Utilisation de l'Interop Forms Toolkit enVFP9 - IV

Un vrai MultiThread en VFP

Dans ce 4ème chapitre, je vais vous montrer comment réaliser un vrai multithread avec VFP et l'Interop ToolKit. C'est une chose impossible en utilisant uniquement VFP, même si on y parvient avec COM et des timers.

La méthode de réalisation est issue du code que vous trouverez ici :
http://www.codeproject.com/KB/vb-interop/VB6InteropToolkit2.aspx
Je vous conseille vivement de lire cet article.

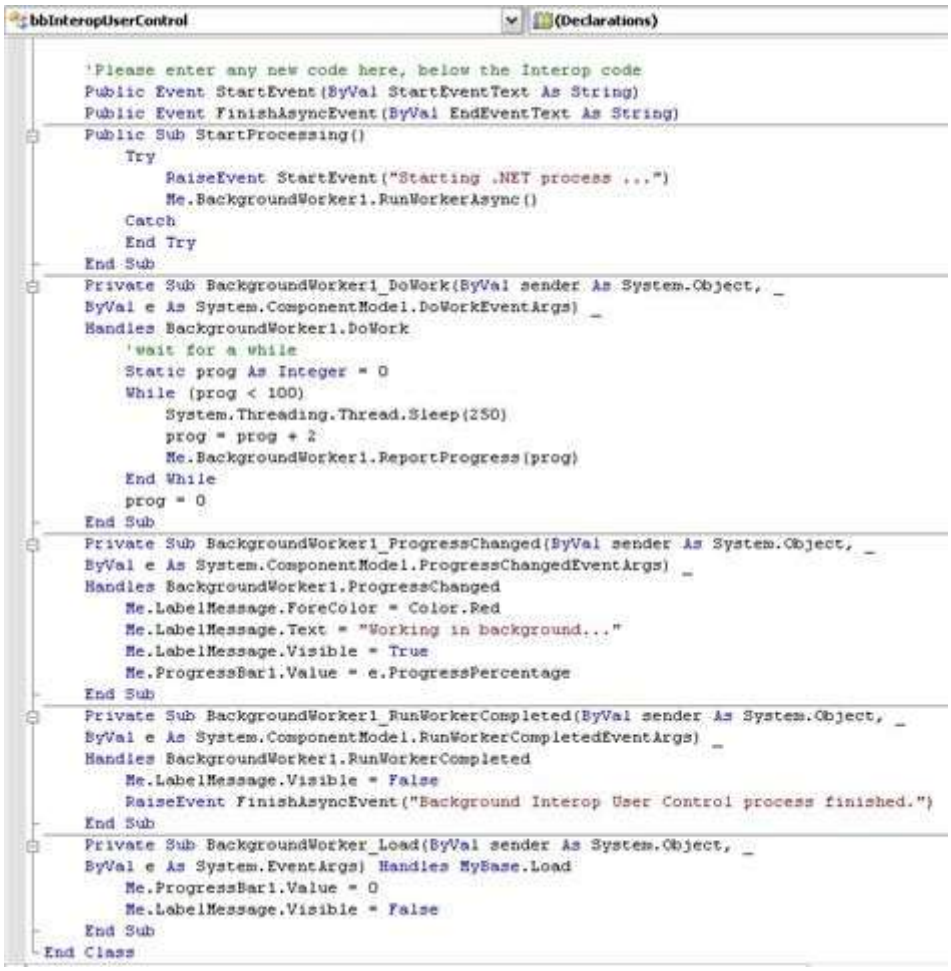
Démarrez VB 2005 Express, et créez un nouveau projet Interop User Control (plus de détails ici sur la façon de le faire). Appelez-le MultiThreadedControl. Renommez InteropUserControl.vb en bbInteropUserControl.vb

Ouvrez le contrôle en mode de conception, et ajoutez les 3 objets suivants sur le contrôle utilisateur :

Label → modifiez la propriété Name en LabelMessage
Progressbar
BackgroundWorker (que vous trouverez dans les composants)

Disposez-les pour que le contrôle ressemble à ceci :

Maintenant, double-cliquez sur le contrôle pour ouvrir la fenêtre de code, et saisissez ce qui suit, ou bien faites un copier-coller du code ci-dessous en vert.

