Fine-Tuning Your Async Application (Visual Basic)

Visual Studio 2015

You can add precision and flexibility to your async applications by using the methods and properties that the Task type makes available. The topics in this section show examples that use CancellationToken and important **Task** methods such as Task.WhenAll and Task.WhenAny.

By using **WhenAny** and **WhenAll**, you can more easily start multiple tasks and await their completion by monitoring a single task.

- **WhenAny** returns a task that completes when any task in a collection is complete.

  For examples that use **WhenAny**, see Cancel Remaining Async Tasks after One Is Complete (Visual Basic)and Start Multiple Async Tasks and Process Them As They Complete (Visual Basic).

- **WhenAll** returns a task that completes when all tasks in a collection are complete.

  For more information and an example that uses **WhenAll**, see How to: Extend the Async Walkthrough by Using Task.WhenAll (Visual Basic).

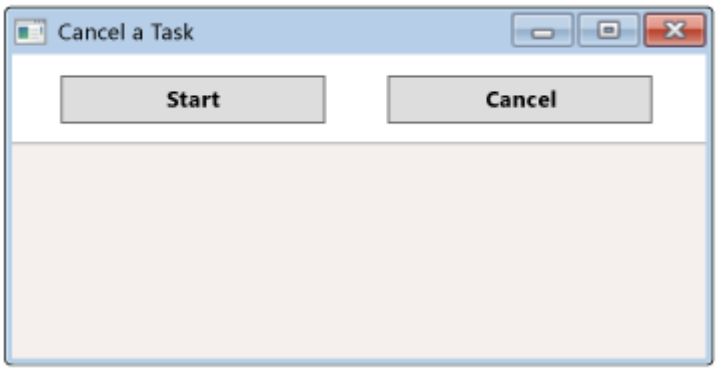
This section includes the following examples.

- Cancel an Async Task or a List of Tasks (Visual Basic).
- Cancel Async Tasks after a Period of Time (Visual Basic)
- Cancel Remaining Async Tasks after One Is Complete (Visual Basic)
- Start Multiple Async Tasks and Process Them As They Complete (Visual Basic)

Note

To run the examples, you must have Visual Studio 2012 or newer and the .NET Framework 4.5 or newer installed on your computer.

The projects create a UI that contains a button that starts the process and a button that cancels it, as the following image shows. The buttons are named `startButton` and `cancelButton`.



You can download the complete Windows Presentation Foundation (WPF) projects from Async Sample: Fine Tuning Your Application.

See Also

Asynchronous Programming with Async and Await (Visual Basic)

Cancel an Async Task or a List of Tasks (Visual Basic)

Visual Studio 2015

You can set up a button that you can use to cancel an async application if you don't want to wait for it to finish. By following the examples in this topic, you can add a cancellation button to an application that downloads the contents of one website or a list of websites.

The examples use the UI that Fine-Tuning Your Async Application (Visual Basic) describes.

> **Note**
>
> To run the examples, you must have Visual Studio 2012 or newer and the .NET Framework 4.5 or newer installed on your computer.

Cancel a Task

The first example associates the **Cancel** button with a single download task. If you choose the button while the application is downloading content, the download is canceled.

Downloading the Example

You can download the complete Windows Presentation Foundation (WPF) project from Async Sample: Fine Tuning Your Application and then follow these steps.

1. Decompress the file that you downloaded, and then start Visual Studio.
2. On the menu bar, choose **File**, **Open**, **Project/Solution**.
3. In the **Open Project** dialog box, open the folder that holds the sample code that you decompressed, and then open the solution (.sln) file for AsyncFineTuningVB.
4. In **Solution Explorer**, open the shortcut menu for the **CancelATask** project, and then choose **Set as StartUp Project**.
5. Choose the F5 key to run the project.

   Choose the Ctrl+F5 keys to run the project without debugging it.

If you don't want to download the project, you can review the MainWindow.xaml.vb files at the end of this topic.

Building the Example

The following changes add a **Cancel** button to an application that downloads a website. If you don't want to download or build the example, you can review the final product in the "Complete Examples" section at the end of this topic. Asterisks mark the changes in the code.

To build the example yourself, step by step, follow the instructions in the "Downloading the Example" section, but choose **StarterCode** as the **StartUp Project** instead of **CancelATask**.

Then add the following changes to the MainWindow.xaml.vb file of that project.

1. Declare a **CancellationTokenSource** variable, `cts`, that's in scope for all methods that access it.

VB

```
Class MainWindow

    ' ***Declare a System.Threading.CancellationTokenSource.
    Dim cts As CancellationTokenSource
```

2. Add the following event handler for the **Cancel** button. The event handler uses the CancellationTokenSource.Cancel method to notify `cts` when the user requests cancellation.

VB

```
' ***Add an event handler for the Cancel button.
Private Sub cancelButton_Click(sender As Object, e As RoutedEventArgs)

    If cts IsNot Nothing Then
        cts.Cancel()
    End If
End Sub
```

3. Make the following changes in the event handler for the **Start** button, `startButton_Click`.

   - Instantiate the **CancellationTokenSource**, `cts`.

VB

```
' ***Instantiate the CancellationTokenSource.
cts = New CancellationTokenSource()
```

   - In the call to `AccessTheWebAsync`, which downloads the contents of a specified website, send the CancellationTokenSource.Token property of `cts` as an argument. The **Token** property propagates the message if cancellation is requested. Add a catch block that displays a message if the user chooses to cancel the download operation. The following code shows the changes.

