Threading in Windows Forms Applications

تكمن المشكلة في أغراض Windows Forms هو أن التحكمات والنموذج ذات نفسه هو أنه يجب الوصول إليهم حصريا من خلال المسار الذي قام بإنشائهم وفي الحقيقة كل أغراض Windows Forms تعتمد على STA Model وذلك بسبب أنها جميعا معتمدة على هيكلية رسائل Win32 والتي ترث مسارات الغرفة Apartment-Threaded مما يعني أنه يمكنك إنشاء النموذج أو التحكم على أي مسار تريده ولكن جميع الطرق المرتبطة به يجب استدعاؤها من نفس المسار. مما يؤدي إلى ظهور العديد من المشاكل بسبب أن أقسام الدوت نيت الأخرى تستخدم Free-Threading model ومزج كلا النوعين بدون حكمة تعتبر فكرة سيئة وحتى لو لم تقم بإنشاء مسار بشكل واضح في كودك ربما ستظهر لك بعض المشاكل في جميع الأحوال فمثلا عندما تحاول الوصول إلى عنصر واجهة مستخدم UI Element من خلال الطريقة Finalize لنوع ما ونحن نعلم أن الطريقة Finalize يتم تنفيذها على مسار مختلف عن المسار الرئيسي

The ISynchronizeInvoke Interface

عناصر التحكم الوحيدة التي يمكنك استدعاؤهم من مسار آخر هم الذين يتم عرضهم من خلال الواجهة ISynchronizeInvoke التي تمتلك الطرائق BeginInvoke و EndInvoke و الخاصية InvokeRequired القابلة للقراءة فقط. حيث تعيد الخاصية InvokeRequired القيمة True إذا كان المستدعي لا يستطيع الوصول إلى التحكم مباشرة وذلك عندما يعمل المستدعي على مسار مختلف عن المسار الذي تم إنشاء التحكم فيه ففي هذه الحالة يجب على المستدعي استدعاء الطريقة Invoke للوصول إلى أي عنصر خاص بالتحكم وهذه الطريقة متزامنة لهذا يتم إيقاف المسار المستدعي حتى يكمل مسار UI تنفيذ الطريقة. أو يمكن للمسار المستدعي استخدام الطرائق BeginInvoke و EndInvoke لتنفيذ العملية بشكل لا متزامن.

تأخذ الطريقة Invoke إجراء مفوض يشير إلى طريقة (Sub أو Function) ويمكنه أخذ مصفوفة من النوع Object كبارامتر ثاني إذا كانت الطريقة تتوقع واحد أو أكثر من البارامترات وتضمن هيكلية نماذج ويندوز أن الإجراء الذي يشير إليه المفوض يتم تنفيذه في المسار UI لهذا يمكنه بأمان الوصول إلى أي تحكم على النموذج.

سنرى كيف يمكننا استخدام الطريقة Invoke للوصول إلى تحكم من مسار غير المسار UI حيث يظهر لنا المثال التالي كيف يمكننا زيارة جميع المجلدات ضمن شجرة مجلد من مسار ثانوي بينما يتم إظهار السم المجلد في تحكم Label وأول شئ سنقوم بعمله هو تحديد طريقة تقوم بعمل الإظهار المطلوب التي يمكنها أن تكون مجرد إجراء بسيط

```
' This method must run in the main UI thread.
Sub ShowMessage(ByVal msg As String)
  Me.lblMessage.Text = msg
  Me.Refresh()
End Sub
```

ثم نقوم بتحديد إجراء مفوض يشير لتلك الطريقة ومتغير يحمل كائن لذلك المفوض يكون معرفا على مستوى النموذج كي تتم مشاركته بين جميع الطرائق ضمن النموذج

```
' A delegate that can point to the ShowMessage procedure
Delegate Sub ShowMessageDelegate(ByVal msg As String)
' An instance of the delegate
Dim threadSafeDelegate As ShowMessageDelegate
```

