

**Module 7:
Asynchronous
Programming**

Overview

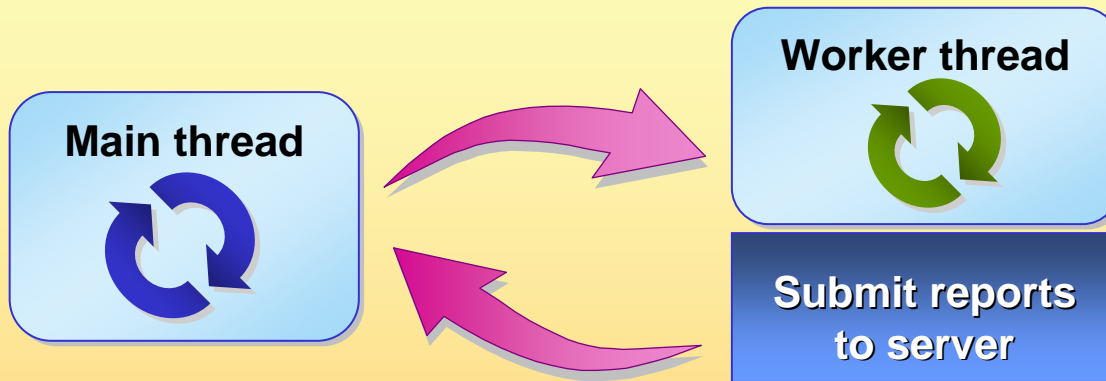
- The .NET Asynchronous Programming Model
- The Asynchronous Programming Model Design Pattern
- Protecting State and Data in a Multithreaded Environment

Lesson: The .NET Asynchronous Programming Model

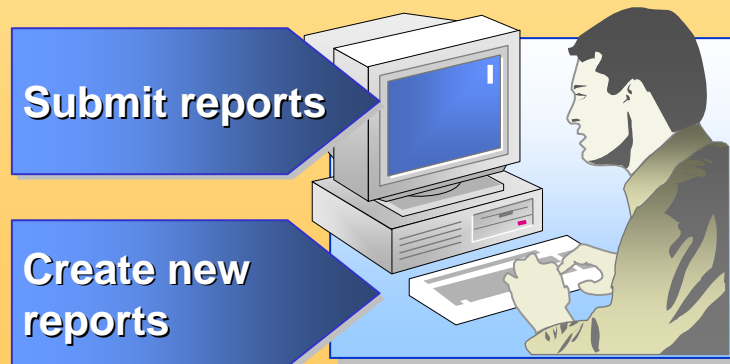
- What Is Asynchronous Programming?
- Demonstration: Comparing Synchronous and Asynchronous Versions of an Application
- Asynchronous Programming Support in the .NET Framework

What Is Asynchronous Programming?

- An application gives some work to other thread(s) while it continues doing other work on the main thread



- Useful in Windows Forms applications so users will not be blocked waiting for results and can instead continue with other work



