Example07~13

CCR Coordination PrimitivesCCR

Example 07: Create a Port<int> and then Activate a Single Item Receiver

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

namespace Example
{
    class Program
    {
        // Example 07: Create a Port<int> and then Activate a Single Item Receiver

        // create dispatcher and dispatcher queue for scheduling tasks
        static Dispatcher dispatcher = new Dispatcher();
        static DispatcherQueue _taskQueue = new DispatcherQueue("sample queue", dispatcher);

        static void Main(string[] args)
        {
            /// <summary>
            /// 示範主動抓取Port內的資料方式
            /// </summary>

            var port = new Port<int>();
            Arbiter.Activate(_taskQueue,  // Arbiter.Activate: 使用某個 DispatcherQueue 來執行一系列的
Tasks.
               Arbiter.Receive(
                   true,  // false: execute one item then un-register from the port
                   port,
                   item => Console.WriteLine(item)
               )
            );

            // post item, so delegate executes
            port.Post(5);
            port.Post(7);  // 平行處理
        }
    }
}
```

Example 08: The Same with Example 07, Passing Arbiter Class Factory Receiver

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

namespace Example
{
    class Program
    {
        // Example 08: The Same with Example 07, Passing Arbiter Class Factory Receiver
        static void Main(string[] args)
        {
            /// <summary>
            /// 間接啟動抓取資料
            /// </summary>

            var port = new Port<int>();

            // alternate version that explicitly constructs a Receiver by passing
            // Arbiter class factory methods
            var persistedReceiver = new Receiver<int>(
                  true, // persisted, false: execute one time
                  port,
                  null, // 新增項目： no predicate
                  new Task<int>(item => Console.WriteLine(item)) // task to execute
                );

            Arbiter.Activate(_taskQueue, persistedReceiver);

            // post item, so delegate executes
            port.Post(5);
            port.Post(7);
        }

        // create dispatcher and dispatcher queue for scheduling tasks
        static Dispatcher dispatcher = new Dispatcher();
        static DispatcherQueue _taskQueue = new DispatcherQueue("sample queue", dispatcher);
    }
}
```

Example 09: Choice Arbiter

```
using System;
using System.Collections.Generic;
```

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

namespace Example
{
    class Program
    {
        // Example 09: Choice Arbiter
        static void Main(string[] args)
        {
            // create a simple service listening on a port
            ServicePort servicePort = SimpleService.Create(_taskQueue);  // 呼叫「ServicePort Create」

            // Updtae the state of SimpleDervice
            UpdateState _state = new UpdateState();  // public string State;
            _state.State = "Test";
            //servicePort.Post(_state);  // success or failure

            // create request
            GetState get = new GetState();

            // post request
            servicePort.Post(get);

            // use the extension method on the PortSet that creates a choice
            // given two types found on one PortSet. This a common use of
            // Choice to deal with responses that have success or failure
            Arbiter.Activate(_taskQueue,
                get.ResponsePort.Choice(
                    s => Console.WriteLine(s), // delegate for success
                    ex => Console.WriteLine(ex) // delegate for failure
            ));
        }

        // create dispatcher and dispatcher queue for scheduling tasks
        static Dispatcher dispatcher = new Dispatcher();
        static DispatcherQueue _taskQueue = new DispatcherQueue("sample queue", dispatcher);

        #region Example 09: function
        /// <summary>
        /// Example 09
        /// </summary>
        public class ServiceOperation
        {
            public PortSet<string, Exception> ResponsePort = new PortSet<string, Exception>();  // 建
立ResponsePort的PortSet
        }

        public class Stop : ServiceOperation
        {
        }

```

```
138         public class UpdateState : ServiceOperation
139         {
140             public string State;
141         }
142
143         public class GetState : ServiceOperation
144         {
145         }
146
147         /// <summary>
148         /// PortSet that defines which messages the services listens to
149         /// </summary>
150         public class ServicePort : PortSet<Stop, UpdateState, GetState>
151         {
152         }
153         /// <summary>
154         /// Simple example of a CCR component that uses a PortSet to abstract
155         /// its API for message passing
156         /// </summary>
157         public class SimpleService
158         {
159             ServicePort _mainPort;
160             DispatcherQueue _taskQueue;
161             string _state;
162
163             public static ServicePort Create(DispatcherQueue taskQueue)
164             {
165                 var service = new SimpleService(taskQueue);
166                 service.Initialize();
167                 return service._mainPort;
168             }
169
170             private void Initialize()
171             {
172                 // using the supplied taskQueue for scheduling, activate two
173                 // persisted receivers, that will run concurrently to each other,
174                 // one for each item type
175                 Arbiter.Activate(_taskQueue,
176                     Arbiter.Receive<UpdateState>(true, _mainPort, UpdateHandler),
177                     Arbiter.Receive<GetState>(true, _mainPort, GetStateHandler)
178                 );
179             }
180
181             private SimpleService(DispatcherQueue taskQueue)
182             {
183                 // create PortSet instance used by external callers to post items
184                 _mainPort = new ServicePort();
185
186                 // cache dispatcher queue used to schedule tasks
187                 _taskQueue = taskQueue;
188             }
189
```

