Example17~20

CCR Iterators

Example 17: Introduction to CCR Iterators

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

namespace Example
{
    class Program
    {
        //Example 17: Introduction to CCR Iterators
        static void Main(string[] args)
        {
            /*
            // arbiter
            int totalSum = 0;
            var portInt = new Port<int>();
            for (int i = 0; i < 100; i++)
            {
                portInt.Post(i);
            }
            //Console.WriteLine("Total:" + totalSum);

            Arbiter.Activate(_taskQueue,
                // Arbiter.FromIteratorHandler: Converts an iterator handler delegate to a task so it
can be scheduled
                Arbiter.Receive<int>(true, portInt, delegate(int item)
                {
                    totalSum += item;
                    Console.WriteLine("Total:" + totalSum);
                }));
             */

            // one core
            int totalSum = 0;
            var portInt = new Port<int>();
            for (int i = 0; i < 1000000; i++)
            {
                // post current iteration value to a port
                portInt.Post(i);
                //Console.WriteLine("portInt(Befor):" + portInt);  // Port Information
```

```
                //Console.WriteLine(portInt);  // Port Data: 取出資料
                //portInt.Post(i);  // Port Data: 放回資料
                totalSum += portInt;
                //Console.WriteLine(portInt);  // Port Data: 0
            }
            Console.WriteLine("Total:" + totalSum);

            // multi-core
            // create an IterativeTask instance and schedule it
            Arbiter.Activate(_taskQueue,
                // Arbiter.FromIteratorHandler: Converts an iterator handler delegate to a task so it
can be scheduled
                Arbiter.FromIteratorHandler(IteratorExample)
            );
        }

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

        /// <summary>
        /// Iterator method scheduled by the CCR
        /// </summary>
        // IEnumerator: Supports a simple iteration over a nongeneric collection. (foreach)
        // 按順序執行，for會自動分散到所有的core計算。
        static IEnumerator<ITask> IteratorExample()  // 可列舉型別容器<ITask>
        {
            // accumulator variable
            int totalSum = 0;
            var portInt = new Port<int>();

            // using CCR iterators we can write traditional loops
            // and still yield to asynchronous I/O !
            for (int i = 0; i < 1000000; i++)
            {
                // post current iteration value to a port
                portInt.Post(i);


                // yield until the receive is satisifed. No thread is blocked
                yield return portInt.Receive(
                    /*
                    delegate(int item) // 取出資料
                    {
                        Console.WriteLine(item);  // 破壞多核執行
                        portInt.Post(item);  // 放回資料
                    }
                    */
                   );

                // retrieve value using simple assignment
                //Console.WriteLine(portInt);  // Port Data: Get Data
```

```
                totalSum += portInt;
            }
            Console.WriteLine("Total:" + totalSum);
        }
    }
}
```

Example 18: Yielding to coordination primitives

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

namespace Example
{
    class Program
    {
        //Example 18: Yielding to coordination primitives
        static void Main(string[] args)
        {
            /// <summary>
            /// Yielding 的使用法
            /// </summary>

            Port<string> portString = new Port<string>();
            Arbiter.Activate(
                _taskQueue,
                // Arbiter.ReceiveWithIterator: Creates a single item receiver for a iterator user
handler
                Arbiter.ReceiveWithIterator(true, portString, StringIteratorHandler)  // false:
execute for one item
                // difference with FromIteratorHandler
            );
            portString.Post("First");
            portString.Post("Second");
        }

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

        static IEnumerator<ITask> StringIteratorHandler(string item)
        {
            Console.WriteLine(item);
            yield break;
        }
     }
}
```

Example 19: IterativeTask Exception

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

namespace Example
{
    class Program
    {
        //Example 19: IterativeTask Exception
        static void Main(string[] args)
        {
            // create an IterativeTask instance and schedule it
            Arbiter.Activate(_taskQueue,
                Arbiter.FromIteratorHandler(IteratorWithChoice)
            );
        }

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

        static IEnumerator<ITask> IteratorWithChoice()
        {
            // create a service instance
            ServicePort servicePort = ServiceWithInterleave.Create(_taskQueue);

            // send an update request
            UpdateState updateRequest = new UpdateState();
            updateRequest.State = null;
            servicePort.Post(updateRequest);  // comment for error

            string result = null;

            // wait for either outcome before continuing
            ///*
            yield return Arbiter.Choice(
                updateRequest.ResponsePort,
                response => result = response,  // delegate (string response) { result = response; }
                ex => Console.WriteLine(ex)
            );

            // if the failure branch of the choice executed, the result will be null
            // and we will terminate the iteration
            if (result == null)
            {
                Console.WriteLine("Result == null");
                yield break;
            }
```

```
            //*/

            // print result from first request
            Console.WriteLine("UpdateState response:" + result);

            // now issue a get request
            GetState get = new GetState();
            servicePort.Post(get);

            // wait for EITHER outcome
            yield return Arbiter.Choice(
                get.ResponsePort,
                delegate(string response) { result = response; },
                delegate(Exception ex) { Console.WriteLine(ex); }
            );