VB

```
    Try
        ' ***Send a token to carry the message if cancellation is requested.
        Dim contentLength As Integer = Await AccessTheWebAsync(cts.Token)

        resultsTextBox.Text &=
            String.Format(vbCrLf & "Length of the downloaded string: {0}." &
    vbCrLf, contentLength)

        ' *** If cancellation is requested, an OperationCanceledException
    results.
    Catch ex As OperationCanceledException
        resultsTextBox.Text &= vbCrLf & "Download canceled." & vbCrLf

    Catch ex As Exception
        resultsTextBox.Text &= vbCrLf & "Download failed." & vbCrLf
    End Try
```

4. In `AccessTheWebAsync`, use the HttpClient.GetAsync(String, CancellationToken) overload of the **GetAsync** method in the HttpClient type to download the contents of a website. Pass `ct`, the CancellationToken parameter of `AccessTheWebAsync`, as the second argument. The token carries the message if the user chooses the **Cancel** button.

   The following code shows the changes in `AccessTheWebAsync`.

   VB

```
' ***Provide a parameter for the CancellationToken.
Async Function AccessTheWebAsync(ct As CancellationToken) As Task(Of Integer)

    Dim client As HttpClient = New HttpClient()

    resultsTextBox.Text &=
        String.Format(vbCrLf & "Ready to download." & vbCrLf)

    ' You might need to slow things down to have a chance to cancel.
    Await Task.Delay(250)

    ' GetAsync returns a Task(Of HttpResponseMessage).
    ' ***The ct argument carries the message if the Cancel button is chosen.
    Dim response As HttpResponseMessage = Await
client.GetAsync("http://msdn.microsoft.com/en-us/library/dd470362.aspx", ct)

    ' Retrieve the website contents from the HttpResponseMessage.
    Dim urlContents As Byte() = Await response.Content.ReadAsByteArrayAsync()

    ' The result of the method is the length of the downloaded website.
    Return urlContents.Length
End Function
```

5. If you don't cancel the program, it produces the following output.

```
Ready to download.
Length of the downloaded string: 158125.
```

If you choose the **Cancel** button before the program finishes downloading the content, the program produces the following output.

```
Ready to download.
Download canceled.
```

Cancel a List of Tasks

You can extend the previous example to cancel many tasks by associating the same **CancellationTokenSource** instance with each task. If you choose the **Cancel** button, you cancel all tasks that aren't yet complete.

Downloading the Example

You can download the complete Windows Presentation Foundation (WPF) project from Async Sample: Fine Tuning Your Application and then follow these steps.

1. Decompress the file that you downloaded, and then start Visual Studio.
2. On the menu bar, choose **File**, **Open**, **Project/Solution**.
3. In the **Open Project** dialog box, open the folder that holds the sample code that you decompressed, and then open the solution (.sln) file for AsyncFineTuningVB.
4. In **Solution Explorer**, open the shortcut menu for the **CancelAListOfTasks** project, and then choose **Set as StartUp Project**.
5. Choose the F5 key to run the project.

   Choose the Ctrl+F5 keys to run the project without debugging it.

If you don't want to download the project, you can review the MainWindow.xaml.vb files at the end of this topic.

Building the Example

To extend the example yourself, step by step, follow the instructions in the "Downloading the Example" section, but choose **CancelATask** as the **StartUp Project**. Add the following changes to that project. Asterisks mark the changes in the program.

1. Add a method to create a list of web addresses.

VB

```
' ***Add a method that creates a list of web addresses.
Private Function SetUpURLList() As List(Of String)

    Dim urls = New List(Of String) From
        {
            "http://msdn.microsoft.com",
            "http://msdn.microsoft.com/en-us/library/hh290138.aspx",
            "http://msdn.microsoft.com/en-us/library/hh290140.aspx",
            "http://msdn.microsoft.com/en-us/library/dd470362.aspx",
            "http://msdn.microsoft.com/en-us/library/aa578028.aspx",
            "http://msdn.microsoft.com/en-us/library/ms404677.aspx",
            "http://msdn.microsoft.com/en-us/library/ff730837.aspx"
        }
    Return urls
End Function
```

2. Call the method in `AccessTheWebAsync`.

VB

```
' ***Call SetUpURLList to make a list of web addresses.
Dim urlList As List(Of String) = SetUpURLList()
```

3. Add the following loop in `AccessTheWebAsync` to process each web address in the list.

VB

```
' ***Add a loop to process the list of web addresses.
For Each url In urlList
    ' GetAsync returns a Task(Of HttpResponseMessage).
    ' Argument ct carries the message if the Cancel button is chosen.
    ' ***Note that the Cancel button can cancel all remaining downloads.
    Dim response As HttpResponseMessage = Await client.GetAsync(url, ct)

    ' Retrieve the website contents from the HttpResponseMessage.
    Dim urlContents As Byte() = Await response.Content.ReadAsByteArrayAsync()

    resultsTextBox.Text &=
        String.Format(vbCrLf & "Length of the downloaded string: {0}." & vbCrLf,
urlContents.Length)
Next
```

4. Because `AccessTheWebAsync` displays the lengths, the method doesn't need to return anything. Remove the return statement, and change the return type of the method to Task instead of Task(Of TResult).

VB

```
Async Function AccessTheWebAsync(ct As CancellationToken) As Task
```

Call the method from `startButton_Click` by using a statement instead of an expression.