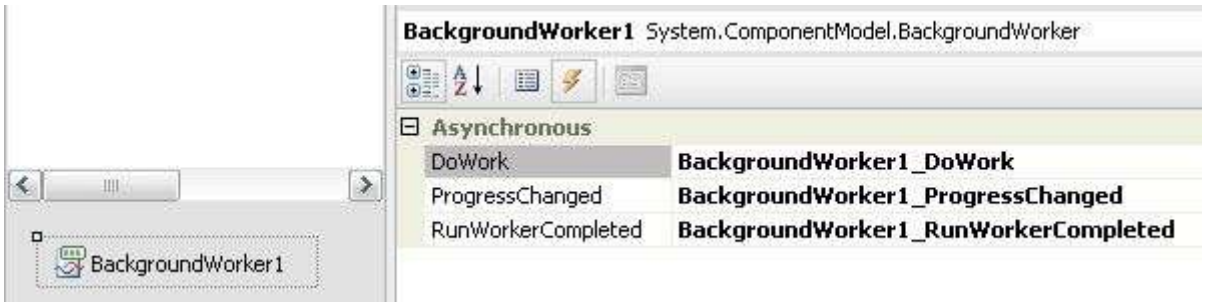



```
'Please enter any new code here, below the Interop code

    Public Event StartEvent(ByVal StartEventText As String)
    Public Event FinishAsyncEvent(ByVal EndEventText As String)

    Public Sub StartProcessing()
        Try
            ' Ceci est mis en commentaire, voir l'explication plus loin
            'RaiseEvent StartEvent("Starting .NET process ...")
            Me.BackgroundWorker1.RunWorkerAsync()
        Catch
        End Try
    End Sub

    Private Sub BackgroundWorker1_DoWork(ByVal sender As System.Object, _
    ByVal e As System.ComponentModel.DoWorkEventArgs) _
    Handles BackgroundWorker1.DoWork
        'wait for a while
        Static prog As Integer = 0
        ' Déplacé depuis le SUB ci-dessus. Voir explication ci-dessous.
        RaiseEvent StartEvent("Starting .NET process ...")
        While (prog < 100)
```

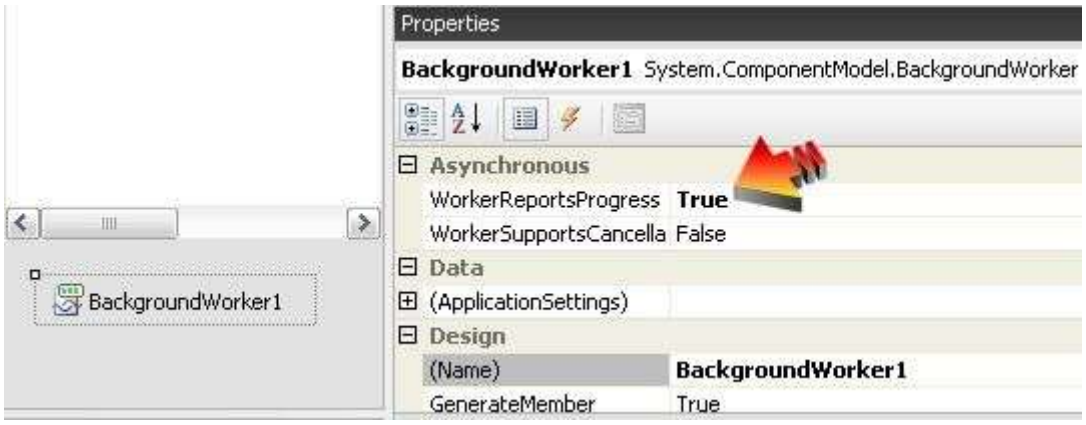
```
            System.Threading.Thread.Sleep(250)
            prog = prog + 2
            Me.BackgroundWorker1.ReportProgress(prog)
        End While
        prog = 0
    End Sub

    Private Sub BackgroundWorker1_ProgressChanged(ByVal sender As System.Object, _
    ByVal e As System.ComponentModel.ProgressChangedEventArgs) _
    Handles BackgroundWorker1.ProgressChanged
        Me.LabelMessage.ForeColor = Color.Red
        Me.LabelMessage.Text = "Working in background..."
        Me.LabelMessage.Visible = True
        Me.ProgressBar1.Value = e.ProgressPercentage
    End Sub

    Private Sub BackgroundWorker1_RunWorkerCompleted(ByVal sender As System.Object,
_
    ByVal e As System.ComponentModel.RunWorkerCompletedEventArgs) _
    Handles BackgroundWorker1.RunWorkerCompleted
        Me.LabelMessage.Visible = False
        RaiseEvent FinishAsyncEvent("Background Interop User Control process finished.")
    End Sub

    Private Sub BackgroundWorker_Load(ByVal sender As System.Object, _
    ByVal e As System.EventArgs) Handles MyBase.Load
        Me.ProgressBar1.Value = 0
        Me.LabelMessage.Visible = False
    End Sub

End Class
```

