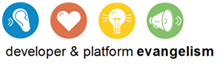
* 1. 

Demo Script

System.Threading.Barrier

* 1. Lab version: 1.0.0
  2. Last updated: 12/29/2010
  3. 

1. Contents

[Overview 3](#_Toc281385282)

[Key Messages 3](#_Toc281385283)

[Key Technologies 3](#_Toc281385284)

[Time Estimates 4](#_Toc281385285)

[Setup and Configuration 4](#_Toc281385286)

[Opening Statement 4](#_Toc281385287)

[Step-by-Step Walkthrough 4](#_Toc281385288)

[Segment #1 – Exploring the Barrier Class 5](#_Toc281385289)

[Segment #2 – Canceling the Threads Execution 9](#_Toc281385290)

[Summary 11](#_Toc281385291)

Overview

* 1. This document provides setup documentation, systematic instructions, and a written script for showing the new Barrier class of the System.Threading namespace. This document can also serve as a tutorial or walkthrough of the exposed technologies.
  2. A group of tasks cooperates by moving through a series of phases, where each in the group signals it has arrived at the Barrier in a given phase and implicitly waits for all others to arrive. The same Barrier can be used for multiple phases.
  3. In **DriveToBoston** method of the Demo Scenario, look at the following code:
  4. C#
  5. sync.SignalAndWait(token);
  6. Visual Basic
  7. sync.SignalAndWait(token)
  8. With every call to **SignalAndWait**, the number of signals received by the barrier is incremented. Once the number of signals received reaches the number of participants the Barrier was constructed with, all threads are then allowed to continue execution.

### Key Messages

* 1. The Barrier class enforces the stopping of execution between a number of threads preventing further execution until all threads have reached the given point.
  2. The Cancellation Token is used to cancel the thread execution when they arrive at a barrier. An OperationCancelledException is thrown at this point.

### Key Technologies

This demo uses the following technologies:

* 1. Microsoft Visual Studio 2010

### Time Estimates

* + Estimated time to complete the demo: **20 min**.

Setup and Configuration

## System Requirements

* + Microsoft Visual Studio 2010

Opening Statement

* 1. Today I would like to walk you through several demo applications built to show the new Barrier class of the System.Threading namespace. This very simple demo relies on the .NET Framework 4 release.
  2. The following is a brief description of the things that will be shown during this demo:
  3. Exploring the Barrier class methods and properties
  4. Canceling the thread execution with the Cancellation Token

Step-by-Step Walkthrough

1. This demo is composed of the following segments:
   1. Exploring the Barrier class
   2. Canceling the threads execution

### Segment #1 – Exploring the Barrier Class

|  |  |  |
| --- | --- | --- |
| Action | Script | Screenshot |
| * 1. Open Microsoft Visual Studio 2010 from **Start | All Programs**.   2. Open the **BarrierDemo.sln** solution located under the **Source** folder of this demo (and choosing the folder that matches the language of your preference.)   3. In the **View** menu, click **Object Browser**. | * + First, we will explore the Barrier class to see its methods and properties. To do this, we will open the Object Browser and search for this class.   + A Barrier enforces the stopping of execution between a number of threads preventing further execution until all threads have reached the given point. |  |
| * 1. Enter **System.Threading.Barrier** (*C#*) or just **Barrier** (*VB*) in the Search box and press **Enter**.   2. Show the following constructors, methods and properties of the class.      + Barrier (*C#*) / New (*VB*)      + AddParticipants      + RemoveParticipants      + SignalAndWait      + ParticipantCount      + ParticipantsRemaining   3. Close the **Object Browser**. | * + Let’s see some members of the Barrier class.   + The AddParticipant/s methods add one or more threads to the barrier.   + The Barrier constructor allows specifying the number of participating threads. Additionally, an overload of the method lets you specify the action that will be performed after the barrier. Keep in mind that the barrier can be used several times. Each time it is used, it is called a phase.   + RemoveParticipant/s eliminates participant threads.   + The SignalAndWait method is an important one. It signals that one participant thread has arrived at the barrier and waits for all the other participants to reach the barrier as well. The method has several overloads for specifying the timeout, and the cancellation token.   + Finally, it has some properties, for getting the phase number, the participant count, and the number of remaining participants to reach the current phase. |  |
| * 1. In **Solution Explorer**, open the Program.cs file by double-clicking it.   2. Show the static Variables and the **Main** method. | * + The scenario of this demo is a group of friends that want to travel to Boston, but before starting the trip, they decided to meet at a gas station. Therefore, the gas station will act as the barrier.   + Looking at the code, notice the two static variables, the Barrier and a CancellationToken.   + In the main method, we initialize the barrier with three participant threads.   + After that, we create one thread for each friend. Each thread executes the **DriveToBoston** method. |  |
| * 1. Scroll down, and show the **DriveToBoston** method. | This method simulates processing by using the sleep method. After this, it sends the signal, notifying that it has arrived to the barrier. The method also echoes the status of the process. |  |
| * 1. Scroll to the Main method and show that each thread has a different time to get to the gas station. | * + Notice that each thread has a different amount of time to arrive to the gas station. As we saw in the DriveToBoston method, under the hoods this is a call to the Thread.Sleep method. |  |
| * 1. Press F5 to run the application.   2. Press any key to close the application when it finishes. | * + Let’s run the application.   + Notice that when the three threads reach the barrier, the execution of them continues. |  |

### Segment #2 – Canceling the Threads Execution

|  |  |  |
| --- | --- | --- |
| Action | Script | Screenshot |
| * 1. Scroll up to the definition of the Static variables and show the **Main** method where the cancellation token is initialized. | * + You may have noticed that we have a static variable that holds a cancellation token. This variable is initialized below in the **Main** method with a new instance. |  |
| * 1. Scroll to the **DriveToBoston** method and show the line where the **SignalAndWait** method is used. | * + Let’s go to the **DriveToBoston** method to see that the **SignalAndWait** method is called passing the cancellation token as a parameter. |  |
| * 1. Go to the Main method.   2. Uncomment the commented line. | * + Now, we will use the Cancellation Token to cancel the tread execution. To do this, in the **Main** method, we will uncomment the commented line, so the **Cancel** method of the token is executed. The **Cancel** method communicates the request for cancellation to the threads. |  |
| * 1. Press F5 to run the application.   2. Press any key to close the application when it finishes. | * + Let’s run the application to see how this works.   + When the each thread signals with the cancellation token active, the tread terminates with an **OperationCanceledException**. |  |

Summary

In this demo, you examined the new Barrier class and its members. Additionally, you saw a simple scenario where this class is used and how the cancellation tokens work.