VB

```
Await AccessTheWebAsync(cts.Token)
```

5. If you don't cancel the program, it produces the following output.

```
Length of the downloaded string: 35939.

Length of the downloaded string: 237682.

Length of the downloaded string: 128607.

Length of the downloaded string: 158124.

Length of the downloaded string: 204890.

Length of the downloaded string: 175488.

Length of the downloaded string: 145790.

Downloads complete.
```

If you choose the **Cancel** button before the downloads are complete, the output contains the lengths of the downloads that completed before the cancellation.

```
Length of the downloaded string: 35939.

Length of the downloaded string: 237682.

Length of the downloaded string: 128607.

Downloads canceled.
```

Complete Examples

The following sections contain the code for each of the previous examples. Notice that you must add a reference for System.Net.Http.

You can download the projects from Async Sample: Fine Tuning Your Application.

Cancel a Task Example

The following code is the complete MainWindow.xaml.vb file for the example that cancels a single task.

VB

```
' Add an Imports directive and a reference for System.Net.Http.
Imports System.Net.Http

' Add the following Imports directive for System.Threading.
Imports System.Threading

Class MainWindow

    ' ***Declare a System.Threading.CancellationTokenSource.
    Dim cts As CancellationTokenSource

    Private Async Sub startButton_Click(sender As Object, e As RoutedEventArgs)
        ' ***Instantiate the CancellationTokenSource.
        cts = New CancellationTokenSource()

        resultsTextBox.Clear()

        Try
            ' ***Send a token to carry the message if cancellation is requested.
            Dim contentLength As Integer = Await AccessTheWebAsync(cts.Token)

            resultsTextBox.Text &=
                String.Format(vbCrLf & "Length of the downloaded string: {0}." &
vbCrLf, contentLength)

            ' *** If cancellation is requested, an OperationCanceledException results.
        Catch ex As OperationCanceledException
            resultsTextBox.Text &= vbCrLf & "Download canceled." & vbCrLf

        Catch ex As Exception
            resultsTextBox.Text &= vbCrLf & "Download failed." & vbCrLf
        End Try

        ' ***Set the CancellationTokenSource to Nothing when the download is complete.
        cts = Nothing
    End Sub

    ' ***Add an event handler for the Cancel button.
    Private Sub cancelButton_Click(sender As Object, e As RoutedEventArgs)

        If cts IsNot Nothing Then
            cts.Cancel()
        End If
    End Sub

    ' ***Provide a parameter for the CancellationToken.
    Async Function AccessTheWebAsync(ct As CancellationToken) As Task(Of Integer)
```

```
        Dim client As HttpClient = New HttpClient()

        resultsTextBox.Text &=
            String.Format(vbCrLf & "Ready to download." & vbCrLf)

        ' You might need to slow things down to have a chance to cancel.
        Await Task.Delay(250)

        ' GetAsync returns a Task(Of HttpResponseMessage).
        ' ***The ct argument carries the message if the Cancel button is chosen.
        Dim response As HttpResponseMessage = Await
client.GetAsync("http://msdn.microsoft.com/en-us/library/dd470362.aspx", ct)

        ' Retrieve the website contents from the HttpResponseMessage.
        Dim urlContents As Byte() = Await response.Content.ReadAsByteArrayAsync()

        ' The result of the method is the length of the downloaded website.
        Return urlContents.Length
    End Function
End Class


' Output for a successful download:

' Ready to download.

' Length of the downloaded string: 158125.


' Or, if you cancel:

' Ready to download.

' Download canceled.
```

Cancel a List of Tasks Example

The following code is the complete MainWindow.xaml.vb file for the example that cancels a list of tasks.

VB

```
' Add an Imports directive and a reference for System.Net.Http.
Imports System.Net.Http

' Add the following Imports directive for System.Threading.
Imports System.Threading

Class MainWindow

    ' Declare a System.Threading.CancellationTokenSource.
```

```
Dim cts As CancellationTokenSource


Private Async Sub startButton_Click(sender As Object, e As RoutedEventArgs)

    ' Instantiate the CancellationTokenSource.
    cts = New CancellationTokenSource()

    resultsTextBox.Clear()

    Try
        ' ***AccessTheWebAsync returns a Task, not a Task(Of Integer).
        Await AccessTheWebAsync(cts.Token)
        '  ***Small change in the display lines.
        resultsTextBox.Text &= vbCrLf & "Downloads complete."

    Catch ex As OperationCanceledException
        resultsTextBox.Text &= vbCrLf & "Downloads canceled." & vbCrLf

    Catch ex As Exception
        resultsTextBox.Text &= vbCrLf & "Downloads failed." & vbCrLf
    End Try

    ' Set the CancellationTokenSource to Nothing when the download is complete.
    cts = Nothing
End Sub


' Add an event handler for the Cancel button.
Private Sub cancelButton_Click(sender As Object, e As RoutedEventArgs)

    If cts IsNot Nothing Then
        cts.Cancel()
    End If
End Sub


' Provide a parameter for the CancellationToken.
' ***Change the return type to Task because the method has no return statement.
Async Function AccessTheWebAsync(ct As CancellationToken) As Task

    Dim client As HttpClient = New HttpClient()

    ' ***Call SetUpURLList to make a list of web addresses.
    Dim urlList As List(Of String) = SetUpURLList()

    ' ***Add a loop to process the list of web addresses.
    For Each url In urlList
        ' GetAsync returns a Task(Of HttpResponseMessage).
        ' Argument ct carries the message if the Cancel button is chosen.
        ' ***Note that the Cancel button can cancel all remaining downloads.
        Dim response As HttpResponseMessage = Await client.GetAsync(url, ct)

        ' Retrieve the website contents from the HttpResponseMessage.
```

```
            Dim urlContents As Byte() = Await response.Content.ReadAsByteArrayAsync()

            resultsTextBox.Text &=
                String.Format(vbCrLf & "Length of the downloaded string: {0}." &
vbCrLf, urlContents.Length)
        Next
    End Function


    ' ***Add a method that creates a list of web addresses.
    Private Function SetUpURLList() As List(Of String)

        Dim urls = New List(Of String) From
            {
                "http://msdn.microsoft.com",
                "http://msdn.microsoft.com/en-us/library/hh290138.aspx",
                "http://msdn.microsoft.com/en-us/library/hh290140.aspx",
                "http://msdn.microsoft.com/en-us/library/dd470362.aspx",
                "http://msdn.microsoft.com/en-us/library/aa578028.aspx",
                "http://msdn.microsoft.com/en-us/library/ms404677.aspx",
                "http://msdn.microsoft.com/en-us/library/ff730837.aspx"
            }
        Return urls
    End Function

End Class


' Output if you do not choose to cancel:

' Length of the downloaded string: 35939.

' Length of the downloaded string: 237682.

' Length of the downloaded string: 128607.

' Length of the downloaded string: 158124.

' Length of the downloaded string: 204890.

' Length of the downloaded string: 175488.

' Length of the downloaded string: 145790.

' Downloads complete.


'  Sample output if you choose to cancel:

' Length of the downloaded string: 35939.

' Length of the downloaded string: 237682.

' Length of the downloaded string: 128607.
```

```
    ' Downloads canceled.
```

