <group id="Calendrier" label="Le calendrier">

  <gallery id="gallery01"
   size="large"
   imageMso="StartAfterPrevious"
   label="Sélectionnez un mois :"
   columns="3"
   rows="4"
   getItemCount="NbMois"
   showItemLabel="true"
   getItemLabel="LabelMois"
   onAction="SelectionMois" >

     <button id="button01"
   label="Sélectionnez une date particulière :"
   imageMso="InkingStart"
   onAction="ChargeDtPicker"/>
  </gallery>

        </group>
      </tab>
    </tabs>
  </ribbon>
</customUI>
```

Les fonctions de rappel associées au code xml, à placer dans un module standard du classeur :

Vba

```
Option Explicit

Public MonRuban As IRibbonUI
Dim Usf As Object


'Callback for customUI.onLoad
'Est déclenché lors du chargement du ruban personnalisé.
Sub objRuban(ribbon As IRibbonUI)
    Set MonRuban = ribbon
End Sub


'Callback for gallery01 getItemCount
Sub NbMois(control As IRibbonControl, ByRef returnedVal)
    returnedVal = 12
End Sub


'Callback for gallery01 getItemLabel
Sub LabelMois(control As IRibbonControl, index As Integer, ByRef returnedVal)
    returnedVal = Application.Proper(MonthName(index + 1))
End Sub


'Callback for gallery01 onAction
Sub SelectionMois(control As IRibbonControl, id As String, index As Integer)
    ActiveCell.Value = Application.Proper(MonthName(index + 1))
End Sub


'Callback for button01 onAction
Sub ChargeDtPicker(control As IRibbonControl)
    LancementProcedureCalendrier
End Sub



'Cet exemple utilise le contrôle DataPicker
'Vous devez disposer de l'ocx MSCOMCT2.ocx sur votre poste
Sub LancementProcedureCalendrier()
    Dim X As Object
    Dim NomdtPicker As String

    NomdtPicker = "DtPicker1"
    Set X = UserForm_Et_DataPicker_Dynamique(NomdtPicker)

    X.Show

    ThisWorkbook.VBProject.VBComponents.Remove Usf
    Set Usf = Nothing
End Sub


Function UserForm_Et_DataPicker_Dynamique(NomObjet As String) As Object
    Dim Obj As Object
    Dim j As Integer
    Set Usf = ThisWorkbook.VBProject.VBComponents.Add(3)
    With Usf
        .Properties("Caption") = "Mon calendrier"
        .Properties("Width") = 130
        .Properties("Height") = 40
    End With

    Set Obj = Usf.Designer.Controls.Add("MSComCtl2.DTPicker.2")

    With Obj
```

Vba
```
        .Left = 0: .Top = 0: .Width = 130: .Height = 20
        .Name = NomObjet
        .CalendarBackColor = &HFF00FF
    End With

    With Usf.CodeModule
        j = .CountOfLines
        .insertlines j + 1, "Sub " & NomObjet & "_Change()"
        .insertlines j + 2, "    ActiveCell.Value = Format(DateSerial(Year(" _
            & NomObjet & "), Month(" & NomObjet & "), Day(" _
            & NomObjet & ")), " & Chr(34) & "dd mmmm yyyy" & Chr(34) & ")"
        'Option pour refermer l'userform après l'insertion de la date.
        '.insertlines j + 3, "    Unload Me"
        .insertlines j + 4, "End Sub"
    End With

    VBA.UserForms.Add (Usf.Name)
    Set UserForm_Et_DataPicker_Dynamique = UserForms(UserForms.Count - 1)

End Function
```

III-F-4 - button

Le contrôle button déclenche une action lorsqu'il reçoit l'évènement clic.
Vous pouvez récupérer des commandes standards (par l'attribut idMso) ou simplement utiliser l'image des commandes (imageMso) pour vos menus.
La troisième option consiste à créer des boutons totalement personnalisés, à partir de l'attribut id. Dans ce cas, vous pouvez ensuite compléter la mise en forme en chargeant des images externes (voir le chapitre consacré aux images).
Il est par également possible de modifier les labels des boutons standards afin de rendre les intitulés plus explicites.
Le code xml ci-dessous présente plusieurs variantes de boutons :

* Réutilisation d'un bouton standard.
* Réutilisation d'un bouton standard et modification de l'intitulé.
* Création d'un bouton personnel.
* Création d'un bouton personnel et réutilisation de l'image d'une commande standard.

Xml
```
<customUI xmlns="http://schemas.microsoft.com/office/2006/01/customui">
   <ribbon startFromScratch="false">
      <tabs>
        <tab id="TB01" label="Test">
   <group id="GR01" label="Essais de boutons">

    <!-- réutilise un bouton standard -->
    <button idMso="CalculateNow" size="normal" visible="true" />