Demonstration: Comparing Synchronous and Asynchronous Versions of an Application

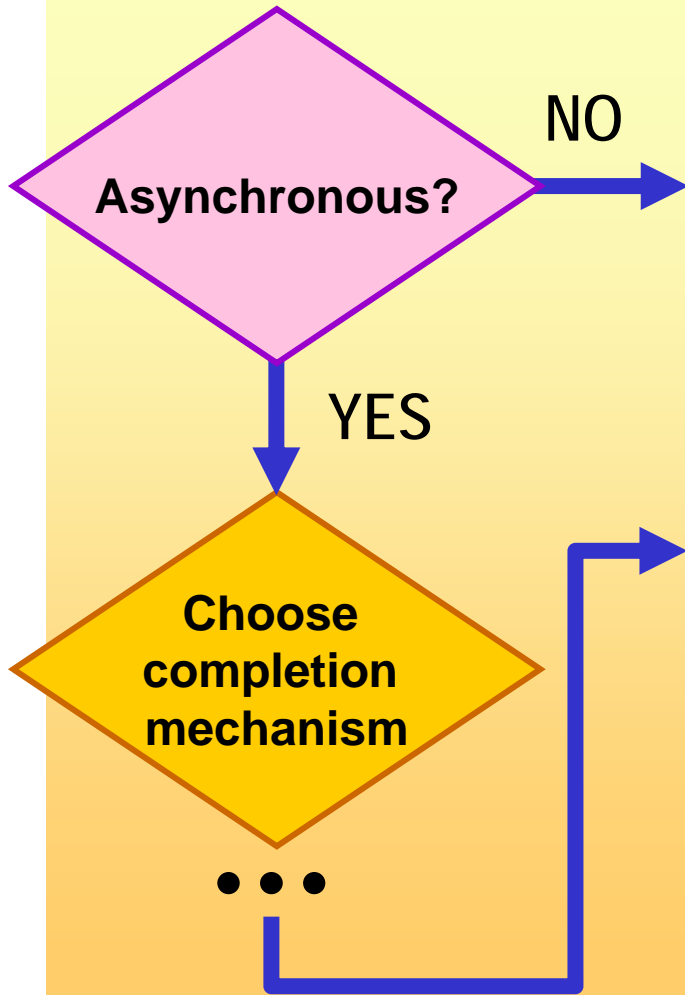


In this demonstration, you will be able to compare the user experience in synchronous and asynchronous versions of the Expense Report application

Asynchronous Programming Support in the .NET Framework

- **A design pattern for asynchronous programming**
 - Used by the .NET Framework to make asynchronous calls uniform across different parts of the framework
 - User-created classes that support asynchronous calls should conform to this design pattern
- **Asynchronous support is provided in many of the logical areas**
 - I/O, sockets, networking, ASP.NET and XML Web services, messaging, and asynchronous delegates
 - Implementation is transparent, call the appropriate methods and let the .NET Framework handle the details

Lesson: The Asynchronous Programming Model Design Pattern



```
'  
  ...  
' ... call Operation  
' ... wait for return  
' ... continue processing  
' ...
```

```
'  
  ...  
' ... call BeginOperation  
' ... operation begun on  
' ... another thread  
' ... continue with other  
' ... processing  
' ... receive results  
' ... process results  
' ...
```

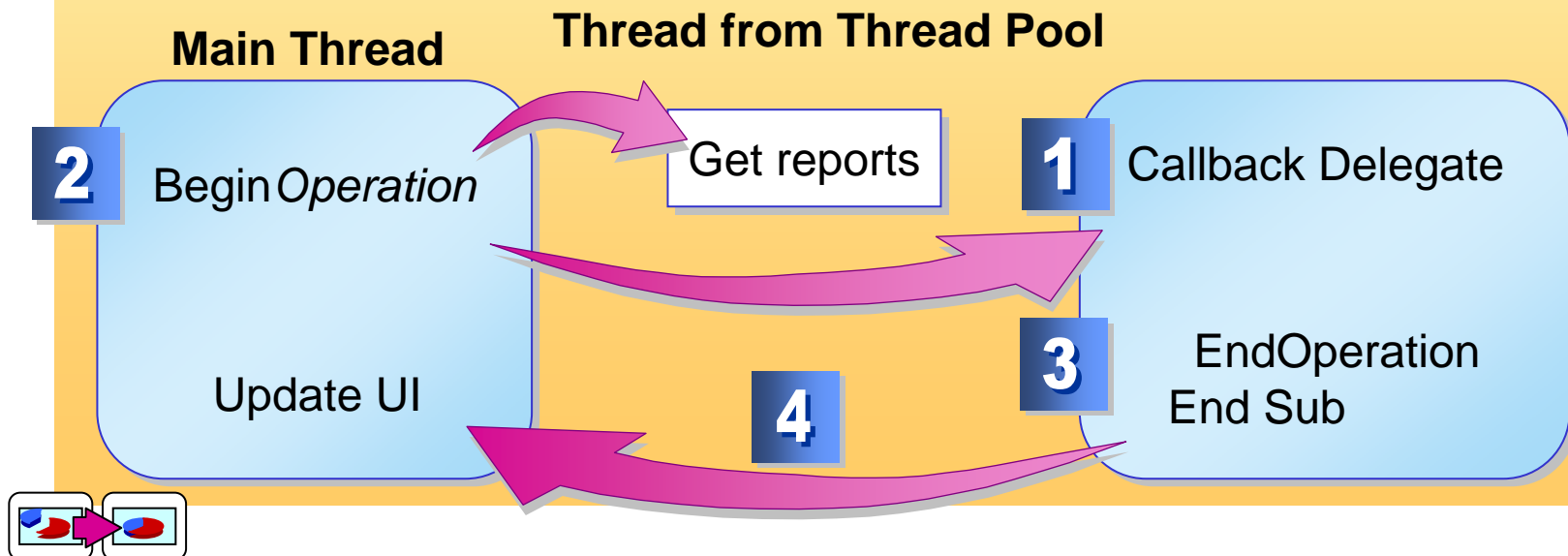
Overview of the Asynchronous Programming Model Design Pattern