See Also

CancellationTokenSource
CancellationToken
Asynchronous Programming with Async and Await (Visual Basic)
Fine-Tuning Your Async Application (Visual Basic)
Async Sample: Fine Tuning Your Application

Cancel Async Tasks after a Period of Time (Visual Basic)

Visual Studio 2015

You can cancel an asynchronous operation after a period of time by using the CancellationTokenSource.CancelAfter method if you don't want to wait for the operation to finish. This method schedules the cancellation of any associated tasks that aren't complete within the period of time that's designated by the **CancelAfter** expression.

This example adds to the code that's developed in Cancel an Async Task or a List of Tasks (Visual Basic) to download a list of websites and to display the length of the contents of each one.

> **Note**
>
> To run the examples, you must have Visual Studio 2012 or later and the .NET Framework 4.5 or later installed on your computer.

Downloading the Example

You can download the complete Windows Presentation Foundation (WPF) project from Async Sample: Fine Tuning Your Application and then follow these steps.

1. Decompress the file that you downloaded, and then start Visual Studio.
2. On the menu bar, choose **File**, **Open**, **Project/Solution**.
3. In the **Open Project** dialog box, open the folder that holds the sample code that you decompressed, and then open the solution (.sln) file for AsyncFineTuningVB.
4. In **Solution Explorer**, open the shortcut menu for the **CancelAfterTime** project, and then choose **Set as StartUp Project**.
5. Choose the F5 key to run the project.

   Choose the Ctrl+F5 keys to run the project without debugging it.
6. Run the program several times to verify that the output might show output for all websites, no websites, or some web sites.

If you don't want to download the project, you can review the MainWindow.xaml.vb file at the end of this topic.

Building the Example

The example in this topic adds to the project that's developed in Cancel an Async Task or a List of Tasks (Visual Basic) to cancel a list of tasks. The example uses the same UI, although the **Cancel** button isn't used explicitly.

To build the example yourself, step by step, follow the instructions in the "Downloading the Example" section, but choose **CancelAListOfTasks** as the **StartUp Project**. Add the changes in this topic to that project.

To specify a maximum time before the tasks are marked as canceled, add a call to **CancelAfter** to `startButton_Click`, as the following example shows. The addition is marked with asterisks.

VB

```
Private Async Sub startButton_Click(sender As Object, e As RoutedEventArgs)

    ' Instantiate the CancellationTokenSource.
    cts = New CancellationTokenSource()

    resultsTextBox.Clear()

    Try
        ' ***Set up the CancellationTokenSource to cancel after 2.5 seconds. (You
        ' can adjust the time.)
        cts.CancelAfter(2500)

        Await AccessTheWebAsync(cts.Token)
        resultsTextBox.Text &= vbCrLf & "Downloads complete."

    Catch ex As OperationCanceledException
        resultsTextBox.Text &= vbCrLf & "Downloads canceled." & vbCrLf

    Catch ex As Exception
        resultsTextBox.Text &= vbCrLf & "Downloads failed." & vbCrLf
    End Try

    ' Set the CancellationTokenSource to Nothing when the download is complete.
    cts = Nothing
End Sub
```

Run the program several times to verify that the output might show output for all websites, no websites, or some web sites. The following output is a sample.

```
Length of the downloaded string: 35990.

Length of the downloaded string: 407399.

Length of the downloaded string: 226091.

Downloads canceled.
```

Complete Example

The following code is the complete text of the MainWindow.xaml.vb file for the example. Asterisks mark the elements that were added for this example.

Notice that you must add a reference for System.Net.Http.

You can download the project from Async Sample: Fine Tuning Your Application.

