Example14~16

CCR Task Scheduling

Example 14: Posting Items

```
using System;
using System.Collections.Generic;
using System.Text;
using Microsoft.Ccr.Core;

namespace Example
{
    class Program
    {
        //Example 14: Posting Items
        static void Main(string[] args)
        {
            /// <summary>
            /// Arbiter即是在實現ITask (IterativeTask)的實作，安排進出資料的處理方式
            /// DispatcherQueue為FIFO的Queue
            ///
            /// Dispatcher為OS系統中的Thread管控者，主要調動DispatcherQueue中所擁有的任務(ITask)
            /// 調度指令的權重與掌握CPU的執行緒
            ///
            /// Example 16 程式描述：
            /// 1.建構管控每顆CPU上之Thread資源的Dispatcher管理者 (ThreadsPerCpu: The number of threads
per CPU)
            /// 2.讓它管一個DispatcherQueue (DispatcherQueue: first in first out (FIFO) queue of Tasks)
            /// 3.建構一個Port<int>，用以存放資料
            /// 4.用Arbiter.Activate啟動一個Arbiter.Receive來監聽Port中的資料
            /// 5.發布一個變數資料到port中
            ///
            /// 當資料被放到具有receiver的port時，會發生下列情況：
            /// 1.產生一個容器(container, IPortElement)來存放資料
            /// 2.The container instance is queued.
            /// 3.發現有東西丟進來，透過ReceiverTask.Evaluate 產生Task<int>
            /// 4.呼叫完ReceiverTask.Evaluate後，port會執行taskQueue.Enqueue
            /// 當完成4之後，Task的實作便受scheduling logic控制
            ///
            /// </summary>

            var dispatcher = new Dispatcher(
                0, // zero means use one thread per CPU, or 2 if only one CPU present
                "sample dispatcher" // friendly name assgined to OS threads
                );
```

```

            var taskQueue = new DispatcherQueue(
                "sample queue", // friendly name
                dispatcher // dispatcher instance
                );

            var port = new Port<int>();
            Arbiter.Activate(taskQueue,
               Arbiter.Receive(
                   true,
                   port,
                   item => Console.WriteLine(item)
               )
            );

            // post item, so delegate executes
            port.Post(5);
        }
    }
}
```

Example 15: Task.Execute

```
using System;
using System.Collections.Generic;
using System.Text;
using Microsoft.Ccr.Core;

namespace Example
{
    class Program
    {
        //Example 15: Task.Execute
        static void Main(string[] args)
        {
            /// <summary>
            /// 與Example 16相同，但不透過port傳遞資料
            /// 而是透過Task.Execute執行任務，直接執行任務
            /// 當資料一傳入後，可馬上執行
            /// </summary>

            var dispatcher = new Dispatcher(
                0, // zero means use one thread per CPU, or 2 if only one CPU present
                "sample dispatcher" // friendly name assgined to OS threads
                );

            var taskQueue = new DispatcherQueue(
                "sample queue", // friendly name
                dispatcher // dispatcher instance
```

```
                );

            // directly enqueue a task with an inlined method plus a parameter
            taskQueue.Enqueue(
                new Task<int>(5, item => Console.WriteLine(item))
                );
        }
    }
}
```

Example 16: Throttling

```
using System;
using System.Collections.Generic;
using System.Text;
using Microsoft.Ccr.Core;

namespace Example
{
    class Program
    {
        //Example 16: Throttling
        static void Main(string[] args)
        {
            /// <summary>
            /// 控制處理事件的閘門
            /// Unconstrained 啥都不設限，預設值
            /// ConstrainQueueDepthDiscardTasks 設定數量上限，當超過這數量將最舊的未處理均丟掉
            /// ConstrainQueueDepthThrottleExecution 設定數量上限，當超過這數量者，停止任務產生等到可放入為止
            /// ConstrainSchedulingRateDiscardTasks 設定時間頻率，確保每段時間執行一次，在期間全部資料丟掉
            /// ConstrainSchedulingRateThrottleExecution 聽取別的執行緒上的訊息產生，固定頻率的執行並管控訊息產生的執行緒啟動，確保所有訊息被處理到
            /// </summary>

            int maximumDepth = 2;
            double rate = 2.0;
            Dispatcher dispatcher = new Dispatcher(0, "throttling example");
            DispatcherQueue depthThrottledQueue = new DispatcherQueue("ConstrainQueueDepthDiscard",
                  dispatcher,
                  TaskExecutionPolicy.ConstrainQueueDepthDiscardTasks, maximumDepth);  // lost data
            //TaskExecutionPolicy.ConstrainQueueDepthThrottleExecution, maximumDepth);  // no data lost
            //TaskExecutionPolicy.ConstrainSchedulingRateDiscardTasks, rate);  // get data by rate, lost data
            //TaskExecutionPolicy.ConstrainSchedulingRateThrottleExecution, rate);  // get data by rate, no data lost

```

```
138             Port<int> intPort = new Port<int>();
139             Arbiter.Activate(depthThrottledQueue,
140                Arbiter.Receive(true, intPort,
141                delegate(int i)
142                {
143                    // only some items will be received since throttling will discard most of them
144                    Console.WriteLine(i);
145                })
146             );
147 
148             // post items as fast as possible so that the depth policy is activated and discards
149             // all the oldest items in the dispatcher queue
150             for (int i = 0; i < maximumDepth * 100000; i++)
151             {
152                 intPort.Post(i);
153             }
154         }
155     }
156 }
```