- Caller decides whether a particular call should be asynchronous
- Asynchronous operation logically split into two parts
 1. Client begins the operation by calling the *BeginOperation* method
 2. Client notified that operation is complete and receives results

Completion Technique	Comments
Use a callback	Supply a callback delegate, method will be called when operation completes
Poll	Poll the IAsyncResult interface's IsCompleted property
Call the <i>EndOperation</i> method	Call the <i>EndOperation</i> method and block till operation completes
Wait on a handle	Wait on IAsyncResult interface's WaitHandle property, then call <i>EndOperation</i> method

Using the Design Pattern with an Asynchronous Callback for Completion

- 1 Create the asynchronous callback delegate
- 2 Invoke the *BeginOperation* method, passing it the callback delegate
- 3 Inside the callback, invoke the *EndOperation* method to notify that asynchronous work is complete and to return results
- 4 Return control to the main thread and update UI



How to Set Up and Initiate the Call

- 1 Create the asynchronous callback delegate

Asynchronous callback delegate



```
Dim delCB As New AsyncCallback(AddressOf _  
    Me.AsyncCB)
```


- 2 Invoke the *BeginOperation* method, passing it the callback delegate

Invoke the *BeginOperation* method



```
WS.BeginGetReportsForEmployee( _  
    username, pwdToken, _  
    RecordCursor, 10, TotalNumRecords, _  
    delCB, Nothing)
```

Callback delegate is passed in to the *BeginOperation* method



How to Receive Completion Notification and Results

3

Inside the callback, invoke the *EndOperation* method to retrieve the results of the asynchronous call

```
' Inside the callback method, AsyncCB, call  
' EndOperation to get results of the async call
```

```
Sub AsyncCB(ByVal result As IAsyncResult)
```

```
    ...  
    ExpRepDataSet = WS.EndGetReportsForEmployee( _  
        ar, TotalNumRecords)
```

```
    ...  
End Sub
```

Invoke *EndOperation* method

Receive results

How to Return Control to the Main Thread

In Windows Forms applications, any calls to methods or properties for controls on the form must be done on the main thread

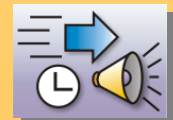
4 Return control to the main thread

```
' Switch back to main thread to update the UI
' First, create a MethodInvoker delegate for
' the method to be called
Dim mi As New MethodInvoker(Me.UpdateUI)

' Use the current form's BeginInvoke to
' invoke the delegate
Me.BeginInvoke(mi)
```

Animation: Making an Asynchronous Call to an XML Web Service

In this animation, you will see how an object makes an asynchronous call to an XML Web service



Practice: Making an Asynchronous Call to an XML Web Service



In this practice, you will

- Observe the behavior of the version of the Expense Report application that makes synchronous calls to the XML Web service
- Modify the Expense Report application so that it makes asynchronous calls to the XML Web service
- Rebuild the application and observe how the behavior of the Expense Report application has changed

**Begin reviewing the objectives for
this practice activity**



15 min

Lesson: How to Make Asynchronous Calls to Any Method

- Overview of How to Make Asynchronous Calls to Any Method
- How to Create the Asynchronous Delegate
- How to Initiate the Asynchronous Call
- How to Complete the Asynchronous Call
- How to Return Control to the Main Thread
- Demonstration: Using Asynchronous Delegates

Overview of How to Make Asynchronous Calls to Any Method

1 You must explicitly create and call a delegate for the method that you want to invoke

2 Follow the design pattern for asynchronous programming

- Initiate the call
- Complete the call
- Return data (if applicable) and control to the main thread