VB

```
' Add an Imports directive and a reference for System.Net.Http.
Imports System.Net.Http

' Add the following Imports directive for System.Threading.
Imports System.Threading

Class MainWindow

    ' Declare a System.Threading.CancellationTokenSource.
    Dim cts As CancellationTokenSource


    Private Async Sub startButton_Click(sender As Object, e As RoutedEventArgs)

        ' Instantiate the CancellationTokenSource.
        cts = New CancellationTokenSource()

        resultsTextBox.Clear()

        Try
            ' ***Set up the CancellationTokenSource to cancel after 2.5 seconds. (You
            ' can adjust the time.)
            cts.CancelAfter(2500)

            Await AccessTheWebAsync(cts.Token)
            resultsTextBox.Text &= vbCrLf & "Downloads complete."

        Catch ex As OperationCanceledException
            resultsTextBox.Text &= vbCrLf & "Downloads canceled." & vbCrLf

        Catch ex As Exception
            resultsTextBox.Text &= vbCrLf & "Downloads failed." & vbCrLf
        End Try

        ' Set the CancellationTokenSource to Nothing when the download is complete.
        cts = Nothing
    End Sub


    ' You can still include a Cancel button if you want to.
    Private Sub cancelButton_Click(sender As Object, e As RoutedEventArgs)
```

```
        If cts IsNot Nothing Then
            cts.Cancel()
        End If
    End Sub


    ' Provide a parameter for the CancellationToken.
    ' Change the return type to Task because the method has no return statement.
    Async Function AccessTheWebAsync(ct As CancellationToken) As Task

        Dim client As HttpClient = New HttpClient()

        ' Call SetUpURLList to make a list of web addresses.
        Dim urlList As List(Of String) = SetUpURLList()

        ' Process each element in the list of web addresses.
        For Each url In urlList
            ' GetAsync returns a Task(Of HttpResponseMessage).
            ' Argument ct carries the message if the Cancel button is chosen.
            ' Note that the Cancel button can cancel all remaining downloads.
            Dim response As HttpResponseMessage = Await client.GetAsync(url, ct)

            ' Retrieve the website contents from the HttpResponseMessage.
            Dim urlContents As Byte() = Await response.Content.ReadAsByteArrayAsync()

            resultsTextBox.Text &=
                String.Format(vbCrLf & "Length of the downloaded string: {0}." &
vbCrLf, urlContents.Length)
        Next
    End Function


    ' Add a method that creates a list of web addresses.
    Private Function SetUpURLList() As List(Of String)

        Dim urls = New List(Of String) From
            {
                "http://msdn.microsoft.com",
                "http://msdn.microsoft.com/en-us/library/hh290138.aspx",
                "http://msdn.microsoft.com/en-us/library/hh290140.aspx",
                "http://msdn.microsoft.com/en-us/library/dd470362.aspx",
                "http://msdn.microsoft.com/en-us/library/aa578028.aspx",
                "http://msdn.microsoft.com/en-us/library/ms404677.aspx",
                "http://msdn.microsoft.com/en-us/library/ff730837.aspx"
            }
        Return urls
    End Function

End Class


' Sample output:

' Length of the downloaded string: 35990.
```

```
' Length of the downloaded string: 407399.

' Length of the downloaded string: 226091.

' Downloads canceled.
```

See Also

Asynchronous Programming with Async and Await (Visual Basic)
Walkthrough: Accessing the Web by Using Async and Await (Visual Basic)
Cancel an Async Task or a List of Tasks (Visual Basic)
Fine-Tuning Your Async Application (Visual Basic)
Async Sample: Fine Tuning Your Application

Cancel Remaining Async Tasks after One Is Complete (Visual Basic)

Visual Studio 2015

By using the Task.WhenAny method together with a CancellationToken, you can cancel all remaining tasks when one task is complete. The **WhenAny** method takes an argument that's a collection of tasks. The method starts all the tasks and returns a single task. The single task is complete when any task in the collection is complete.

This example demonstrates how to use a cancellation token in conjunction with **WhenAny** to hold onto the first task to finish from the collection of tasks and to cancel the remaining tasks. Each task downloads the contents of a website. The example displays the length of the contents of the first download to complete and cancels the other downloads.

> **Note**
>
> To run the examples, you must have Visual Studio 2012 or newer and the .NET Framework 4.5 or newer installed on your computer.

Downloading the Example

You can download the complete Windows Presentation Foundation (WPF) project from Async Sample: Fine Tuning Your Application and then follow these steps.

1. Decompress the file that you downloaded, and then start Visual Studio.
2. On the menu bar, choose **File**, **Open**, **Project/Solution**.
3. In the **Open Project** dialog box, open the folder that holds the sample code that you decompressed, and then open the solution (.sln) file for AsyncFineTuningVB.
4. In **Solution Explorer**, open the shortcut menu for the **CancelAfterOneTask** project, and then choose **Set as StartUp Project**.
5. Choose the F5 key to run the project.

   Choose the Ctrl+F5 keys to run the project without debugging it.
6. Run the program several times to verify that different downloads finish first.

If you don't want to download the project, you can review the MainWindow.xaml.vb file at the end of this topic.

Building the Example

The example in this topic adds to the project that's developed in Cancel an Async Task or a List of Tasks (C# and Visual Basic) to cancel a list of tasks. The example uses the same UI, although the **Cancel** button isn't used explicitly.

To build the example yourself, step by step, follow the instructions in the "Downloading the Example" section, but choose **CancelAListOfTasks** as the **StartUp Project**. Add the changes in this topic to that project.