وستحتاج لطريقة تبدأ المسار الثانوي مثلا إجراء معالجة الحدث Click لزر أوامر Button

```
' Parse the c:\Windows directory when the user clicks this button.
Private Sub btnSearch_Click(ByVal sender As Object, ByVal e As EventArgs) _
    Handles btnSearch.Click
  Dim t As New Thread(AddressOf SearchFiles)
  t.Start("c:\windows")
End Sub
```

وأخيرا تقوم بكتابة الكود الذي سيعمل على المسار الثانوي حيث أنه من الضروري لذلك الكود أن يستطيع الوصول للتحكم lblMessage باستدعاء الطريقة ShowMessage وهذا يتم من خلال الطريقة Invoke في فئة النموذج Form Class أو الطريقة Invoke لأي تحكم موجود على النموذج والتي تكون مكافئة لها تماما

```
' (This method runs in a non-UI thread.)
Sub SearchFiles(ByVal arg As Object)
```

```
  ' Retrieve the argument.
  Dim path As String = arg.ToString()
  ' Prepare the delegate
  threadSafeDelegate = New ShowMessageDelegate(AddressOf ShowMessage)
  ' Invoke the worker procedure. (The result isn't used in this demo.)
  Dim files As List(Of String) = GetFiles(path)
  ' Show that execution has terminated.
  Dim msg As String = String.Format("Found {0} files", files.Count)
  Me.Invoke(threadSafeDelegate, msg)
End Sub

' A recursive function that retrieves all the files in a directory tree
' (This method runs in a non-UI thread.)
Function GetFiles(ByVal path As String) As List(Of String)
  ' Display a message.
  Dim msg As String = String.Format("Parsing directory {0}", path)
  Me.Invoke(threadSafeDelegate, msg)
  ' Read the files in this folder and all subfolders.
  Dim files As New List(Of String)
  For Each fi As String In Directory.GetFiles(path)
    files.Add(fi)
  Next
  For Each di As String In Directory.GetDirectories(path)
    files.AddRange(GetFiles(di))
  Next
  Return files
End Function
```

وستتعقد العملية أكثر إن احتجنا لاستخدام الطريقة ShowMessage على جميع المسارات فالطريقة GetFiles مثلا يمكن استدعاؤها من المسار UI وفي هذه الحالة عمل الاستدعاء باستخدام الطريقة Invoke يضيف استباقا للأمور يجب تجنبه لذلك يجب علينا فحص قيمة الخاصية InvokeRequired واستخدام الطريقة العادية إن كانت تعيد القيمة False

```
' (Inside the SearchFiles and GetFiles methods)
If Me.InvokeRequired Then
  Me.Invoke(threadSafeDelegate, msg)
Else
  ShowMessage(msg)
End If
```

والطريقة الأفضل من ذلك بدلا من فحص الخاصية InvokeRequired من أجل كل مستدعي سنقوم بفحصها من داخل الطريقة ShowMessage

```
' This method can run in the UI thread or in a non-UI thread.
Sub ShowMessage(ByVal msg As String)
  ' Use the Invoke method only if necessary.

  If Me.InvokeRequired Then
    Me.Invoke(threadSafeDelegate, msg)
    Return
  End If

  Me.lblMessage.Text = msg
  Me.Refresh()
End Sub
```

فبعد هذا التغيير أي قطعة من الكود ستحتاج لإظهار رسالة على التحكم lblMessage ستحتاج فقط لاستدعاء ShowMessage بدون القلق حول أي مسار يتم تنفيذ الكود عليه
وفي بعض الظروف في فيجول بايزيك 2005 أو الفريموورك رقم 2 يقوم التطبيق بالوصول للتحكم عن طريق مسار غير مسار الإظهار non-UI thread بدون التسبب بأية مشاكل فيمكن حدوث ذلك مثلا عندما تحاول الوصول إلى تحكمات بسيطة مثل Label أو عندما تقوم بعمليات لا تسبب إرسال رسائل Win32 في الخلفية كما أن العديد من الخصائص يمكن قراءتها وليس تعديلها بدون التسبب بمشاكل وذلك لأن قيمة تلك الخصائص مخزنة في عنصر ضمن تحكم الدوت نيت