```
            void GetStateHandler(GetState get)
            {
                if (_state == null)
                {
                    // To demonstrate a failure response,
                    // when state is null will post an exception
                    get.ResponsePort.Post(new InvalidOperationException());  //
InvalidOperationException 內建錯誤回傳函式
                    return;
                }

                // return the state as a message on the response port
                get.ResponsePort.Post(_state);
            }
            void UpdateHandler(UpdateState update)
            {
                // update state from field in the message
                _state = update.State;

                // as success result, post the state itself
                update.ResponsePort.Post(_state);
            }
        }
        #endregion
    }
}
```

Example 10: Joint

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

namespace Example
{
    class Program
    {
        // Example 10: Joint
        static void Main(string[] args)
        {
            /// <summary>
            /// Using Joint to listen the same port will be a race, It cant guarentee that the Arbiter
will run.
            /// </summary>

            var portDouble = new Port<double>();
            var portString = new Port<string>();

```

```
            // activate a joined receiver that will execute only when one
            // item is available in each port.
            Arbiter.Activate(_taskQueue,
                portDouble.Join(
                    portString, // port to join with
                    (value, stringValue) => // delegate
                    {
                        value /= 2.0;
                        stringValue = value.ToString();
                        // post back updated values
                        portDouble.Post(value + 1);
                        portString.Post(stringValue + 1);
                        Console.WriteLine("stringValue: " + stringValue);
                        Console.WriteLine("value:" + value);
                    })
                );

            // post items. The order does not matter, which is what Join its power
            portDouble.Post(3.14159);
            portString.Post("0.1");

            double item;
            long totalSum = 0;
            for (long i = 1; i <= 10000000; i++)
                totalSum += i;
            portDouble.Test(out item);
            Console.WriteLine("portDouble Received item:" + item);

            //after the last post the delegate above will execute
        }

        // create dispatcher and dispatcher queue for scheduling tasks
        static Dispatcher dispatcher = new Dispatcher();
        static DispatcherQueue _taskQueue = new DispatcherQueue("sample queue", dispatcher);
    }
}
```

Example 11: Joint Race

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

namespace Example
{
    class Program
    {
        // Example 11: Joint Race
```

```
284         static void Main(string[] args)
285         {
286             /// <summary>
287             /// Using Joint to listen the same port will be a race, It cant guarentee that the Arbiter
288 will run.
289             /// </summary>
290 
291             var portInt = new Port<int>();
292             var portDouble = new Port<double>();
293             var portString = new Port<string>();
294 
295             // activate a joined receiver that will execute only when one
296             // item is available in each port.
297             Arbiter.Activate(_taskQueue,
298                 portDouble.Join(
299                     portString, // second port to listen
300                     (value, stringValue) =>
301                     {
302                         value /= 2.0;
303                         stringValue = value.ToString();
304                         // post back updated values
305                         portDouble.Post(value);
306                         portString.Post(stringValue);
307                         Console.WriteLine("First stringValue:" + stringValue);
308                         Console.WriteLine("First value:" + value);
309                     })
310                 );
311 
312             // activate a second joined receiver that also listens on portDouble
313             // and on a new port, portInt. Because the two joins share a common port
314             // between them (portDouble), there is contention when items are posted on
315             // that port
316             Arbiter.Activate(_taskQueue,
317                 portDouble.Join(
318                     portInt, // second port to listen
319                     (value, intValue) =>
320                     {
321                         value /= 2.0;
322                         intValue = (int)value;
323                         // post back updated values
324                         portDouble.Post(value);
325                         portInt.Post(intValue);
326                         Console.WriteLine("Second intValue:" + intValue);
327                         Console.WriteLine("Second value:" + value);
328                     })
329                 );
330 
331             // post items.
332             portString.Post("Test");
333             portInt.Post(128);
334 
335             // when the double is posted there will be a race
```

```
            // between the two joins to determine who will execute first
            // The delegate that executes first will then post back a double,
            // allowing the delegate that "lost", to execute.
            portDouble.Post(3.14159);
            //portString.Post("Test");
        }