In the MainWindow.xaml.vb file of the **CancelAListOfTasks** project, start the transition by moving the processing steps for each website from the loop in `AccessTheWebAsync` to the following async method.

VB

```
' ***Bundle the processing steps for a website into one async method.
Async Function ProcessURLAsync(url As String, client As HttpClient, ct As
CancellationToken) As Task(Of Integer)

    ' GetAsync returns a Task(Of HttpResponseMessage).
    Dim response As HttpResponseMessage = Await client.GetAsync(url, ct)

    ' Retrieve the website contents from the HttpResponseMessage.
    Dim urlContents As Byte() = Await response.Content.ReadAsByteArrayAsync()

    Return urlContents.Length
End Function
```

In `AccessTheWebAsync`, this example uses a query, the ToArray(Of TSource) method, and the **WhenAny** method to create and start an array of tasks. The application of **WhenAny** to the array returns a single task that, when awaited, evaluates to the first task to reach completion in the array of tasks.

Make the following changes in `AccessTheWebAsync`. Asterisks mark the changes in the code file.

1. Comment out or delete the loop.

2. Create a query that, when executed, produces a collection of generic tasks. Each call to `ProcessURLAsync` returns a Task(Of TResult) where **TResult** is an integer.

   VB

   ```
   ' ***Create a query that, when executed, returns a collection of tasks.
   Dim downloadTasksQuery As IEnumerable(Of Task(Of Integer)) =
       From url In urlList Select ProcessURLAsync(url, client, ct)
   ```

3. Call **ToArray** to execute the query and start the tasks. The application of the **WhenAny** method in the next step would execute the query and start the tasks without using **ToArray**, but other methods might not. The safest practice is to force execution of the query explicitly.

   VB

   ```
   ' ***Use ToArray to execute the query and start the download tasks.
   ```

```
Dim downloadTasks As Task(Of Integer)() = downloadTasksQuery.ToArray()
```

4. Call **WhenAny** on the collection of tasks. **WhenAny** returns a **Task(Of Task(Of Integer))** or **Task<Task<int>>**. That is, **WhenAny** returns a task that evaluates to a single **Task(Of Integer)** or **Task<int>** when it's awaited. That single task is the first task in the collection to finish. The task that finished first is assigned to `firstFinishedTask`. The type of `firstFinishedTask` is Task(Of TResult) where **TResult** is an integer because that's the return type of `ProcessURLAsync`.

VB

```
' ***Call WhenAny and then await the result. The task that finishes
' first is assigned to firstFinishedTask.
Dim firstFinishedTask As Task(Of Integer) = Await Task.WhenAny(downloadTasks)
```

5. In this example, you're interested only in the task that finishes first. Therefore, use CancellationTokenSource.Cancel to cancel the remaining tasks.

VB

```
' ***Cancel the rest of the downloads. You just want the first one.
cts.Cancel()
```

6. Finally, await `firstFinishedTask` to retrieve the length of the downloaded content.

VB

```
Dim length = Await firstFinishedTask
resultsTextBox.Text &= String.Format(vbCrLf & "Length of the downloaded website:
{0}" & vbCrLf, length)
```

Run the program several times to verify that different downloads finish first.

Complete Example

The following code is the complete MainWindow.xaml.vb or MainWindow.xaml.cs file for the example. Asterisks mark the elements that were added for this example.

Notice that you must add a reference for System.Net.Http.

You can download the project from Async Sample: Fine Tuning Your Application.