The BackgroundWorker Component

على الرغم من أن الواجهة ISynchronizeInvoke تجنبك من الوقوع في المشاكل المتعلقة بالمسارات في تطبيقات نماذج ويندوز يحتاج معظم مطوري فيجول بايزيك لطريقة أفضل وأقل أخطاء فأنت تحتاج مثلا لطريقة بسيطة لإلغاء طريقة غير متزامنة بأسلوب آمن الشئ الذي لا توفره الواجهة المذكورة بشكل تلقائي. ومن أجل هذا السبب قامت مايكروسوفت بإضافة المكون BackgroundWorker إلى صندوق الأدوات واستخدامه سهل جدا مما يسهل عملية إنشاء تطبيقات ويندوز متعددة المسارات.

يمتلك المكون BackgroundWorker خاصيتان مثيرتان للاهتمام فالخاصية WorkerReportsProgress تكون قيمتها True إذا أطلق المكون الحدث ProgressChanged والخاصية WorkerSupportsCancellation تكون قيمتها True إذا كان المكون يدعم الطريقة CancelAsync وتكون القيمة الافتراضية لكلا الطريقتين False لذا يجب عليك ضبط قيمتهم إلى True إذا أردت الاستفادة من جميع مزايا هذا التحكم والمثال الذي سيطرح هنا يفترض أنه قد تم ضبط كلتا القيمتين إلى True ويتطلب استخدام المكون BackgroundWorker بشكل عام العمليات التالية:

١. إنشاء إجراء معالجة للحدث DoWork وملؤها بالكود الذي تريد أن يتم تنفيذه على المسار الثانوي ويتم تشغيل هذا الكود عندما يتم استدعاء الطريقة RunWorkerAsync وهي تقبل بارامترا يتم تمريره لإجراء معالجة الحدث DoWork حيث لا يمكن للكود الموجود هناك الوصول مباشرة للتحكمات على النموذج لأنه يعمل في مسار آخر
٢. استخدم الطريقة ReportProgress من داخل الحدث DoWork عندما تريد الوصول إلى عنصر على النموذج وهذه الطريقة تطلق الحدث ProgressChanged إذا كانت قيمة الخاصية Worker-ReportsProgress هي True وإلا سيتم إطلاق استثناء Worker-ReportsProgress في حالة كون قيمتها False والكود في إجراء معالجة الحدث ProgressChanged يعمل في نفس المسار UI ولهذا يمكنه الوصول بأمان لأي من تحكمات النموذج
٣. استخدم الطريقة CancelAsync للتحكم BackgroundWorker لإيقاف المسار الثانوي مباشرة وهذه الطريقة تستدعي ضبط الخاصية WorkerSupportsCancellation إلى True وإلا سيتم إطلاق استثناء InvalidOperationException في حالة كون قيمتها False ويجب على الكود في DoWork التحقق دوريا من الخاصية CancellationPending والخروج بأمان عندما تصبح قيمتها True
٤. كتابة إجراء معالجة للحدث RunWorkerCompleted إن كنت تريد القيام بأية أعمال عندما ينتهي عمل المسار الثانوي إما بشكل طبيعي أو بواسطة الإلغاء والكود في إجراء معالجة هذا الحدث يعمل في المسار UI لذا يستطيع الوصول لجميع عناصر النموذج