Vérifiez bien votre saisie.

Assurez-vous que les évènements de votre BackgroundWorker sont correctement définis :

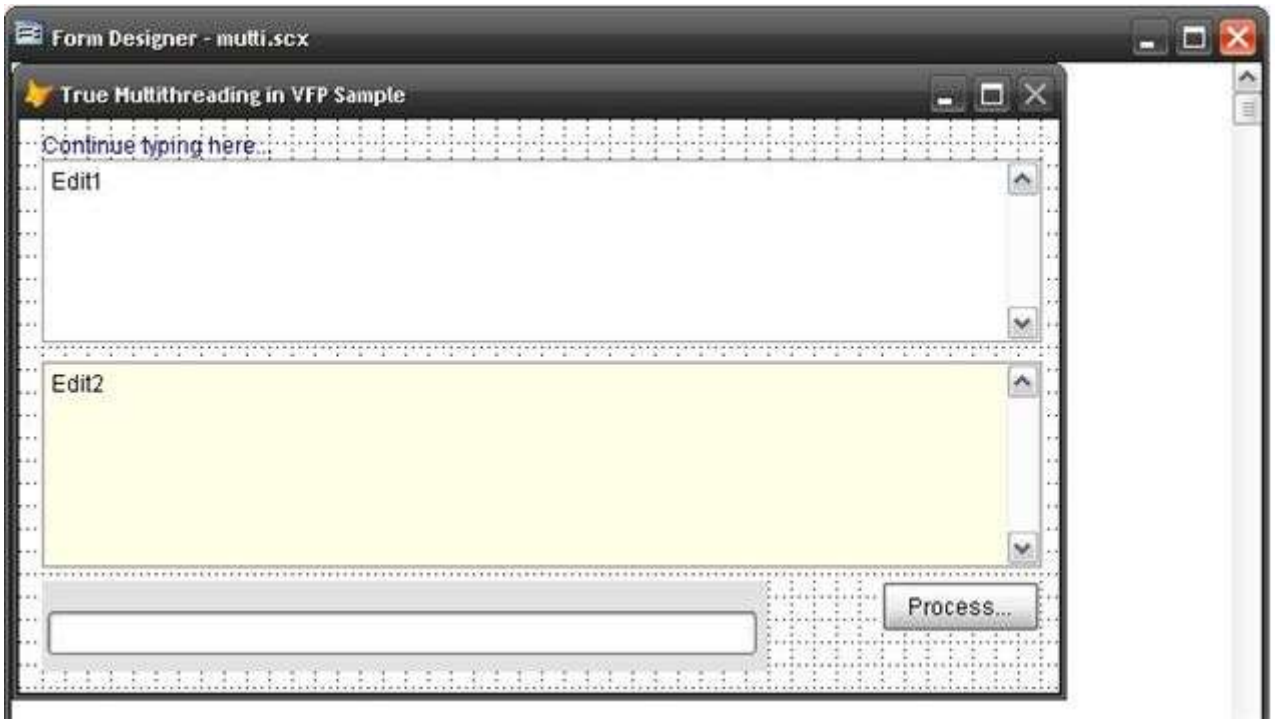


Vérifiez aussi que vous avez bien activé la progression comme ci-dessous :



Voilà pour la partie .Net. Générons la solution, ce qui va créer un ActiveXdll du nom de MultiThreadedControl.DLL qui sera vu comme un ActiveX dans VFP.