VB

```
' Add an Imports directive and a reference for System.Net.Http.
Imports System.Net.Http

' Add the following Imports directive for System.Threading.
Imports System.Threading
```

```
Class MainWindow

    ' Declare a System.Threading.CancellationTokenSource.
    Dim cts As CancellationTokenSource


    Private Async Sub startButton_Click(sender As Object, e As RoutedEventArgs)

        ' Instantiate the CancellationTokenSource.
        cts = New CancellationTokenSource()

        resultsTextBox.Clear()

        Try
            Await AccessTheWebAsync(cts.Token)
            resultsTextBox.Text &= vbCrLf & "Download complete."

        Catch ex As OperationCanceledException
            resultsTextBox.Text &= vbCrLf & "Download canceled." & vbCrLf

        Catch ex As Exception
            resultsTextBox.Text &= vbCrLf & "Download failed." & vbCrLf
        End Try

        ' Set the CancellationTokenSource to Nothing when the download is complete.
        cts = Nothing
    End Sub


    ' You can still include a Cancel button if you want to.
    Private Sub cancelButton_Click(sender As Object, e As RoutedEventArgs)

        If cts IsNot Nothing Then
            cts.Cancel()
        End If
    End Sub


    ' Provide a parameter for the CancellationToken.
    ' Change the return type to Task because the method has no return statement.
    Async Function AccessTheWebAsync(ct As CancellationToken) As Task

        Dim client As HttpClient = New HttpClient()

        ' Call SetUpURLList to make a list of web addresses.
        Dim urlList As List(Of String) = SetUpURLList()

        '' Comment out or delete the loop.
        ''For Each url In urlList
        ''    ' GetAsync returns a Task(Of HttpResponseMessage).
        ''    ' Argument ct carries the message if the Cancel button is chosen.
        ''    ' Note that the Cancel button can cancel all remaining downloads.
        ''    Dim response As HttpResponseMessage = Await client.GetAsync(url, ct)
```

```
        ''    ' Retrieve the website contents from the HttpResponseMessage.
        ''    Dim urlContents As Byte() = Await response.Content.ReadAsByteArrayAsync()

        ''    resultsTextBox.Text &=
        ''        String.Format(vbCrLf & "Length of the downloaded string: {0}." &
vbCrLf, urlContents.Length)
        ''Next

        ' ***Create a query that, when executed, returns a collection of tasks.
        Dim downloadTasksQuery As IEnumerable(Of Task(Of Integer)) =
            From url In urlList Select ProcessURLAsync(url, client, ct)

        ' ***Use ToArray to execute the query and start the download tasks.
        Dim downloadTasks As Task(Of Integer)() = downloadTasksQuery.ToArray()

        ' ***Call WhenAny and then await the result. The task that finishes
        ' first is assigned to firstFinishedTask.
        Dim firstFinishedTask As Task(Of Integer) = Await Task.WhenAny(downloadTasks)

        ' ***Cancel the rest of the downloads. You just want the first one.
        cts.Cancel()

        ' ***Await the first completed task and display the results
        ' Run the program several times to demonstrate that different
        ' websites can finish first.
        Dim length = Await firstFinishedTask
        resultsTextBox.Text &= String.Format(vbCrLf & "Length of the downloaded
website:  {0}" & vbCrLf, length)
    End Function


    ' ***Bundle the processing steps for a website into one async method.
    Async Function ProcessURLAsync(url As String, client As HttpClient, ct As
CancellationToken) As Task(Of Integer)

        ' GetAsync returns a Task(Of HttpResponseMessage).
        Dim response As HttpResponseMessage = Await client.GetAsync(url, ct)

        ' Retrieve the website contents from the HttpResponseMessage.
        Dim urlContents As Byte() = Await response.Content.ReadAsByteArrayAsync()

        Return urlContents.Length
    End Function


    ' Add a method that creates a list of web addresses.
    Private Function SetUpURLList() As List(Of String)

        Dim urls = New List(Of String) From
            {
                "http://msdn.microsoft.com",
                "http://msdn.microsoft.com/en-us/library/hh290138.aspx",
                "http://msdn.microsoft.com/en-us/library/hh290140.aspx",
```

```
                "http://msdn.microsoft.com/en-us/library/dd470362.aspx",
                "http://msdn.microsoft.com/en-us/library/aa578028.aspx",
                "http://msdn.microsoft.com/en-us/library/ms404677.aspx",
                "http://msdn.microsoft.com/en-us/library/ff730837.aspx"
            }
        Return urls
    End Function

End Class


' Sample output:

' Length of the downloaded website:  158856

' Download complete.
```

See Also

WhenAny
Fine-Tuning Your Async Application (Visual Basic)
Asynchronous Programming with Async and Await (Visual Basic)
Async Sample: Fine Tuning Your Application

Start Multiple Async Tasks and Process Them As They Complete (Visual Basic)

Visual Studio 2015

By using Task.WhenAny, you can start multiple tasks at the same time and process them one by one as they're completed rather than process them in the order in which they're started.

The following example uses a query to create a collection of tasks. Each task downloads the contents of a specified website. In each iteration of a while loop, an awaited call to **WhenAny** returns the task in the collection of tasks that finishes its download first. That task is removed from the collection and processed. The loop repeats until the collection contains no more tasks.

> **Note**
>
> To run the examples, you must have Visual Studio 2012 or newer and the .NET Framework 4.5 or newer installed on your computer.

Downloading the Example

You can download the complete Windows Presentation Foundation (WPF) project from Async Sample: Fine Tuning Your Application and then follow these steps.

1. Decompress the file that you downloaded, and then start Visual Studio.
2. On the menu bar, choose **File**, **Open**, **Project/Solution**.
3. In the **Open Project** dialog box, open the folder that holds the sample code that you decompressed, and then open the solution (.sln) file for AsyncFineTuningVB.
4. In **Solution Explorer**, open the shortcut menu for the **ProcessTasksAsTheyFinish** project, and then choose **Set as StartUp Project**.
5. Choose the F5 key to run the project.

   Choose the Ctrl+F5 keys to run the project without debugging it.
6. Run the project several times to verify that the downloaded lengths don't always appear in the same order.

If you don't want to download the project, you can review the MainWindow.xaml.vb file at the end of this topic.

Building the Example

This example adds to the code that's developed in Cancel Remaining Async Tasks after One Is Complete (Visual Basic) and uses the same UI.

To build the example yourself, step by step, follow the instructions in the "Downloading the Example" section, but choose **CancelAfterOneTask** as the **StartUp Project**. Add the changes in this topic to the `AccessTheWebAsync` method in that project. The changes are marked with asterisks.

The **CancelAfterOneTask** project already includes a query that, when executed, creates a collection of tasks. Each call to `ProcessURLAsync` in the following code returns a Task(Of TResult) where **TResult** is an integer.

VB

```
Dim downloadTasksQuery As IEnumerable(Of Task(Of Integer)) =
    From url In urlList Select ProcessURLAsync(url, client, ct)
```

In the MainWindow.xaml.vb file of the project, make the following changes to the `AccessTheWebAsync` method.

- Execute the query by applying Enumerable.ToList(Of TSource) instead of ToArray(Of TSource).

  VB

  ```
  Dim downloadTasks As List(Of Task(Of Integer)) = downloadTasksQuery.ToList()
  ```

- Add a while loop that performs the following steps for each task in the collection.