    <!-- réutilise un bouton standard en modifiant l'intitulé -->
    <button idMso="Spelling" label="Intitulé personnalisé" size="normal"/>

    <!-- crée un bouton personnel et réutilise l'image de la commande 'Orthographe' -->
    <button id="BtPerso02" imageMso="Spelling" onAction="MaProcedure"
       label="Lancez la procédure" size="normal"/>

       <separator id="Sep01"/>

    <!-- crée un bouton personnel -->
    <button id="BtPerso01" onAction="MaProcedure"
       label="Lancez la procédure" keytip="DVP"
       supertip="Ce bouton possède&#13;un raccourci clavier : DVP"
       size="normal"/>

   </group>
        </tab>
      </tabs>
```

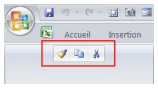
```Xml
    </ribbon>
</customUI>
```

Et la macro associée à la fonction onAction, à placer dans un module standard:

```Vba
'Callback for MaProcedure onAction
Sub MaProcedure(control As IRibbonControl)
    MsgBox "Vous avez cliqué sur le bouton " & control.ID
End Sub
```

III-F-5 - buttonGroup

Le contrôle **buttonGroup** sert à rassembler plusieurs boutons dans un même ensemble.

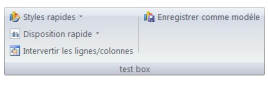


Consultez le chapitre consacré aux toggleButton pour visualiser un exemple d'utilisation.

III-F-6 - Box

Le contrôle **box** est utilisé pour agencer le contenu du ruban en regroupant des contrôles horizontalement ou verticalement :

```Xml
<customUI xmlns="http://schemas.microsoft.com/office/2006/01/customui">
<ribbon startFromScratch="false">

    <tabs>
        <tab id="TB01" label="Test">

  <group id="GR01" label="test box" >
   <box id="Box01" boxStyle="vertical">
    <control idMso="ChartStylesGallery"/>
    <control idMso="ChartLayoutGallery"/>
    <control idMso="ChartSwitchRowColumn"/>
   </box>

   <separator id="Sep01"/>

   <control idMso="ChartSaveTemplates" />
  </group>
        </tab>
    </tabs>

</ribbon>
</customUI>
```



III-F-7 - checkBox

Le contrôle **checkBox** est une case à cocher qui renvoie la valeur "vrai" lorsque l'option est sélectionnée. Dans le cas contraire, la valeur Faux est renvoyée.

Xml

```
<customUI xmlns="http://schemas.microsoft.com/office/2006/01/customui">
   <ribbon startFromScratch="false">
      <tabs>
         <tab id="tab1" label="Test" >
            <group id="GR01" label="Essai de checkbox">
               <checkBox id="CheckBox1"
                  label="Option"
                  onAction="Validation" />
            </group>
         </tab>
      </tabs>
   </ribbon>
</customUI>
```

Ensuite, placez cette procédure dans un module standard du classeur.

Vba

```
'Callback for CheckBox1 onAction
Sub Validation(control As IRibbonControl, pressed As Boolean)
    MsgBox control.ID & vbCrLf & pressed
End Sub
```

III-F-8 - editBox

Le contrôle **editBox** est un champ de saisie. Vous pouvez notamment spécifier un nombre de caractères maximum et la largeur (en taille de chaine de caractères).

Xml

```
<customUI xmlns="http://schemas.microsoft.com/office/2006/01/customui">
    <ribbon startFromScratch="false">
        <tabs>
            <tab id="OngletPerso" label="Test">
                <group id="GR01" label="Exemple utilisation editBox">

   <box id="Box01" boxStyle="horizontal">
    <editBox id="editBox01"
     label="Saisissez le code :"
     onChange="RecupDonnee"
     screentip="Le code doit contenir 5 caractères maximum."
     sizeString="99999"
     maxLength="5"/>
   </box>

   <button id="button01"
    onAction="ValidationCode"
    label="Validez le code"
    imageMso="FileServerTasksMenu"
    size="normal"/>

                </group>
            </tab>
        </tabs>
    </ribbon>
</customUI>
```

La fonction d'appel à placer dans un module standard du classeur :

Vba

```
Option Explicit
```

Vba

```
Dim Cible As String


'Callback for editBox01 onChange
Sub RecupDonnee(control As IRibbonControl, text As String)
    Cible = text
End Sub


'Callback for button01 onAction
Sub ValidationCode(control As IRibbonControl)
    MsgBox Cible
End Sub
```

III-F-9 - toggleButton

Le contrôle **toggleButton** (Bouton bascule) est un bouton qui change d'aspect lorsque vous cliquez dessus. Il permet de renvoyer les valeurs:

Vrai (Lorsque le bouton est activé)
Faux (Lorsque le bouton est désactivé)

Un exemple d'utilisation :

Xml

```
<customUI xmlns="http://schemas.microsoft.com/office/2006/01/customui"
onLoad="objRuban" >

