1 SPSafelet

```
1  /** Spacecraft - Mode Change Example
2   *
3   * This safelet is the top level of the application and loads the main mission
        sequencer
4   *
5   *    @author Matt Luckcuck <ml881@york.ac.uk>
6   */
7  package scjlevel2examples.spacecraft;
8
9  import javax.realtime.PriorityParameters;
10 import javax.safetycritical.Mission;
11 import javax.safetycritical.MissionSequencer;
12 import javax.safetycritical.Safelet;
13 import javax.safetycritical.StorageParameters;
14 import javax.scj.util.Const;
15
16 import devices.Console;
17
18 public class SPSafelet implements Safelet<Mission>
19 {
20
21 //  public static StorageParameters storageParametersSchedulable;
22   public static StorageParameters storageParameters_topLevelSequencer;
23   public static StorageParameters storageParameters_nestedSequencer;
24   public static StorageParameters storageParameters_Schedulable;
25
26
27   @Override
28   public MissionSequencer<Mission> getSequencer()
29   {
30     //TODO FIx Memory Parameters
31      storageParameters_topLevelSequencer =
32          new StorageParameters(
33              Const.OUTERMOST_SEQ_BACKING_STORE_DEFAULT ,
34              new long[] { Const.HANDLER_STACK_SIZE },
35              Const.PRIVATE_MEM ,
36              10000*2,
37              Const.MISSION_MEM);
38
39
40      storageParameters_nestedSequencer =
41          new StorageParameters(
42              Const.OVERALL_BACKING_STORE- Const.OUTERMOST_SEQ_BACKING_STORE_DEFAULT
                    ,
43              new long[] { Const.HANDLER_STACK_SIZE },
44              Const.PRIVATE_MEM,
45              10000*2,
46              Const.MISSION_MEM);
47
48      storageParameters_Schedulable =
49          new StorageParameters(
50              Const.PRIVATE_BACKING_STORE_DEFAULT,
51              new long[] { Const.HANDLER_STACK_SIZE },
52              Const.PRIVATE_MEM ,
53              10000,
54              Const.MISSION_MEM);
55
56     return new MainMissionSequencer(new PriorityParameters(5),
57         storageParameters_topLevelSequencer);
58   }
59
60   @Override
61   public long immortalMemorySize()
62   {
63     return Const.IMMORTAL_MEM_DEFAULT;
64   }
65
66   @Override
```

```
67    public void initializeApplication()
68    {
69      Console.println("SPSafelet: Init");
70
71      //Apparently this is never called.
72
73       Console.println("SPSafelet: Begin");
74    }
75
76 }
```

2 MainMissionSequencer

```
1  /** Spacecraft - Mode Change Example
2   *
3   *  The main mission sequencer for the application
4   *
5   *   @author Matt Luckcuck <ml881@york.ac.uk>
6   */
7  package scjlevel2examples.spacecraft;
8
9  import javax.realtime.PriorityParameters;
10 import javax.safetycritical.Mission;
11 import javax.safetycritical.MissionSequencer;
12 import javax.safetycritical.StorageParameters;
13
14 import devices.Console;
15
16 public class MainMissionSequencer extends MissionSequencer<Mission>
17 {
18
19   /**
20    * Has this single mission been returned?
21    */
22   private boolean returnedMission;
23
24   /**
25    * Class Constructor
26    *
27    * @param pp
28    *             the PriorityParameters for the sequencer
29    * @param sp
30    *             the StorageParameters for the sequencer
31    */
32   public MainMissionSequencer(PriorityParameters pp, StorageParameters sp)
33   {
34     super(pp, sp);
35     Console.println("MainMissioNSequencer: constructor");
36     returnedMission = false;
37   }
38
39   /**
40    * Returns the new mission
41    */
42   @Override
43   protected Mission getNextMission()
44   {
45     Console.println("MainMissioNSequencer: getNextMission");
46     // This returns the main mission once only
47     if (!returnedMission)
48     {
49       returnedMission = true;
50       return new MainMission();
51     } else
52     {
53       return null;
54     }
55   }
56
57 }
```