  1. Awaits a call to **WhenAny** to identify the first task in the collection to finish its download.

     VB

     ```
     Dim firstFinishedTask As Task(Of Integer) = Await Task.WhenAny(downloadTasks)
     ```

  2. Removes that task from the collection.

     VB

     ```
     downloadTasks.Remove(firstFinishedTask)
     ```

  3. Awaits `firstFinishedTask`, which is returned by a call to `ProcessURLAsync`. The `firstFinishedTask` variable is a Task(Of TResult) where **TReturn** is an integer. The task is already complete, but you await it to retrieve the length of the downloaded website, as the following example shows.

     VB

     ```
     Dim length = Await firstFinishedTask
     resultsTextBox.Text &= String.Format(vbCrLf & "Length of the downloaded
     website:  {0}" & vbCrLf, length)
     ```

You should run the project several times to verify that the downloaded lengths don't always appear in the same order.

> **Caution**
>
> You can use **WhenAny** in a loop, as described in the example, to solve problems that involve a small number of tasks. However, other approaches are more efficient if you have a large number of tasks to process. For more information and examples, see Processing Tasks as they complete.

Complete Example

The following code is the complete text of the MainWindow.xaml.vb file for the example. Asterisks mark the elements that were added for this example.

Notice that you must add a reference for System.Net.Http.

You can download the project from Async Sample: Fine Tuning Your Application.

VB

```
' Add an Imports directive and a reference for System.Net.Http.
Imports System.Net.Http

' Add the following Imports directive for System.Threading.
Imports System.Threading

Class MainWindow

    ' Declare a System.Threading.CancellationTokenSource.
    Dim cts As CancellationTokenSource


    Private Async Sub startButton_Click(sender As Object, e As RoutedEventArgs)

        ' Instantiate the CancellationTokenSource.
        cts = New CancellationTokenSource()

        resultsTextBox.Clear()

        Try
            Await AccessTheWebAsync(cts.Token)
            resultsTextBox.Text &= vbCrLf & "Downloads complete."

        Catch ex As OperationCanceledException
            resultsTextBox.Text &= vbCrLf & "Downloads canceled." & vbCrLf

        Catch ex As Exception
            resultsTextBox.Text &= vbCrLf & "Downloads failed." & vbCrLf
        End Try
```

```
        ' Set the CancellationTokenSource to Nothing when the download is complete.
        cts = Nothing
    End Sub


    ' You can still include a Cancel button if you want to.
    Private Sub cancelButton_Click(sender As Object, e As RoutedEventArgs)

        If cts IsNot Nothing Then
            cts.Cancel()
        End If
    End Sub


    ' Provide a parameter for the CancellationToken.
    ' Change the return type to Task because the method has no return statement.
    Async Function AccessTheWebAsync(ct As CancellationToken) As Task

        Dim client As HttpClient = New HttpClient()

        ' Call SetUpURLList to make a list of web addresses.
        Dim urlList As List(Of String) = SetUpURLList()

        ' ***Create a query that, when executed, returns a collection of tasks.
        Dim downloadTasksQuery As IEnumerable(Of Task(Of Integer)) =
            From url In urlList Select ProcessURLAsync(url, client, ct)

        ' ***Use ToList to execute the query and start the download tasks.
        Dim downloadTasks As List(Of Task(Of Integer)) = downloadTasksQuery.ToList()

        ' ***Add a loop to process the tasks one at a time until none remain.
        While downloadTasks.Count > 0
            ' ***Identify the first task that completes.
            Dim firstFinishedTask As Task(Of Integer) = Await
Task.WhenAny(downloadTasks)

            ' ***Remove the selected task from the list so that you don't
            ' process it more than once.
            downloadTasks.Remove(firstFinishedTask)

            ' ***Await the first completed task and display the results.
            Dim length = Await firstFinishedTask
            resultsTextBox.Text &= String.Format(vbCrLf & "Length of the downloaded
website:  {0}" & vbCrLf, length)
        End While

    End Function


    ' Bundle the processing steps for a website into one async method.
    Async Function ProcessURLAsync(url As String, client As HttpClient, ct As
CancellationToken) As Task(Of Integer)

        ' GetAsync returns a Task(Of HttpResponseMessage).
```

```
        Dim response As HttpResponseMessage = Await client.GetAsync(url, ct)

        ' Retrieve the website contents from the HttpResponseMessage.
        Dim urlContents As Byte() = Await response.Content.ReadAsByteArrayAsync()

        Return urlContents.Length
    End Function


    ' Add a method that creates a list of web addresses.
    Private Function SetUpURLList() As List(Of String)

        Dim urls = New List(Of String) From
            {
                "http://msdn.microsoft.com",
                "http://msdn.microsoft.com/en-us/library/hh290138.aspx",
                "http://msdn.microsoft.com/en-us/library/hh290140.aspx",
                "http://msdn.microsoft.com/en-us/library/dd470362.aspx",
                "http://msdn.microsoft.com/en-us/library/aa578028.aspx",
                "http://msdn.microsoft.com/en-us/library/ms404677.aspx",
                "http://msdn.microsoft.com/en-us/library/ff730837.aspx"
            }
        Return urls
    End Function

End Class


' Sample output:

' Length of the download:  226093
' Length of the download:  412588
' Length of the download:  175490
' Length of the download:  204890
' Length of the download:  158855
' Length of the download:  145790
' Length of the download:  44908
' Downloads complete.
```

See Also

WhenAny(Of TResult)
Fine-Tuning Your Async Application (Visual Basic)
Asynchronous Programming with Async and Await (Visual Basic)
Async Sample: Fine Tuning Your Application