   <ribbon startFromScratch="false">
      <tabs>
        <tab id="Tab01" label="Test">
   <group id="GR01" label="Essais de buttonGroup et toggleButton">

    <buttonGroup id="BG01">

     <toggleButton id="TB01" imageMso="FormatPainter"
      getPressed="ProcPressed"
      onAction="MaProcedure"/>

     <toggleButton id="TB02" imageMso="Copy"
      getPressed="ProcPressed"
      onAction="MaProcedure"/>

     <toggleButton id="TB03" imageMso="Cut"
      getPressed="ProcPressed"
      onAction="MaProcedure"/>

    </buttonGroup>

   </group>
        </tab>
      </tabs>
   </ribbon>
</customUI>
```

L'argument **pressed** prend la valeur "vrai" quand le togleButton est activé et "true" lorsque vous recliquez sur le bouton.

Xml

Xml
```
Option Explicit

Public MonRuban As IRibbonUI


'Callback for customUI.onLoad
Sub objRuban(ribbon As IRibbonUI)
    Set MonRuban = ribbon
End Sub


'Callback for toggleButton.getPressed
Sub ProcPressed(control As IRibbonControl, ByRef returnedVal)
    'returnedVal=
End Sub


'Callback for toggleButton.onAction
Sub MaProcedure(control As IRibbonControl, pressed As Boolean)
    MsgBox control.ID & vbCrLf & pressed
End Sub
```

III-F-10 - dropDown

Le contrôle **dropDown** permet aussi d'afficher un menu déroulant.
Contrairement au comboBox, le contrôle dropDown :
* Peut contenir des éléments de type bouton (button).
* Ne peut pas utiliser l'attribut "invalidateContentOnDrop".
* Ne possède pas de champ de saisie manuel.

Xml
```
<customUI xmlns="http://schemas.microsoft.com/office/2006/01/customui">
    <ribbon startFromScratch="false">
        <tabs>
            <tab id="OngletPerso" label="Exemples dropDown">
                <group id="GR1" label="Test">
   <dropDown id="Drop01" label="Choix" onAction="NomProcedure" >
         <item id="it1" label="AA" imageMso="MacroPlay" />
         <item id="it2" label="BB" imageMso="CellsInsertDialog" />
    <item id="it3" label="CC" imageMso="FileOpen" />
         <item id="it4" label="DD" imageMso="Spelling" />
   </dropDown>
                </group>
            </tab>
        </tabs>
    </ribbon>
</customUI>
```

Ensuite, placez cette procédure dans un module standard du classeur.

Vba
```
'Callback for Drop01 onAction
Sub NomProcedure(control As IRibbonControl, id As String, index As Integer)
    MsgBox control.id & vbCrLf & id & vbCrLf & index
End Sub
```

III-F-11 - command

Le contrôle **command** est utilisé pour la personnalisation et la réaffectation des commandes prédéfinies.
Le code suivant désactive le bouton "Couleur de remplissage", grâce à l'argument "enabled" :

Xml

```
<customUI xmlns="http://schemas.microsoft.com/office/2006/01/customui">

  <commands>
    <!-- Désactive le bouton "Couleur de remplissage" -->
    <command idMso="CellFillColorPicker" enabled="false" />
  </commands>

<ribbon startFromScratch="false">

</ribbon>
</customUI>
```

Les paramètres **Commands** doivent être placés en début de fichier, et avant la ligne ribbon startFromScratch="boolean".
Vous remarquerez que cette action désactive le bouton, à la fois dans l'onglet "Accueil" et aussi dans la mini barre d'outils.
Sur le même principe, l'utilisation d'onAction permet de réattribuer une autre procédure aux contrôles prédéfinis de type button, toggleButton et checkBox:

Xml

```
<customUI xmlns="http://schemas.microsoft.com/office/2006/01/customui">

  <commands>
    <!-- Réattribue l'action du bouton "Insérer un lien hypertexte" -->
    <command idMso="HyperlinkInsert" onAction="MacroDeRemplacement" />
  </commands>

<ribbon startFromScratch="false">

</ribbon>
</customUI>
```

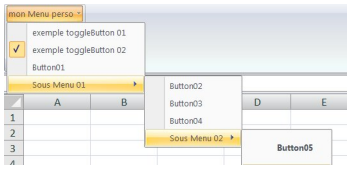
Dans un module standard du classeur :

Vba

```
'Callback for HyperlinkInsert onAction
Sub MacroDeRemplacement(control As IRibbonControl, ByRef cancelDefault)
    MsgBox "Procédure de remplacement du bouton : " & vbCrLf & control.id

    'Attribuez la valeur False à l'argument cancelDefault pour réactiver par VBA
    'la fonctionnalité prédéfinie:

