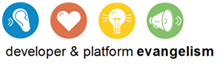
* 1. 

Demo Script

System.Threading.CountdownEvent

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

1. Contents

[Overview 3](#_Toc281387838)

[Key Messages 3](#_Toc281387839)

[Key Technologies 3](#_Toc281387840)

[Time Estimates 3](#_Toc281387841)

[Setup and Configuration 3](#_Toc281387842)

[Opening Statement 4](#_Toc281387843)

[Step-by-Step Walkthrough 4](#_Toc281387844)

[Segment #1 – Exploring the CountdownEvent Class 4](#_Toc281387845)

[Segment #2 – The CountdownEvent Class in Action 7](#_Toc281387846)

[Summary 9](#_Toc281387847)

Overview

* 1. This document provides setup documentation, systematic instructions, and a written script for showing the new **CountdownEvent** class of the System.Threading namespace. This document can also serve as a tutorial or walkthrough of the exposed technologies.
  2. The CountdownEvent class is used to wait until a set of tasks are completed to continue the execution of a program.
  3. With the **AddCount** method, the "signal to wait" count increases while with the **Signal** method this count is decreased. Once the count reaches 0, the program execution continues.

### Key Messages

* 1. The CoundownEvent class represents a synchronization primitive that is signaled when its count reaches zero.

### Key Technologies

This demo uses the following technologies:

* 1. Microsoft Visual Studio 2010

### Time Estimates

* + Estimated time to complete the demo: **15 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 CountdownEvent class of the System.Threading namespace. This very simple demo relies on the .NET Framework 4 release. The CountdownEvent represents a synchronization primitive that is signaled when its count reaches zero.
  2. The following is a brief description of the things that will be shown during this demo:
  3. Exploring the CountdownEvent class methods and properties

Step-by-Step Walkthrough

1. This demo is composed of the following segments:
   1. Exploring the CountDownEvent Class
   2. The CountDownEvent Class in Action

### Segment #1 – Exploring the CountdownEvent Class

|  |  |  |
| --- | --- | --- |
| Action | Script | Screenshot |
| * 1. Open Microsoft Visual Studio 2010 from **Start | All Programs**.   2. Open the **CountdownEventDemo.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 **CountDownEvent** class to see its methods and properties. To do this, we will open the Object Browser and search for this class.   + The CountdownEvent class is used to wait until a set of tasks are completed in order to continue the execution of a program. That is when the count reaches zero. |  |
| * 1. Enter **System.Threading.CountDownEvent** (*C#*) or just **CountDownEvent** (*VB*) in the Search box and press **Enter**.   2. Show the following methods and properties of the class.      + CountdownEvent (*C#*) / New (*VB*)      + AddCount      + Reset      + Signal      + Wait      + CurrentCount      + InitialCount   3. Close the **Object Browser**. | * + Let’s see some members of the CountDownEvent class.   + The AddCount method increments the current count by one or by the number specified in the parameter.   + The CountdownEvent constructor creates a new instance of this class and it requires the initial count to be passed as a parameter.   + The reset method resets the current count to the initial count.   + The signal method block the current thread until the count reaches zero. Notice that a cancellation token can be passed as a parameter to this method.   + The Wait method. 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, **CurrentCount** and **InitialCount**. Each **CountdownEvent** instance stores both the current and the initial count. The initial count is stored in case a reset is run. |  |

### Segment #2 – The CountdownEvent Class in Action

|  |  |  |
| --- | --- | --- |
| Action | Script | Screenshot |
| * 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 twenty customers that do shopping.   + The customers are generated by the Range method of the Enumerable class.   + The **CountdownEvent** instance is initialized with an initial counter of one. Then the counter is incremented using the **AddCount** method for each customer that want to buy things.   + When a customer finishes shopping the Signal method is invoked to decrement the current count. |  |
| * 1. Scroll down, and show the **BuySomeStuff** method. | * + This method simulates processing by using the **Thread.SpinWait** method. After this, it prints a message saying that the customer has finished shopping. |  |
| * 1. Scroll to the end of the Main method and show the final two lines. | * + Notice that when all the twenty customers are working and additional Signal is sent. This is because the initial count started in one. After this, the **Wait** method is invoked. At this point, the application will wait until the counter is zero. |  |
| * 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 all customers finished shopping, and thus the current counter reaches zero, the application will continue to the end, printing the ending message. |  |

Summary

In this demo, you examined the new **CoundownEvent** class and its members. Additionally, you saw a simple scenario where this class is used.