How to Create the Asynchronous Delegate

1 Declare the delegate

Delegate keyword




```
Delegate Function CalcDelegate ( _  
    ByVal startingValue As Integer, _  
    ByVal interestRate As Integer) _  
    As Integer
```

The delegate's signature matches that of the method it will point to

2 Instantiate the delegate, passing in the method that the delegate points to

```
' Instantiate class that contains method delegate points to  
Dim tr As New TotalReturnCalc()  
' Instantiate the delegate, passing it the method to call  
Dim cd As New CalcDelegate(tr.CalculateReturn)
```

The method that you want the delegate to point to



How to Initiate the Asynchronous Call

1 Create the delegate to the callback method

2 Call the BeginInvoke method

- When using a callback method, pass in the delegate for the callback method
- Returns an object implementing **IAsyncResult**

**Method that will receive the
callback notification**

```
' create AsyncCB delegate to callback method
Dim cb As New AsyncCallback(Me.ResultsCB)

' call BeginInvoke to asynchronously call the method
Dim ar As IAsyncResult = cd.BeginInvoke(startVal, intRate, _
    cb, Nothing)
```

Callback delegate passed in to BeginInvoke

How to Complete the Asynchronous Call

1 Call the EndInvoke method

Returns a return value or a data structure that includes a return value

```
' inside the callback method called ResultsCB
Sub ResultsCB(ByVal ar As IAsyncResult)

    ...
    Dim result As Integer = cd.EndInvoke(ar)
    ...
End Sub
```

Use EndInvoke to return results



2 Update the UI to reflect the results of the operation

When using Windows Forms, this involves returning control back to the main UI thread because Windows Forms can only be safely called from the main thread

How to Return Control to the Main Thread and Update the UI

1 Instantiate a MethodInvoker delegate for the UI update method

' Switch back to main thread before updating UI

```
Dim mi As New MethodInvoker(Me.UpdateUI)
```

2 Asynchronously call the MethodInvoker delegate

' Use BeginInvoke to call the MethodInvoker, because
' it returns worker thread to thread pool faster

```
Me.BeginInvoke(mi)
```

Demonstration: Using Asynchronous Delegates



In this demonstration, you will see how to call any method asynchronously from a Windows Forms application

Lesson: Protecting State and Data in a Multithreaded Environment

- How to Protect State and Data in a Multithreaded Environment
- Demonstration: Protecting State and Data in a Multithreaded Environment

How to Protect State and Data in a Multithreaded Environment

- **Automatic Synchronization**
 - Potential overhead incurred
- **Synchronized Code Region**
 - Monitor class
- **Manual Synchronization**
 - **Mutex class**
 - **ReaderWriterLock class**
 - **Interlocked.Increment and Interlocked.Decrement methods**
- **Design applications to try to minimize synchronization needs**

Demonstration: Protecting State and Data in a Multithreaded Environment

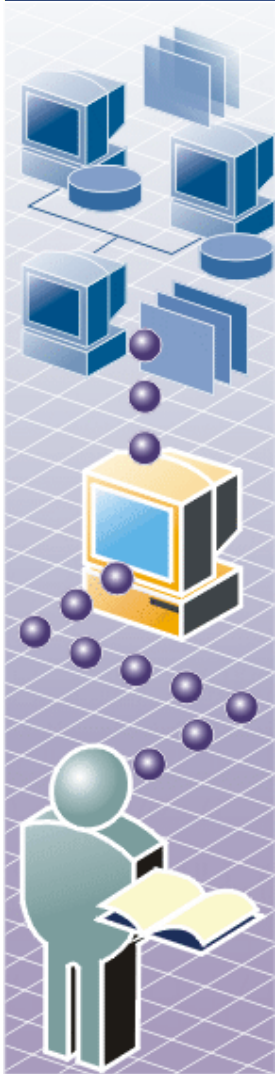


In this demonstration, you will see how to protect state and data by using a synchronized code region

Review

- The .NET Asynchronous Programming Model
- The Asynchronous Programming Model Design Pattern
- Protecting State and Data in a Multithreaded Environment

Lab 7.1: Making Asynchronous Calls to an XML Web Service



- Exercise 1: Converting Synchronous Calls to Asynchronous Calls