    'cancelDefault = False
End Sub
```

III-F-12 - menu

Le contrôle **menu** crée une barre de menu constituée de boutons ou d'autres menus imbriqués sous forme d'arborescence.



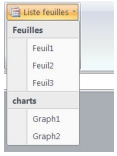
Voici un exemple de menus en cascade:

Xml

```
<customUI xmlns="http://schemas.microsoft.com/office/2006/01/customui">
   <ribbon startFromScratch="false">

 <tabs>
        <tab id="Tab01" label="Test">
   <group id="GR01" label="Essais de menus">

    <menu id="Menu01" label="mon Menu perso" itemSize="normal">
        <toggleButton id="ToggleButton01" label="exemple toggleButton 01" />
     <toggleButton id="ToggleButton02" label="exemple toggleButton 02" />
        <button id="bt01" label="Button01" />

      <menu id="SousMenu01" label="Sous Menu 01" itemSize="normal">
            <button id="bt02" label="Button02" />
            <button id="bt03" label="Button03" />
            <button id="bt04" label="Button04" />

       <menu id="SousMenu02" label="Sous Menu 02" itemSize="large">
             <button id="bt05" label="Button05" />
                <button id="bt06" label="Button06" />
            </menu>
         </menu>
    </menu>

   </group>
        </tab>
      </tabs>

   </ribbon>
</customUI>
```

III-F-13 - dynamicMenu

Le contrôle **dynamicMenu** affiche une barre de menus, dont la particularité réside dans sa structure, qui peut être créée ou modifiée dynamiquement par VBA, contrairement aux autres contrôles proposant aussi des listes de choix (dropDown, comboBox, gallery).
L'attribut **getContent** déclenche une procédure VBA qui va définir le code xml de personnalisation à la volée.
Des contrôles **dynamicMenu** peuvent être imbriqués afin de présenter des arborescences dans vos menus.
Le code suivant crée un menu dynamique qui liste les feuilles de calcul et les feuilles graphiques du classeur actif. Chaque type de feuille est regroupé dans un menu de séparation (contrôle menuSeparator). L'élément sélectionné active la feuille correspondante.



Xml

```
<customUI xmlns="http://schemas.microsoft.com/office/2006/01/customui">
<ribbon startFromScratch="false">

    <tabs>
```

Xml

```
        <tab id="Ongletperso" label="Liste feuilles">

            <group id="GR01" label="Test">

            <dynamicMenu id="ListeDynamique"
                label="Liste feuilles"
                getContent="CreationMenuDynamique"
                invalidateContentOnDrop="true"
                size="normal"
                imageMso="PrintTitles"/>
            </group>

        </tab>
    </tabs>

</ribbon>
</customUI>
```

Ensuite, placez ces procédures dans un module standard du classeur.

Xml

```
Option Explicit

'Callback for ListeDynamique getContent
'Procédure pour construire le menu dynamique
Public Sub CreationMenuDynamique(ctl As IRibbonControl, ByRef content)
    'ouverture de la balise menu
    content = "<menu xmlns=""http://schemas.microsoft.com/office/2006/01/customui"">"

    'liste les feuilles de calcul du classeur actif
    content = content & ListeFeuilles(ActiveWorkbook)

    'liste les feuilles graphiques du classeur actif
    content = content & ListeCharts(ActiveWorkbook)

    'fermeture de la balise
    content = content & "</menu>"
End Sub



Private Function ListeFeuilles(Wb As Workbook) As String
    Dim strTemp As String
    Dim Ws As Worksheet

    ' Insertion d'un titre de menu
    strTemp = "<menuSeparator id=""Feuilles"" title=""Feuilles""/>"

    ' ajoute un bouton dans le menu pour chaque feuille du classeur
    For Each Ws In Wb.Worksheets
        strTemp = strTemp & _
            "<button " & _
            CreationAttribut("id", "Bt" & Ws.Name) & " " & _
            CreationAttribut("label", Ws.Name) & " " & _
            CreationAttribut("tag", Ws.Name) & " " & _
            CreationAttribut("onAction", "ActivationFeuille") & "/>"
    Next

    ListeFeuilles = strTemp
End Function


'Liste les feuilles graphiques contenues dans le classeur
Private Function ListeCharts(Wb As Workbook) As String
    Dim strTemp As String
    Dim Ch As Chart

    If Wb.Charts.Count = 0 Then Exit Function
```

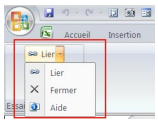
Xml

```
    strTemp = "<menuSeparator id=""charts"" title=""charts""/>"

    For Each Ch In Wb.Charts
        strTemp = strTemp & _
            "<button " & _
            CreationAttribut("id", "Bt" & Ch.Name) & " " & _
            CreationAttribut("label", Ch.Name) & " " & _
            CreationAttribut("tag", Ch.Name) & " " & _
            CreationAttribut("onAction", "ActivationFeuille") & "/>"
    Next