Passons maintenant à VFP et créons un nouveau Form. Ajoutons lui des objets dont nous réglons les propriétés comme suit :

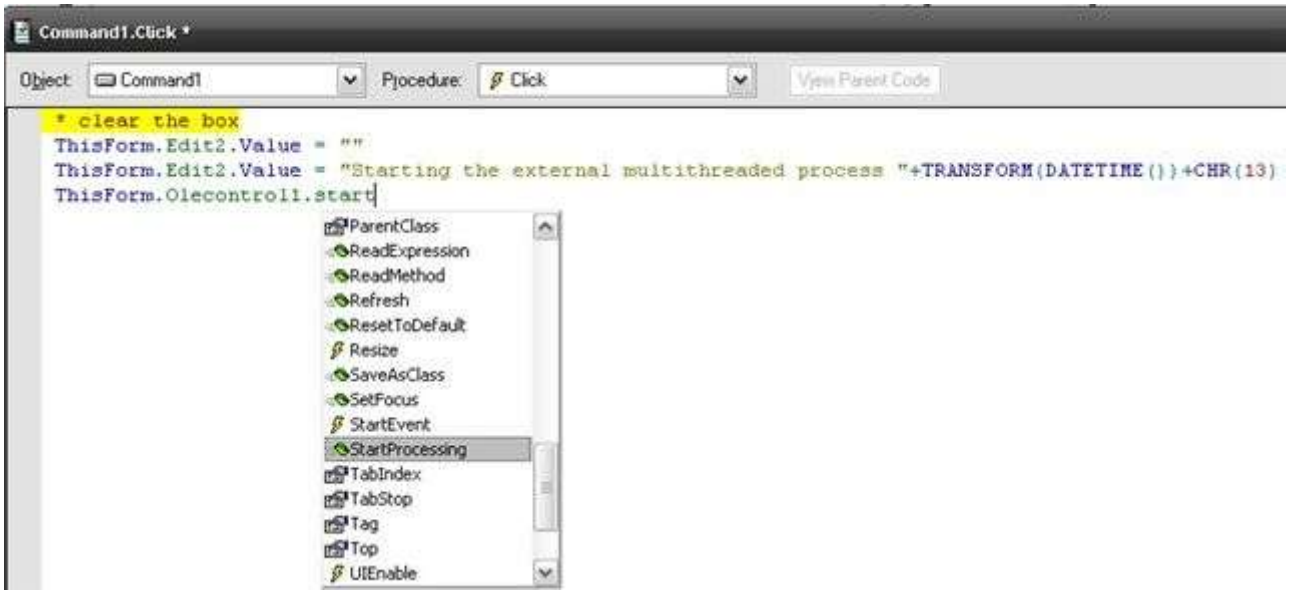


Edit2 est un editbox en ReadOnly dont le DisabledBackColor est jaune.
Nous avons ajouté notre Usercontrol en bas du form, comme on ajoute n'importe quel ActiveX.

Voyons maintenant le code, côté VFP.

Dans l'évènement Click du bouton, ajoutons ce code, qui tout d'abord efface le contenu de l'EditBox, affiche ensuite un message, et enfin démarre le process en arrière-plan. Au fur et à mesure de votre saisie, vous constatez que l'Intellisense fonctionne et affiche les Sub que nous avons exposés en .Net

```
Public Sub StartProcessing()
```



Raccordons-nous maintenant aux 2 évènements que nous avons exposés :

Public Event StartEvent(ByVal StartEventText As String)
Public Event FinishAsyncEvent(ByVal EndEventText As String)

Pour ce faire, double-cliquez sur le contrôle Interop, et choisissez StartEvent dans la liste. Vous constatez que VFP ajoute la déclaration du paramètre attendu. Ajoutez ce code pour annoncer le début de l'évènement.