            // print result from second request

            // stop function
            Stop stop = new Stop();
            servicePort.Post(stop);
        }
    }

    /// <summary>
    /// Base type for all service messages. Defines a response PortSet used
    /// by all message types.
    /// </summary>
    public class ServiceOperation
    {
        public PortSet<string, Exception> ResponsePort = new PortSet<string, Exception>();
    }

    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>
    {
    }
```

```
245
246         /// <summary>
247         /// Simple example of a CCR component that uses a PortSet to abstract
248         /// its API for message passing
249         /// </summary>
250         public class ServiceWithInterleave
251         {
252             ServicePort _mainPort;
253             DispatcherQueue _taskQueue;
254             string _state;
255
256             public static ServicePort Create(DispatcherQueue taskQueue)
257             {
258                 var service = new ServiceWithInterleave(taskQueue);
259                 service.Initialize();
260                 return service._mainPort;
261             }
262
263             private void Initialize()
264             {
265                 // activate an Interleave Arbiter to coordinate how the handlers of the service
266                 // execute in relation to each other and to their own parallel activations
267                 Arbiter.Activate(_taskQueue,
268                     Arbiter.Interleave(
269                     // TeardownReceiverGroup: Receivers that will execute one time only after all concurrent
270 and exclusive handlers are finished.
271                     // The interleave will then unregister all receivers from ports and prevent any further
272 handlers form executing.
273                     // If multiple teardown receivers are present, only one will execute, the one that first
274 has its constraints
275                     // (first one that receives a message for simple receives) met.
276                     new TeardownReceiverGroup(  // 會跟據<Stop>,<UpdateState>選擇
277 TeardownReceiverGroup,ExclusiveReceiverGroup
278                     // one time, atomic teardown
279                         Arbiter.Receive<Stop>(false, _mainPort, StopHandler)  // true, error
280                     ),
281                     // ExclusiveReceiverGroup: Receivers with user delegates that must run exclusive to each
282 other and any receiver in a Concurrent group
283                     new ExclusiveReceiverGroup(
284                     // Persisted Update handler, only runs if no other handler running
285                         Arbiter.Receive<UpdateState>(true, _mainPort, UpdateHandler)
286                     ),
287                     // ConcurrentReceiverGroup: Receivers with user delegates that can run concurrently with
288 each other
289                     new ConcurrentReceiverGroup(
290                     // Persisted Get handler, runs in parallel with all other activations of itself
291                     // but never runs in parallel with Update or Stop
292                         Arbiter.Receive<GetState>(true, _mainPort, GetStateHandler)
293                     ))
294                 );
295             }
296
```

```
297         private ServiceWithInterleave(DispatcherQueue taskQueue)
298         {
299             // create PortSet instance used by external callers to post items
300             _mainPort = new ServicePort();
301 
302             // cache dispatcher queue used to schedule tasks
303             _taskQueue = taskQueue;
304         }
305 
306         void GetStateHandler(GetState get)
307         {
308             if (_state == null)
309             {
310                 Console.WriteLine("_state == null");
311                 // when state is null will post an exception
312                 get.ResponsePort.Post(new InvalidOperationException());
313                 return;
314             }
315 
316             // return the state as a message on the response port
317             get.ResponsePort.Post(_state);
318         }
319 
320         void UpdateHandler(UpdateState update)
321         {
322             // update state from field in the message
323             // Because the update requires a read, a merge of two strings
324             // and an update, this code needs to run un-interrupted by other updates.
325             // The Interleave Arbiter makes this guarantee since the UpdateHandler is in the
326             // ExclusiveReceiverGroup
327             _state = update.State + _state;
328             //_state = null;
329             // as success result, post the state itself
330             update.ResponsePort.Post(_state);
331             Console.WriteLine("update.ResponsePort.Post: " + _state);
332         }
333 
334         void StopHandler(Stop stop)
335         {
336             Console.WriteLine("Service stopping. No other handlers are running or will run after this");
337         }
338     }
339 
340 }
```

Example 20: Nesting of iterators, Yielding to another iterator

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

namespace Example
{
    class Program
    {
        //Example 20: Nesting of iterators, Yielding to another iterator
        static void Main(string[] args)
        {
            // create an IterativeTask instance and schedule it
            Arbiter.Activate(_taskQueue,
                Arbiter.FromIteratorHandler(ParentIteratorMethod)
            );
        }

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

        static IEnumerator<ITask> ParentIteratorMethod()
        {
            Console.WriteLine("Yielding to another iterator that will execute N asynchronous steps");
// 多核執行
            yield return new IterativeTask<int>(10, ChildIteratorMethod);
            Console.WriteLine("Child iterator completed");
        }

        static IEnumerator<ITask> ChildIteratorMethod(int count)
        {
            Port<int> portInt = new Port<int>();
            for (int i = 0; i < count; i++)
            {
                portInt.Post(i);
                // short form of receive that leaves item in port
                yield return portInt.Receive();
                // implicit operator extracts item from port
                int result = portInt;
                //Console.WriteLine(result);  // 破壞多核執行
            }
        }
    }
}
```