    ListeCharts = strTemp
End Function


Function CreationAttribut(strAttribut As String, Donnee As String) As String
    CreationAttribut = strAttribut & "=" & Chr(34) & Donnee & Chr(34)
End Function



'Active la feuille sélectionnée lorsque vous cliquez sur un nom
'dans le menu.
Sub ActivationFeuille(control As IRibbonControl)
    Sheets(control.Tag).Activate
End Sub
```

Il est important de noter que les menus sont automatiquement actualisés lorsque vous attribuez la valeur true à l'argument **invalidateContentOnDrop**.

III-F-14 - splitButton

Le contrôle **splitButton** présente une liste de boutons dans un menu déroulant.



Xml

```
<customUI xmlns="http://schemas.microsoft.com/office/2006/01/customui">
   <ribbon startFromScratch="false">

 <tabs>
        <tab id="Tab01" label="Test">
   <group id="GR01" label="Essais de splitButton">

    <splitButton id="Split01" >
    <menu>
     <button id="button01" imageMso="FileLinksToFiles" label="Lier" />
     <button id="button02" imageMso="FileExit" label="Fermer" />
     <button id="button03" imageMso="FileProperties" label="Aide" />

    </menu>
    </splitButton>

   </group>
        </tab>
      </tabs>

   </ribbon>
</customUI>
```

III-F-15 - separator

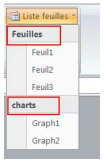
Le contrôle **separator** ajoute une barre verticale afin de séparer des contrôles dans un groupe. Vous pouvez ainsi créer des colonnes et différencier de manière visuelle plusieurs objets.
Ce code insère un séparateur entre les 2ème et 3ème boutons placés dans un groupe personnalisé.

Xml

```
<customUI xmlns="http://schemas.microsoft.com/office/2006/01/customui">
    <ribbon startFromScratch="false">
        <tabs>
            <tab id="OngletPerso" label="Exemple de séparateur">
                <group id="GR1" label="Test">
                 <button id="BT01" label="Lancement"
         onAction="Macro01"
    size="normal" imageMso="CalculateNow" />
                 <button id="BT02" label="Répéter"
         onAction="Macro02"
    size="normal" imageMso="Repeat" />

     <separator id="Sep01"/>

                 <button id="BT03" label="Connection"
         onAction="Macro03"
    size="normal" imageMso="HyperlinkInsert" />

                </group>
            </tab>
        </tabs>
    </ribbon>
</customUI>
```

III-F-16 - menuSeparator

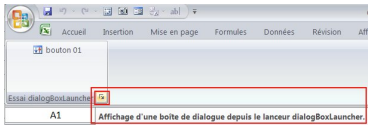
Le contrôle **menuSeparator** crée un titre dans les barres de menus. Cet objet est pratique pour présenter les groupes d'éléments dans les listes.
Reportez vous au chapitre consacré au contrôle **dynamicMenu** si vous recherchez un exemple d'utilisation.



Si vous ne précisez par l'argument "title", les éléments sont séparés par de simples traits horizontaux.

III-F-17 - dialogBoxLauncher

Le contrôle **dialogBoxLauncher** ajoute un bouton de lancement dans l'angle inférieur droit du groupe.
Dans le ruban standard, ce bouton permet d'afficher les boîtes de dialogue d'interface utilisateur. De la même manière, il est possible de lancer un **UserForm** ou un MsgBox depuis ce contrôle. C'est la macro définie dans l'attribut "onAction" qui va déclencher l'affichage des boîtes de dialogue.



Voici un exemple qui insère un bouton standard de tri personnalisé et ajoute un bouton lanceur dans le groupe.

Xml

```
<customUI xmlns="http://schemas.microsoft.com/office/2006/01/customui">
```

Xml
```
    <ribbon startFromScratch="false">
       <tabs>
          <tab id="TAB01" label="Test" >
             <group id="GR01" label="Essai dialogBoxLauncher">
                <button id="BT01"
                   idMso="SortCustomExcel"
                   label="bouton 01"
                   size="normal" />
                <dialogBoxLauncher>
                   <button id="BT02"

 screentip= "Affichage d'une boîte de dialogue depuis le lanceur dialogBoxLauncher."
                         onAction="MaProcedure02" />
                </dialogBoxLauncher>
             </group>
          </tab>
       </tabs>
    </ribbon>
</customUI>
```

Placez la macro "MaProcedure02" dans un module standard. Le code contient les instructions qui vont afficher le MsgBox (ou l'UserForm).

Vba
```
Public Sub MaProcedure02(ByVal control As IRibbonControl)

    MsgBox "MsgBox affiché depuis le lanceur "