Bizarrement, je n'ai pas réussi à déclencher cet évènement bien que le code de lancement du process fonctionne.

Correction ultérieure.
Manifestement, **on ne peut pas déclencher un évènement depuis une méthode exposée dans un COM .Net en appellant directement cette méthode. On ne peut déclencher cet évènement que par un déclencheur situé dans .Net**

Dans ce code:

```
Public Sub StartProcessing()
    Try
```

```
      RaiseEvent StartEvent("Starting .NET process ...")
      Me.BackgroundWorker1.RunWorkerAsync()
   Catch
   End Try
End Sub
```

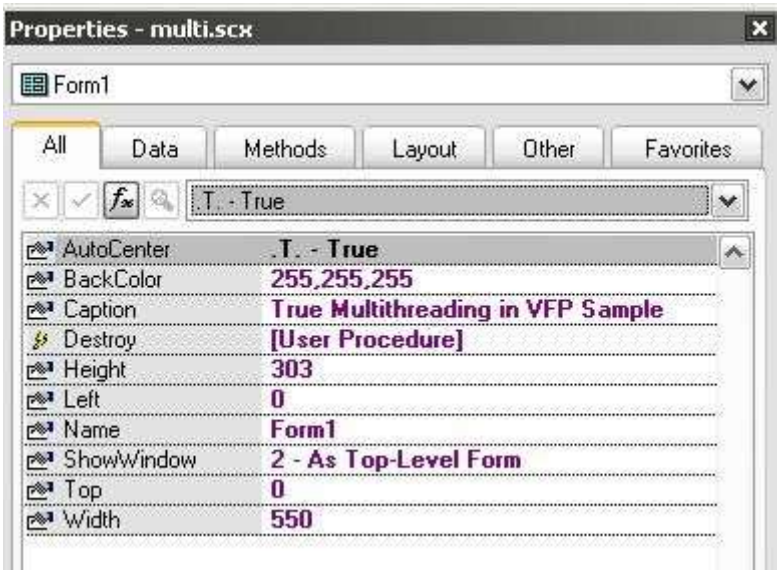
j'appelle le SUB exposé StartProcessing depuis mon bouton, et celui-ci à son tour déclenche le StartEvent. Ceci ne fonctionne pas dans VFP et .Net

Il nous faut donc déplacer l'évènement dans la méthode qui fait le travail. Son déclenchement est lancé depuis .Net, et on peut donc lancer cet évènement ici.

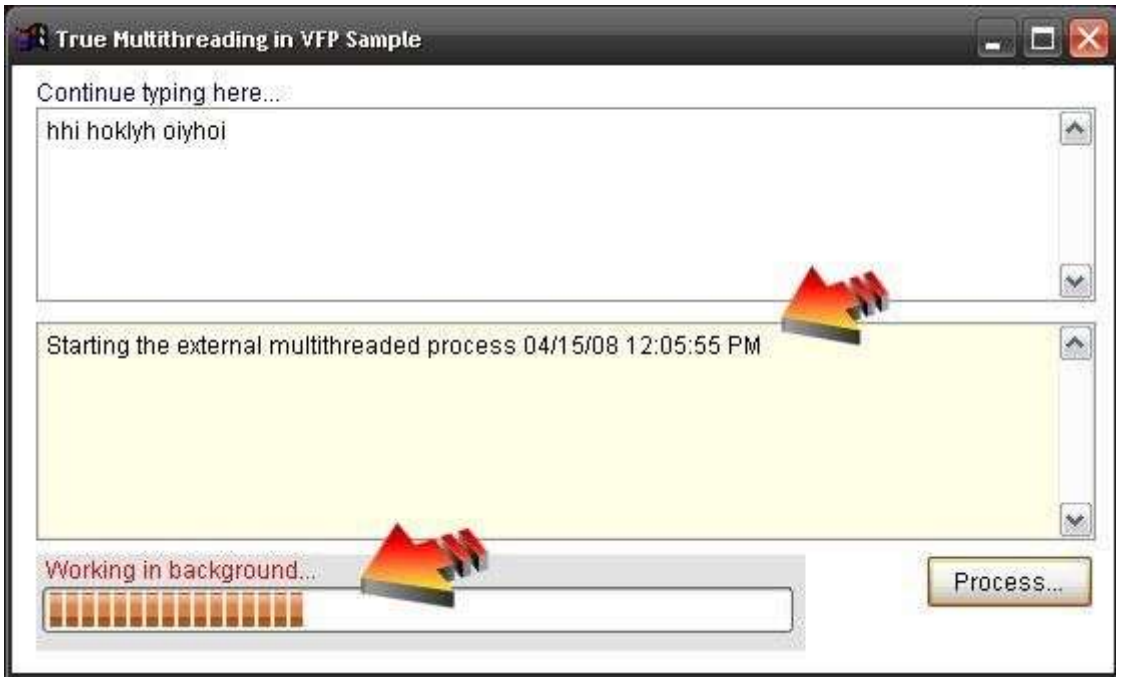
```
RaiseEvent StartEvent("Starting .NET process ...")
 While (prog < 100)
     System.Threading.Thread.Sleep(250)
     prog = prog + 2
     Me.BackgroundWorker1.ReportProgress(prog)
End While
```

