

WordB code snippets - January 2009 (written for Visual Basic 2008 Professional)

WORDB Codesnippets

(this one is not for the faint hearted...)

Asynchronous execution pattern used in our MSA program - non-commercial use only!

WordB code snippets - January 2009 (written for Visual Basic 2008 Professional)

Welcome to WordB code snippets. Here we will share some of the more useful code that is generated in the process of developing the MSA program.

This particular pattern is the result of needing to write quite a bit of async code - file move classes, background status checks, background network replication and so forth. It occurred to me that a nice reliable pattern that would take a class as an argument and simply run it async would be the answer.

The code that follows is the result of that logic. It works, works well, and has proven to be a comprehensive time saver - I can now implement an async class in a few minutes rather than re-writing blocks of code time and again.

I hope it helps you in whatever you are doing... or only just for interest sake.

However, if you want to use this work commercially, please contact me for a license.

Thanks & cheers,

Regards
Jan Jeltés
PO Box 611
Blackwood South Australia 5051
admin@ipresources.com

Asynchronous execution pattern used in our MSA program - non-commercial use only!

WordB code snippets - January 2009 (written for Visual Basic 2008 Professional)

POC Form - contains textbox1, label1, richtextbox1, button1. No other settings

if you have no parameters or retvals then you simply clear the parameter & return value classes and:

```
parameters.aWorkerClass = new clsDoAllTheWorkHere  
dim clsAsyn as New clsAsynPattern  
clsAsyn.start(Parameters)
```

That's all... and your class is run asynchronously for you!

```
''' <summary>
```

```
'''
```

```
''' Async delegate pattern uses advanced patterns to
```

```
''' create async functionality and optional
```

```
''' class-based invoke to update form control
```

```
'''
```

```
''' Not for the faint hearted...
```

```
'''
```

```
''' </summary>
```

```
''' <remarks></remarks>
```

```
Public Class Form1
```

```
Private Sub Button1_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles Button1.Click
```

```
Dim Parameters As New clsAsynParameters
```

```
'--Set aWorkerClass to any class you want to run asynchronously...
```

```
Parameters.aWorkerClass = New clsDoAllTheWorkHere
```

```
'--Initialise the list of calling parameters
```

```
Parameters.aListOfControls = New List(Of Control)
```

```
Parameters.aListOfControls.Add(Me.TextBox1)
```

```
Parameters.aListOfControls.Add(Me.Label1)
```

```
Parameters.aListOfControls.Add(Me.RichTextBox1)
```

```
Parameters.aDouble = 0
```

```
Parameters.aMessage = "This will appear in the status windows"
```

```
'--Run the aWorkerClass asynchronously !
```

```
Dim clsAsyn As New clsAsynPattern
```

```
clsAsyn.start(Parameters)
```

```
End Sub
```

```
End Class
```

Asynchronous execution pattern used in our MSA program - non-commercial use only!

Class which specifies calling parameters

Public Class clsAsynParameters

```
''' <summary>  
''' You probably don't want to remove the aWorkerClass object from here...  
''' </summary>
```

Public aWorkerClass As Object

```
''' <summary>  
''' But change below to whatever you require  
''' </summary>  
''' <remarks>(c) Jan Jeltens, 2009</remarks>  
Public aDouble As Double 'Thread sleep time  
Public aMessage As String 'return message to display in...  
Public aListOfControls As List(Of Control) = Nothing '...this list of controls (if any)
```

End Class

Class that specifies the return values

```
Public Class clsAsynReturnVals
```

```
    ''' <summary>  
    ''' Change this class to whatever instantiation / return values you require  
    ''' </summary>
```

```
    Public aString As String  
    Public aControlList As List(Of Control)
```

```
    Public Sub New(ByVal params As clsAsynParameters)  
        aString = params.aMessage  
        aControlList = params.aListOfControls  
    End Sub
```

```
End Class
```

This is the main pattern class. Do not change this...

```
Imports System
```

```
Imports System.Runtime.Remoting.Messaging
```

```
Public Class clsAsynPattern
```

```
''' <summary>  
''' This class is completely generic and centred around  
''' the clsAsynParameters (calling params) and clsAsynReturnVals  
''' (return values). It should therefore not require any changes.  
''' Rather, make the changes in the classes that are designed  
''' for that purpose...  
''' </summary>  
''' <remarks>  
''' This and related classes in this document are  
''' (c) Jan Jeldes, 2009, licensed for non-commercial use only  
''' Contact: admin@ipresources.com  
''' </remarks>
```

```
Private Delegate Function d_WorkDelegate(ByVal aValue As clsAsynParameters) As clsAsynReturnVals
```

```
Private _Params As clsAsynParameters
```

```
Private Property Parameters() As clsAsynParameters
```

```
Get
```

```
Return _Params
```

```
End Get
```

```
Set(ByVal value As clsAsynParameters)
```

```
_Params = value
```

```
End Set
```

```
End Property
```

```
Public Sub start(ByRef theParams As clsAsynParameters)
```

```
Parameters = theParams
```

```
If Not Parameters.aWorkerClass Is Nothing Then
```

```
Dim del As d_WorkDelegate = New d_WorkDelegate(AddressOf Worker)
```

```
Dim cb As AsyncCallback = New AsyncCallback(AddressOf Complete)
```

Asynchronous execution pattern used in our MSA program - non-commercial use only!