    'Pour lancer un UserForm
    'UserForm1.Show
End Sub
```

IV - Les fonctions d'appel: Callbacks

Une partie des attributs contenus dans le fichier xml de personnalisation peuvent être associés à des fonctions d'appel VBA (CallBacks).
Les paramètres définis dans votre code xml sont alors liés à des procédures placées dans un **module standard** du classeur. Vous pouvez déclencher une macro depuis le ruban (à l'aide des évènements onAction, onChange ...), mais également paramétrer et modifier les attributs des contrôles par VBA, en ajoutant le préfixe **get**.
La caractéristique d'un attribut **getNomAttribut** est de pouvoir mettre à jour dynamiquement sa propriété **NomAttribut**.
Par exemple, au lieu d'écrire le paramètre itemHeight="LaValeur" en dur dans le fichier xml de personnalisation, vous écrirez getItemHeight="NomMacroDefinitionHauteurItem".
Ensuite, il vous restera à placer la fonction d'appel dans un module standard du classeur:
Sub GetItemHeight(control As IRibbonControl, ByRef returnedVal)
Vous trouverez la liste des fonctions CallBack, pour chaque type de contrôle, dans l'article de **Christophe Warin** :
Création de rubans personnalisés sous Microsoft Access 2007.
Nota :
L'utilitaire custom UI Editor, présenté dans le chapitre II-B, est intéressant car le bouton "CallBacks" affiche automatiquement la structure des procédures, après avoir défini les attributs des contrôles.
En complément, voici une description des arguments présents dans les fonctions CallBacks:
L'argument **control**.
Exemple: Sub ___OnAction(control As IRibbonControl)
Cet argument identifie le contrôle manipulé par VBA. Il possède 3 propriétés en lecture seule:
* **id** est l'identifiant unique qui permet d'identifier chaque contrôle. Si vous attribuez la même procédure à plusieurs contrôles du fichier xml de personnalisation, l'id vous permet de repérer lequel est en cours d'utilisation.
* **tag** est l'information personnelle complémentaire du contrôle, que vous avez défini dans le fichier xml.
* **context** représente la fenêtre active d'ou a été déclenchée la fonction d'appel. Par exemple :

Vba

```
'Callback for btnLance01 onAction
Sub ProcLancement01(control As IRibbonControl)
    Dim Ww As Window

    Set Ww = control.Context
    MsgBox Ww.Caption
End Sub
```

Le mot clé **Byref**.
Exemple : Sub ___GetEnabled(control As IRibbonControl, ByRef enabled)
Lorsqu'un argument est précédé de Byref, cela signifie que la donnée spécifiée dans votre fonction va être utilisée pour mettre à jour l'attribut du contrôle.
L'argument **pressed**.
Exemple : Sub ____OnAction(control As IRibbonControl, pressed As Boolean)
Cet argument renvoie la valeur Vrai si le contrôle est coché(pour les checkBox) ou si la touche est enfoncée (pour les toggleButton). La valeur Faux est renvoyée dans le cas contraire.
L'argument **index**.
Exemple : Sub ___GetItemLabel(control As IRibbonControl, index As Integer, ByRef label)
Cet argument définit le numéro de l'élément dans les contrôles de type menu déroulant (ComboBox, dropDown , Gallery). 0= le premier élément dans la liste, 1= le deuxième élément ... etc ...
L'argument **Id**.
Exemple : Sub NomProcedure(control As IRibbonControl, Id As String, index As Integer)
Cet argument renvoie l'id de l'item sélectionné (pour les contrôles dropDown et Gallery).
L'argument **text**.
Exemple : Sub ___OnChange(control As IRibbonControl, text As String)
Renvoie la donnée contenue dans les contrôles de saisie (editBox, comboBox).
Remarques :
Bien entendu les CallBacks ne fonctionnent pas si vous désactivez les macros à l'ouverture du classeur.
Si l'onglet Développeur n'apparaît pas dans le ruban:

- Cliquez sur le bouton **Office**,
- puis sur le bouton **Options Excel**.
- Cliquez sur le menu **Standard**.
- Cochez l'option **Afficher l'onglet Développeur dans le ruban**.
- Cliquez sur le bouton **OK** pour valider.

Pour plus de détail sur les fonctions disponibles et pour obtenir d'autres exemples d'utilisation, consultez le tutoriel : **Personnalisation du ruban. Les fonctions d'appel CallBacks**.

V - L'actualisation des contrôles dans le ruban

Une partie des attributs de contrôles peut être paramétrée dynamiquement par VBA, comme nous l'avons vu dans le chapitre précédent.
En complément, il est aussi possible d'appliquer des actualisations à n'importe quel moment par macro, grâce à la fonction de rappel **onLoad**.
Ce rappel est déclenché une seule fois, après l'ouverture du classeur. Celui-ci va charger une variable objet, déclarée en type **IRibbonUI**, placée dans un module standard.
Par exemple : Public objRuban As IRibbonUI.
L'objet IRibbonUI possède deux méthodes :
Invalidate() qui actualise en une seule fois tous les contrôles personnalisés du classeur.
InvalidateControl(ControlID As String) qui actualise un contrôle particulier (ControlID correspond à l'identificateur unique du contrôle).
Cet exemple active un bouton de manière conditionnelle.
Le bouton contrôle est utilisable uniquement si le contenu de la cellule A1 contient la valeur 1.

Xml