Personne ne sait vraiment pourquoi ça marche comme ça, mais c'est ainsi ; allez savoir…

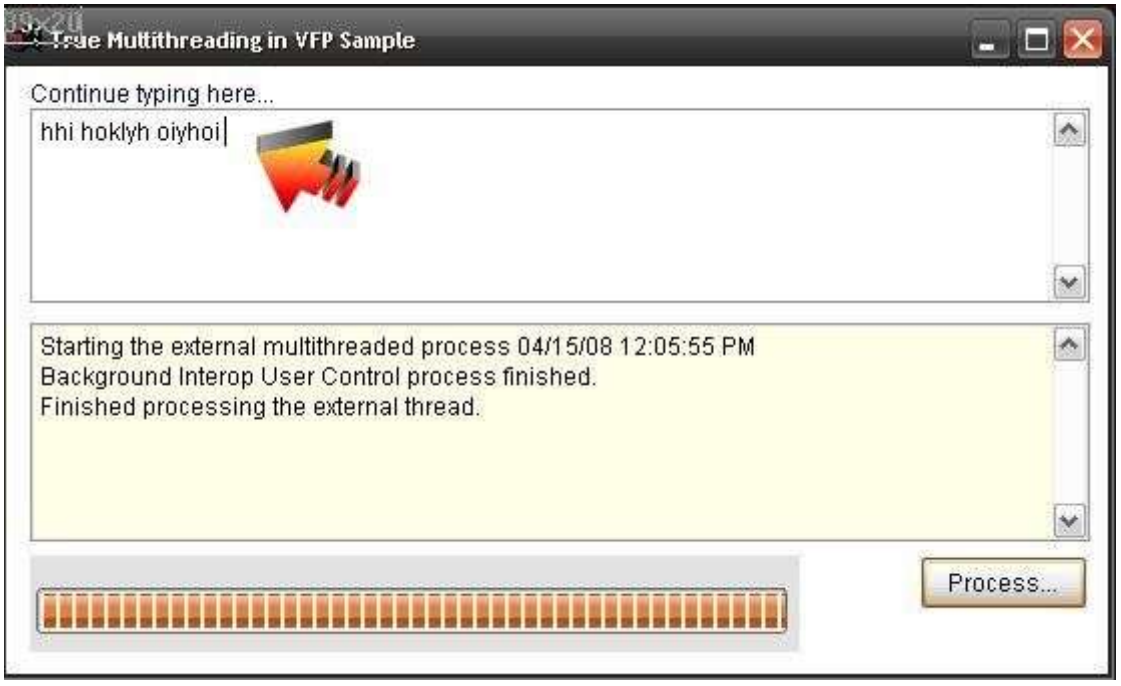
Ajoutez également le code pour gérer le **FinishAsyncEvent**.
Et pour terminer, ajustons ces quelques propriétés du Form :