WordB code snippets - January 2009 (written for Visual Basic 2008 Professional)

```
    Dim ar As IAsyncResult = del.BeginInvoke(Parameters, cb, New Object)
Else
    Err.Raise(513, Nothing, "Working class missing!")
End If

End Sub

Private Function Worker(ByVal Params As clsAsyncParameters) As clsAsyncReturnVals

    Dim o_clsDoAllTheWorkHere As Object = Parameters.aWorkerClass
    Return (o_clsDoAllTheWorkHere.RunClassAsync(Parameters))

End Function

Private Function Complete(ByVal ar As IAsyncResult)

    Dim aresult As AsyncResult = CType(ar, AsyncResult)
    Dim temp As d_WorkDelegate = CType(aresult.AsyncDelegate, d_WorkDelegate)
    Dim ProcessReturnvalues As New clsDoReturnValueWork(temp.EndInvoke(ar))
    Return ProcessReturnvalues.Start

End Function

End Class
```

Asynchronous execution pattern used in our MSA program - non-commercial use only!

This is the worker class

```
Public Class clsDoAllTheWorkHere
```

```
    ''' <summary>  
    ''' Required function.  
    ''' </summary>
```

```
    Public Function RunClassAsync(ByVal params As clsAsynParameters) As clsAsynReturnVals
```

```
        'use this class pattern to do all the asyn work with clsAsynParameters and  
        ' return clsAsynReturnVals
```

```
        System.Threading.Thread.Sleep(params.aDouble)  
        Return (New clsAsynReturnVals(params))
```

```
    End Function
```

```
End Class
```

Use this class to perform any work that you want on the return values

```
Public Class clsDoReturnValueWork
```

```
    Private _result As clsAsynReturnVals
```

```
    ''' <summary>
```

```
    ''' Required function
```

```
    ''' </summary>
```

```
    Public Sub New(ByRef result As clsAsynReturnVals)
```

```
        _result = result
```

```
    End Sub
```

```
    ''' <summary>
```

```
    ''' Required function
```

```
    ''' </summary>
```

```
    ''' <returns></returns>
```

```
    Public Function Start() As Boolean
```

```
        'change anything you like within this function
```

```
        Try
```

```
            If Not _result.aControlList Is Nothing Then
```

```
                For Each ctl_control As Control In _result.aControlList
```

```
                    Dim cls_tb As New clsCrossThreadControlUpdater(ctl_control)
```

```
                    cls_tb.text = _result.aString
```

```
                Next
```

```
            End If
```

```
            Return True
```

```
        Catch ex As Exception
```

```
            Return False
```

```
        End Try
```

```
    End Function
```

```
End Class
```

Utility class that does cross-thread calls on "text-enabled" controls

```
Public Class clsCrossThreadControlUpdater
```

```
    ''' <summary>  
    ''' This is just a utility class to update text controls  
    ''' across threads  
    ''' </summary>
```

```
    Private destControlHasTextProperty As Control  
    Private Delegate Sub d_UpdateControlWithString(ByVal aString As String)
```

```
    Public Sub New(ByRef aControl As Control)
```

```
        If (TypeOf (aControl) Is TextBox) Or _  
            (TypeOf (aControl) Is RichTextBox) Or _  
            (TypeOf (aControl) Is Label) Then  
            destControlHasTextProperty = aControl  
        Else  
            destControlHasTextProperty = Nothing  
            'put other control logic here  
        End If
```

```
    End Sub
```

```
    Public Property text() As String
```

```
        Get  
            If Not destControlHasTextProperty Is Nothing Then  
                Return destControlHasTextProperty.Text  
            Else  
                'handle other controls here  
                Return Nothing  
            End If
```

```
        End Get
```

```
        Set(ByVal value As String)
```

```
            If Not destControlHasTextProperty Is Nothing Then  
                InvokeUpdateControlWithTextProperty(value)  
            Else
```

Asynchronous execution pattern used in our MSA program - non-commercial use only!

WordB code snippets - January 2009 (written for Visual Basic 2008 Professional)

```
'handle other controls here
End If
End Set
End Property

Private Sub InvokeUpdateControlWithTextProperty(ByVal value As String)
    destControlHasTextProperty.Invoke(New d_UpdateControlWithString(AddressOf HasTextPropertyUpdate), New String() {value})
End Sub

Private Sub HasTextPropertyUpdate(ByVal aString As String)

    Try
        destControlHasTextProperty.Text = aString
    Catch ex As Exception
        'Place your error handling routine here...
    End Try

End Sub

End Class
```

Asynchronous execution pattern used in our MSA program - non-commercial use only!