```
<customUI xmlns="http://schemas.microsoft.com/office/2006/01/customui" onLoad="RubanCharge">
<!-- onLoad="RubanCharge" est déclenché lors du chargement du ruban personnalisé. -->
```

Xml
```
<ribbon startFromScratch="false">
<tabs>

<tab id="OngletPerso" label="OngletPerso" visible="true">
   <group id="Projet01" label="Projet 01">

      <!-- onAction="ProcLancement" définit la macro déclenchée lorsque vos cliquez sur le
 bouton. -->
      <!-- getEnabled="Bouton1_Enabled" gére la condition d'activation ou de désactivation. --
>
      <button id="Bouton1" label="Lancement" onAction="ProcLancement" size="normal"
         imageMso="Repeat" getEnabled="Bouton1_Enabled"/>

   </group>
</tab>

</tabs>
</ribbon>
</customUI>
```

Ouvrez le classeur Excel. Validez l'éventuel message d'erreur "Impossible d'exécuter la macro 'RubanCharge'...". Ce message est normal car le classeur ne contient pas encore les fonctions de rappel.
Placez ce code dans un module standard :

Vba
```
Option Explicit

Public objRuban As IRibbonUI
Public boolResult As Boolean


'Callback for customUI.onLoad
'Est déclenché lors du chargement du ruban personnalisé.
Sub RubanCharge(ribbon As IRibbonUI)
    boolResult = False
    Set objRuban = ribbon
End Sub


'Callback for Bouton1 onAction
'La procédure déclenchée lorsque vous cliquez sur le bouton.
Sub ProcLancement(control As IRibbonControl)
    MsgBox "ma procédure."
End Sub


'Callback for Bouton1 getEnabled
'Active ou désactive le bouton en fonction de la variable boolResult
Sub Bouton1_Enabled(control As IRibbonControl, ByRef returnedVal)
    returnedVal = boolResult
End Sub
```

Et placez ce code dans le module objet Worksheet de la feuille (où vous allez modifier le contenu de la cellule A1):

Vba
```
Option Explicit

'Evenement Change dans la feuille de calcul.
Private Sub Worksheet_Change(ByVal Target As Range)
    'Vérifie si la cellule A1 est modifiée et si la cellule contient la valeur 1.
    If Target.Address = "$A$1" And Target = 1 Then
        boolResult = True
    Else
        boolResult = False
    End If
```

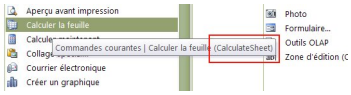
Vba

```
    'Rafraichit le bouton personnalisé
    If Not objRuban Is Nothing Then objRuban.InvalidateControl "Bouton1"
End Sub
```

Refermez puis ré-ouvrez le classeur (les macros doivent impérativement être activées). La variable "objRuban" va être chargée depuis la procédure "RubanCharge" : Set objRuban = ribbon
Si les conditions sont ensuite remplies (en fonction des modifications dans la cellule A1), le contrôle "Bouton1" va être activé ou désactivé.
Rappel:
Pour les contrôles de type comboBox, gallery et dynamicMenu, vous pouvez insérer l'attribut **invalidateContentOnDrop** dans le fichier xml de personnalisation afin d'actualiser automatiquement le contenu du contrôle lorsque celui-ci est sélectionné.

VI - Le chargement des images

La pluspart des contrôles (button, comboBox, dropDown, dynamicMenu, editBox, gallery, menu, splitButton, toggleButton) autorisent l'association d'image. Plusieurs méthodes sont disponibles si vous souhaitez ainsi agrémenter votre ruban personnalisé.
La solution la plus simple, car elle ne nécessite pas de gérer des fichiers externes, est d'utiliser l'attribut **imageMso**.
Ce paramètre permet de réaffecter les images des contrôles prédéfinis (équivalent des faceIds dans les versions précédentes d'Excel): imageMso="StartAfterPrevious".
Plusieurs exemples sont proposés dans le chapitre III-F.
Pour réutiliser les images des contrôles prédéfinis, vous devez connaître leur identificateur.
Cliquez sur le bouton "Office".
Cliquez sur le bouton "Options Excel".
Utilisez le menu "Personnaliser".
Le volet permet d'accéder à tous les contrôles (visibles et masqués).
Les identificateurs apparaissent dans une info-bulle lorsque vous passez le curseur sur les icônes.



Microsoft met à votre disposition un classeur qui présente l'intégralité de la galerie d'images et leur identificateur : **2007 Office System Add-In: Icons Gallery.**
Si les images de la galerie Office ne vous conviennent pas, vous avez aussi la possibilité de récupérer une image extérieure qui sera stockée dans le classeur. Vous appelez ensuite l'image en utilisant l'attribut **image**.
La syntaxe est image="NomFichierImage".
L'application est optimisée pour visualiser des fichiers types .png, mais vous pouvez également choisir des fichiers .jpg, .bmp, .gif, .ico...
Nota:
Attention, les fichiers png ne peuvent pas être chargés dynamiquement par VBA, via la fonction "LoadPicture".
L'utilitaire customUI Editor (voir le chapitre II-B) gère automatiquement l'ajout d'images personnelles par le bouton "Insert Icons".
Si vous modifier le ruban manuellement (comme dans le chapitre II-A), ajoutez un sous dossier nommé "images" dans le répertoire "customUI" et insérez y le fichier image.
Ensuite, ajoutez un sous dossier nommé "_rels" dans le répertoire "customUI". Créez un fichier nommé "customUI.xml.rels" qui contiendra un code de ce style (NomBouton.ico étant le fichier que vous avez précédemment glissé dans le sous dossier "images"):

Xml