وبشكل عام فالكود في معالج الحدث DoWork يجب أن يعيد قيمة للمسار الأساسي بدلا من تعيين هذه القيمة في حقل على مستوى الفئة فعلى الكود تعيين هذه القيمة للخاصية Result للغرض DoWorkEventArgs فتكون هذه القيمة متوفرة للمسار الأساسي بواسطة الخاصية Result للغرض RunWorkerCompletedEventArgs الممرر للحدث RunWorkerCompleted وهذا كود نموذجي يستخدم العنصر BackgroundWorker

```
' The button that starts the asynchronous operation
Private Sub btnStart_Click(ByVal sender As Object, ByVal e As EventArgs) _
    Handles btnStart.Click
  Dim argument As Object = "abcde"        ' The argument

  BackgroundWorker1.RunWorkerAsync(argument)
  ' Disable this button, and enable the "Stop" button.
  btnStart.Enabled = False
  btnStop.Enabled = True
End Sub
```

```
' The button that cancels the asynchronous operation
Private Sub btnStop_Click(ByVal sender As Object, ByVal e As EventArgs) _
    Handles btnStop.Click
  BackgroundWorker1.RunWorkerAsync(argument)
End Sub

' The code that performs the asynchronous operation
Private Sub BackgroundWorker1_DoWork(ByVal sender As Object, _
    ByVal e As DoWorkEventArgs) Handles BackgroundWorker1.DoWork
  ' Retrieve the argument.
  Dim argument As Object = e.Argument
  Dim percentage As Integer = 0
  …
  ' The core of the asynchronous task
  Do Until BackgroundWorker1.CancellationPending
    …
    ' Report progress when it makes sense to do so.
    BackgroundWorker1.ReportProgress(percentage)
  Loop
  ' Return the result to the caller.
  e.Result = primes
End Sub

' This method runs when the ReportProgress method is invoked.
Private Sub BackgroundWorker1_ProgressChanged(ByVal sender As Object, _
  ByVal e As ProgressChangedEventArgs) Handles BackgroundWorker1.ProgressChanged
  ' It is safe to access the user interface from here.
  ' For example, show the progress on a progress bar or another control.
  ToolStripProgressBar1.Value = e.ProgressPercentage
End Sub

' This method runs when the asynchronous task is completed (or canceled).
Private Sub BackgroundWorker1_RunWorkerCompleted(ByVal sender As Object, _
      ByVal e As RunWorkerCompletedEventArgs) Handles BackgroundWorker1.RunWorkerCompleted
  ' It is safe to access the user interface from here.
  …
  ' Reset the Enabled state of the Start and Stop buttons.
  btnStart.Enabled = True
  btnStop.Enabled = False
End Sub
```

يظهر لك المثال التالي كيف يمكن استخدام العنصر BackgroundWorker للبحث عن الملفات في مسار غير متزامن وهي نفس المشكلة التي طرحت عند الحديث عن The ISynchronizeInvoke Interface في هذا الموضوع سابقا وبهذا يمكنك مقارنة الطريقتين بسهولة. وستكون النسخة الجديدة المعتمدة على BackgroundWorker أكثر تعقيدا بقليل بسبب أنها تدعم الإلغاء لعمل غير متزامن

```
' The result from the SearchFiles procedure
Dim files As List(Of String)
' We need this variable to avoid nested calls to ProgressChanged.
Dim callInProgress As Boolean

' The same button works as a Start and a Stop button.
Private Sub btnStart_Click(ByVal sender As Object, ByVal e As EventArgs) _
    Handles btnStart.Click
  If btnStart.Text = "Start" Then
    lstFiles.Items.Clear()
```

```
    Me.BackgroundWorker1.RunWorkerAsync("c:\windows")
    Me.btnStart.Text = "Stop"
  Else
    Me.BackgroundWorker1.CancelAsync()
  End If
End Sub

' The code that starts the asynchronous file search
Private Sub BackgroundWorker1_DoWork(ByVal sender As Object, ByVal e As DoWorkEventArgs) _
    Handles BackgroundWorker1.DoWork
  ' Retrieve the argument.
  Dim path As String = e.Argument.ToString()
  ' Invoke the worker procedure.
  files = New List(Of String)
  SearchFiles(path)
  ' Return a result to the RunWorkerCompleted event.
  Dim msg As String = String.Format("Found {0} files", files.Count)
  e.Result = msg
End Sub

' A recursive function that retrieves all the files in a directory tree.
Sub SearchFiles(ByVal path As String)
  ' Display a message.

  Dim msg As String = String.Format("Parsing directory {0}", path)
  ' Notice that we don't really use the percentage;
  ' instead, we pass the message in the UserState property.
  Me.BackgroundWorker1.ReportProgress(0, msg)

  ' Read the files in this folder and all subfolders.
  ' Exit immediately if the task has been canceled.
  For Each fi As String In Directory.GetFiles(path)
    If Me.BackgroundWorker1.CancellationPending Then Return
    files.Add(fi)
  Next
  For Each di As String In Directory.GetDirectories(path)
     If Me.BackgroundWorker1.CancellationPending Then Return
     SearchFiles(di)
  Next
End Sub


Private Sub BackgroundWorker1_ProgressChanged(ByVal sender As Object, _
    ByVal e As ProgressChangedEventArgs) _
    Handles BackgroundWorker1.ProgressChanged
  ' Reject nested calls.
  If callInProgress Then Return
  callInProgress = True
  ' Display the message, received in the UserState property.
  Me.lblMessage.Text = e.UserState.ToString()
  ' Display all files added since last call.
  For i As Integer = lstFiles.Items.Count To files.Count - 1
    lstFiles.Items.Add(files(i))
  Next
  Me.Refresh()
  ' Let the Windows operating system process message in the queue.
```

```
  ' If you omit this call, clicks on buttons are ignored.
  Application.DoEvents()
  callInProgress = False
End Sub

Private Sub BackgroundWorker1_RunWorkerCompleted(ByVal sender As Object, _
    ByVal e As RunWorkerCompletedEventArgs) _
    Handles BackgroundWorker1.RunWorkerCompleted
  ' Display the last message and reset button's caption.
  Me.lblMessage.Text = e.Result.ToString()
  btnStart.Text = "Start"
End Sub
```

