Module 03:
"Signaling with Events and Handles"

TEKNOLOGISK
INSTITUT

Agenda

▸ **Signaling with Events**
▸ Using Wait Handles

AutoResetEvent

AutoResetEvent

- ▸ Derives from the general `EventWaitHandle`
  - Is automatically reset when first thread passes `WaitOne()`

```
void Update()
{
    _event.Set();    // Signals event
    ...;
}
```

```
void Compute()
{
    _event.WaitOne(); // <-- Waits for event to be signalled
                      // <-- Automatically resets event
    ...
}
```

- ▸ `WaitOne()` has overloads providing timeouts

- ▸ Can be both local and cross-process (*)!

ManualResetEvent

ManualResetEvent

- Derives from the general `EventWaitHandle`
  - Needs to be manually reset by invoking `Reset()`

```
void Update()
{
    _event.Set();    // Signals event
    ...;
    _event.Reset(); // <-- Note: Necessary to reset manually
```

```
void Compute()
{
    _event.WaitOne(); // <-- Waits for event to be signalled
    ...
}
```

- `WaitOne()` has overloads providing timeouts

- Can be both local and cross-process (*)!

ManualResetEventSlim

▸ Introduced in .NET 4.0
▸ Optimized, local-only version of **ManualResetEvent**
  • About 10-50 times more performant!

▸ Does not derive from **EventWaitHandle**, but exposes a **WaitHandle** property

▸ **WaitHandle**   has   **WaitOne()**
  • **EventWaitHandle**   has   **Set() + Reset()**
    • **AutoResetEvent**
    • **ManualResetEvent**

CountdownEvent

CountdownEvent

▸ Counts down from specific count (e.g. 5) by `Signal()`
  • `Wait()` blocks until count reaches 0

```
void Update()
{

    _event.Wait();  // Wait for 5 x Signal()
    ...;
    _event.Reset(); // <-- Resets counter to 5
}
```

```
void Compute()
{

    _event.Signal(); // <-- Decrements counter by 1
    ...
}
```

▸ `Wait()` has overloads providing timeouts and cancellation

▸ Fully managed (not an `EventWaitHandle`, but exposes `WaitHandle`)
▸ Can only be local (*)

Agenda

▸ Signaling with Events
▸ **Using Wait Handles**

WaitHandle

▸ Why is **WaitHandle** or not so important? ☺

▸ **WaitHandle**   has   **WaitOne()**
   • **EventWaitHandle**   has   **Set() + Reset()**
      • **AutoResetEvent**
      • **ManualResetEvent**
   • **Mutex**
   • **Semaphore**

▸ Everything is **WaitHandle** or exposes **WaitHandle**
▸ Can be treated uniformly in certain scenarios

Advanced Synchronization

▸ `WaitHandle.`
- `WaitAny()`
- `WaitAll()`
- `SignalAndWait()`   `~ Set() + WaitOne()`

▸ Can implement many advanced scenarios such as
- Thread Rendezvous
- Compound waiting
- …

Barriers

▸ **Barrier**
- Introduced in .NET 4.0 as an atomic Thread Execution Barrier
- Signals 1 and waits for the total count to be signaled
- Atomically resets count when 0 is reached and unblocks
- **Barrier** ~ "AutoCountdownEvent"

```
Barrier _barrier = new Barrier(4, barrier =>
    WriteLine($"{barrier.ParticipantCount}-thread rendezvous")
);
```

```
void Compute()
{
    ...
    _barrier.SignalAndWait();
    Console.WriteLine(result);
}
```

Summary

- ▸ **Signaling with Events**
- ▸ **Using Wait Handles**

WINCUBATE

Jesper Gulmann Henriksen
PhD, MCT, MCSD, MCPD

Phone : +45 22 12 36 31
Email   : jgh@wincubate.net
WWW : http://www.wincubate.net

Ringgårdsvej 4A
8270 Højbjerg
Denmark