        // create dispatcher and dispatcher queue for scheduling tasks
        static Dispatcher dispatcher = new Dispatcher();
        static DispatcherQueue _taskQueue = new DispatcherQueue("sample queue", dispatcher);
    }
}
```

Example 12: Joint Wait

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

namespace Example
{
    class Program
    {
        // Example 12: Joint Wait
        static void Main(string[] args)
        {
            /// <summary>
            /// 全員到齊才會跑
            /// </summary>

            int itemCount = 10;
            var portDouble = new Port<double>();

            // post N items to a port
            for (int i = 0; i < itemCount; i++)
            {
                portDouble.Post(i * 3.14159);
                long totalSum = 0;
                for (long j = 1; j <= 100000000; j++)
                    totalSum += j;
                Console.WriteLine("i: " + i);
            }

            // activate a Join that
            // waits for N items on the same port
            Arbiter.Activate(_taskQueue,
                portDouble.Join(
                    itemCount,  // wait itemCount times
```

```
                    items =>
                    {
                        foreach (double d in items)
                        {
                            Console.WriteLine(d);
                        }
                    }
                )
            );
        }

        // create dispatcher and dispatcher queue for scheduling tasks
        static Dispatcher dispatcher = new Dispatcher();
        static DispatcherQueue _taskQueue = new DispatcherQueue("sample queue", dispatcher);
    }
}
```

Example 13: Multiple Item receivers

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

namespace Example
{
    class Program
    {
        // Example 13: Multiple Item receivers
        static void Main(string[] args)
        {
            /// <summary>
            /// 全員到齊才會跑
            /// </summary>

            // create a simple service listening on a port
            var servicePort = SimpleService.Create(_taskQueue);  // 呼叫「ServicePort Create」

            // shared response port
            var responsePort = new PortSet<string, Exception>();

            // number of requests
            int requestCount = 10;

            // scatter phase: Send N requests as fast as possible
            for (int i = 0; i < requestCount; i++)
            {
                long totalSum = 0;
                for (long j = 1; j <= 100000000; j++)
```

```
                    totalSum += j;
                Console.WriteLine("i: " + i);

                // Updtae the state of SimpleDervice
                string item;
                UpdateState _state = new UpdateState();  // public string State;
                _state.State = "Test";
                if (i > 6)
                    servicePort.Post(_state);  // success

                // create request
                GetState get = new GetState();

                // set response port to shared port
                get.ResponsePort = responsePort;

                // post request
                servicePort.Post(get);
            }

            // gather phase:
            // activate a multiple item receiver that waits for a total
            // of N responses, across the ports in the PortSet.
            // The service could respond with K failures and M successes (K+M == N)
            Arbiter.Activate(_taskQueue,
                responsePort.MultipleItemReceive(
                    requestCount, // wait requestCount times
                    (successes, failures) => Console.WriteLine("Total received (s vs f):" +
successes.Count + " " + failures.Count)
                    )
                );
        }

        // create dispatcher and dispatcher queue for scheduling tasks
        static Dispatcher dispatcher = new Dispatcher();
        static DispatcherQueue _taskQueue = new DispatcherQueue("sample queue", dispatcher);

        #region Example 13: function
        /// <summary>
        /// Example 13
        /// </summary>
        public class ServiceOperation
        {
            public PortSet<string, Exception> ResponsePort = new PortSet<string, Exception>();  // 建
立ResponsePort的PortSet
        }

        public class Stop : ServiceOperation
        {
        }

        public class UpdateState : ServiceOperation
```

```
{
    public string State;
}

public class GetState : ServiceOperation
{
}

/// <summary>
/// PortSet that defines which messages the services listens to
/// </summary>
public class ServicePort : PortSet<Stop, UpdateState, GetState>
{
}
/// <summary>
/// Simple example of a CCR component that uses a PortSet to abstract
/// its API for message passing
/// </summary>
public class SimpleService
{
    ServicePort _mainPort;
    DispatcherQueue _taskQueue;
    string _state;

    public static ServicePort Create(DispatcherQueue taskQueue)
    {
        var service = new SimpleService(taskQueue);
        service.Initialize();
        return service._mainPort;
    }

    private void Initialize()
    {
        // using the supplied taskQueue for scheduling, activate two
        // persisted receivers, that will run concurrently to each other,
        // one for each item type
        Arbiter.Activate(_taskQueue,
            Arbiter.Receive<UpdateState>(true, _mainPort, UpdateHandler),
            Arbiter.Receive<GetState>(true, _mainPort, GetStateHandler)
        );
    }

    private SimpleService(DispatcherQueue taskQueue)
    {
        // create PortSet instance used by external callers to post items
        _mainPort = new ServicePort();

        // cache dispatcher queue used to schedule tasks
        _taskQueue = taskQueue;
    }

    void GetStateHandler(GetState get)
```

```
534             {
535                 if (_state == null)
536                 {
537                     // To demonstrate a failure response,
538                     // when state is null will post an exception
539                     get.ResponsePort.Post(new InvalidOperationException());  //
540 InvalidOperationException 內建錯誤回傳函式
541                     return;
542                 }
543 
544                 // return the state as a message on the response port
545                 get.ResponsePort.Post(_state);
546             }
547             void UpdateHandler(UpdateState update)
548             {
549                 // update state from field in the message
550                 _state = update.State;
551 
552                 // as success result, post the state itself
553                 update.ResponsePort.Post(_state);
554             }
555         }
556         #endregion
557     }
558 }
```