والكود هنا يشرح نفسه ماعدا إجراء الحدث ProgressChanged حيث يجب أن يتضمن الكود استدعاء للطريقة Application.DoEvents وإلا لن يتمكن التطبيق من معالجة الأحداث المنطلقة مثل حدث النقر على الزر Stop أو أي عمل آخر ممكن إضافته للواجهة ومع ذلك فاستدعاء هذه الطريقة سيسبب استدعاءات معششة للإجراء ProgressChanged مما قد يسبب إطلاق استثناء StackOverflowException ومن أجل عدم حدوث هذا يتم استخدام حقل منطقي مساعد callInProgress لتجنب حدوث مثل هذه الاستدعاءات المعششة

لاحظ أيضا أن هذا التطبيق لا يحتاج للإعلام عن نسبة التقدم للمسار الرئيسي ويستخدم الطريقة ReportProgress فقط لتنفيذ جزء من الكود في المسار الرئيسي للبرنامج والرسالة الفعلية للإظهار يتم تمريرها للخاصية UserState وإن كان تطبيقك يستخدم progress bar أو أي مؤشر آخر للتقدم يجب عليك تجنب استدعاء الطريقة ReportProgress بدون داعي لأنها تتسبب بتبديل المسارات وتكون مكلفة كثيرا عندما يتعلق الأمر بوقت المعالجة وفي هذه الحالة يجب عليك تخزين مؤشر التقدم في حقل في الفئة واستدعاء الطريقة فقط في حالة حدوث تقدم فعلي

```
Dim currentPercentage As Integer

Private Sub BackgroundWorker1_DoWork(ByVal sender As Object, ByVal e As DoWorkEventArgs) _
    Handles BackgroundWorker1.DoWork
  Const TotalSteps = 5000
  For i As Integer = 1 To TotalSteps
    ...
    ' Evaluate progress percentage.
    Dim percentage As Integer = (i * 100) \ TotalSteps
    ' Report to UI thread only if percentage has changed.
    If percentage <> currentPercentage Then
      BackgroundWorker1.ReportProgress(percentage)
      currentPercentage = percentage
    End If
  Next
End Sub
```

مترجم بتصرف

خاص بمنتدى الفريق العربي للبرمجة

محمد سامر أبو سلو