```
<?xml version="1.0" encoding="utf-8"?>
<Relationships xmlns="http://schemas.openxmlformats.org/package/2006/relationships">
 <Relationship
  Type="http://schemas.openxmlformats.org/officeDocument/2006/relationships/image"
  Target="images/NomBouton.ico"
  Id="NomBouton" />
</Relationships>
```

Enfin, placez le fichier "customUI.xml.rels" dans le répertoire "_rels".
La fonction de rappel **getImage** permet de charger ou de modifier dynamiquement par VBA une image stockée sur le disque dur.

Xml

```
<customUI xmlns="http://schemas.microsoft.com/office/2006/01/customui">

  <ribbon>
    <tabs>
      <tab id="Tab01" label="Test" >
        <group id="Groupe01" label="Essai chargement image">
              <button id="button01"
                getImage="MacroChargementImage"
          tag="logo16.gif"/>
        </group>
      </tab>
    </tabs>
  </ribbon>

</customUI>
```

La fonction de chargement d'image, à placer dans un module standard.
Ici, le nom du fichier est stocké dans l'attribut "tag" du bouton.

Vba

```
'Callback for button01 getImage
Sub MacroChargementImage(control As IRibbonControl, ByRef returnedVal)
    Set returnedVal = LoadPicture("C:\dossier\" & control.Tag)
End Sub
```

Une autre option consiste à charger des images externes dynamiquement par VBA, depuis la fonction de rappel **loadImage**.
De cette manière, toutes les images sont récupérées en une seule fois, au moment ou vous activez un élément personnalisé de votre ruban.

Xml

```
<customUI xmlns="http://schemas.microsoft.com/office/2006/01/customui"
  loadImage="MacroChargementImage">

  <ribbon>
    <tabs>
      <tab id="Tab01" label="Test" >
        <group id="Groupe01" label="Essai chargement image">
              <button id="button01"
                image="logo16.gif"/>
        </group>
      </tab>
    </tabs>
  </ribbon>

</customUI>
```

Et la fonction de chargement d'image, à placer dans un module standard.
L'exemple présume qu'il existe une image nommée "logo16.gif" dans le répertoire "C:\dossier\" et que ce même nom est spécifié dans le fichier xml de personnalisation (image="logo16.gif").

Vba

```
'Callback for customUI.loadImage
Sub MacroChargementImage(imageID As String, ByRef returnedVal)
    Set returnedVal = LoadPicture("C:\dossier\" & imageID)
End Sub
```

Avec cette méthode, si le classeur doit être partagé, l'idéal est de stocker les fichiers images sur un répertoire en réseau.

VII - Conclusion

Cet article est une première présentation des attributs et des contrôles que vous pouvez utiliser dans vos projets. Vous disposez d'une panoplie de contrôles plus importante que dans les anciennes barres d'outils Excel. Dans certains cas, la personnalisation du ruban pourra même totalement se substituer aux classiques UserForm.
Le menu Office, la barre d'outils accès rapide (QAT), les menus contextuels sont personnalisables sur le même principe. Ces points feront l'objet d'un prochain article.

VIII - Liens

Création de rubans personnalisés sous Microsoft Access 2007 , par Christophe Warin.
Customizing the 2007 Office Fluent Ribbon for Developers
Custom UI Editor Tool
La FAQ Access
La FAQ Excel
Le XML dans Microsoft Office (OpenXML), par Olivier Lebeau.
2007 Office System Document: Lists of Control IDs

IX - Remerciements

Je remercie toute l'équipe Office de DVP pour la relecture et la correction du tutoriel.

X - Téléchargement