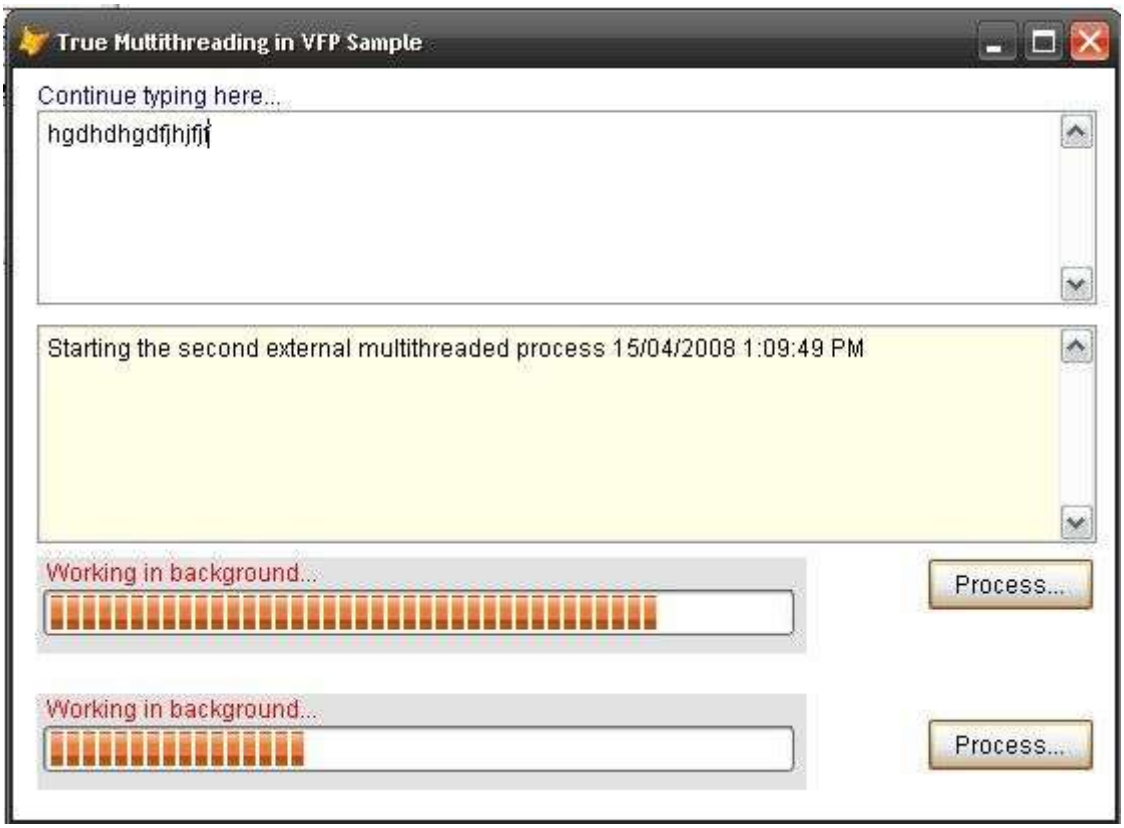
Tout est prêt pour le lancement, alors allons-y et lançons ce form. Saisissez n'importe quoi dans l'EditBox du haut, puis cliquez sur le bouton Process pour démarrer le process en arrière-plan dans un autre thread.



Vous allez constater que vous pouvez continuer de saisir dans l'EditBox ou faire autre chose, pendant que le process tourne. La barre de progression vous renseigne en retour sur son avancement. Une fois le process terminé, le déclenchement du **FinishAsyncEvent** vous en informe. La dernière partie de notre code est alors lancée pour afficher le message dans l'Editbox jaune.



Nous y sommes. **Un vrai MultiThred en VFP avec .Net**. mais pourquoi s'arrêter en si bon chemin ? ajoutons un autre composant MultiThread. Voici l'image d'un form (multi2.scx) sur lequel fonctionnent 2 process pendant que vous saisissez et que vous effectuer d'autres process.



Dans un prochain article, j'essaierai de vous montrer comment mettre plusieurs process dans différents threads, en utilisant COM et un objet .Net MultiThredFactory.

Tout le code source de ce qui précède est téléchargeable depuis le lien ci-dessous :

Première publication le 15 avril 2007 par bbout
Pièces jointes : VFPMultiThreaded.zip