3 MainMission

```
1  /** Spacecraft - Mode Change Example
2   *
3   * This is the main mission, it represents the Spacecraft.
4   * It loads the persistent handlers and the sequencer for the modes.
5   *
6   *    @author Matt Luckcuck <ml881@york.ac.uk>
7   */
8  package scjlevel2examples.spacecraft;
9
10 import javax.realtime.AperiodicParameters;
11 import javax.realtime.PeriodicParameters;
12 import javax.realtime.PriorityParameters;
13 import javax.realtime.RelativeTime;
14 import javax.safetycritical.Mission;
15 import javax.scj.util.Const;
16
17 import devices.Console;
18
19 public class MainMission extends Mission
20 {
21   /**
22    * Initilises the Mission, loading the ModeChanger and the persistent
23    * handlers
24    */
25   @Override
26   protected void initialize()
27   {
28     Console.println("Main Mission: Init ");
29
30     // Load the nested mission sequencer and persistent handlers
31     SPModeChanger sPModeChanger = new SPModeChanger(new PriorityParameters(
32         5), SPSafelet.storageParameters_nestedSequencer, this);
33
34     sPModeChanger.register();
35
36     EnvironmentMonitor environmentMonitor = new EnvironmentMonitor(
37         new PriorityParameters(5), new PeriodicParameters(
38             new RelativeTime(0, 0), new RelativeTime(2000, 0)),
39         SPSafelet.storageParameters_Schedulable, this);
40     environmentMonitor.register();
41
42     ControlHandler controlHandler = new ControlHandler(
43         new PriorityParameters(5), new AperiodicParameters(),
44         SPSafelet.storageParameters_Schedulable);
45     controlHandler.register();
46
47     AperiodicSimulator controlSim = new AperiodicSimulator(
48         new PriorityParameters(5), new PeriodicParameters(
49             new RelativeTime(0, 0), new RelativeTime(1000, 0)),
50         SPSafelet.storageParameters_Schedulable, controlHandler);
51     controlSim.register();
52
53     Console.println("Main Mission: Begin ");
54   }
55
56   /**
57    * Returns the required size of this Mission's private memory
58    */
59   @Override
60   public long missionMemorySize()
61   {
62     return Const.MISSION_MEM_DEFAULT;
63   }
64
65   public void environmentBad()
66   {
67     // This would cause the system to check and attempt to remedy the bad
68     // internal environment
```

```
69          Console.println("Envrionment Bad");
70
71      }
72  }
```

3.1 Schedulables of MainMission

3.2 EnvironmentMonitor

```
1  /** Spacecraft - Mode Change Example
2   *
3   *  This class monitors the craft's environment -- Oxygen levels, internal pressure
        , fuel levels etc.
4   *
5   *   @author Matt Luckcuck <ml881@york.ac.uk>
6   */
7  package scjlevel2examples.spacecraft;
8
9  import javax.realtime.PeriodicParameters;
10 import javax.realtime.PriorityParameters;
11 import javax.safetycritical.PeriodicEventHandler;
12 import javax.safetycritical.StorageParameters;
13
14 import devices.Console;
15
16 public class EnvironmentMonitor extends PeriodicEventHandler
17 {
18
19   private final MainMission mainMission;
20
21   /**
22    * Class Constructor
23    *
24    * @param priority
25    *             priority parameters
26    * @param periodic
27    *             periodic parameters
28    * @param storage
29    *             storage parameters
30    * @param size
31    *             private memory size
32    */
33   public EnvironmentMonitor(PriorityParameters priority,
34       PeriodicParameters periodic, StorageParameters storage,
35       MainMission mainMission)
36   {
37     super(priority, periodic, storage);
38     this.mainMission = mainMission;
39   }
40
41   /**
42    * Called when the handler is fired
43    */
44   @SuppressWarnings("unused")
45   @Override
46   public void handleAsyncEvent()
47   {
48     Console.println("Checking Environment");
49
50     // **Obviously these is for testing purposes only toggle the true and
51     // false values to test behaviour
52
53     // if environment conditions are bad
54     // if(true)
55
56     // if environment conditions are fine
57     if (false)
58     {
59       // To get here the environment conditions should be below safe
60       // levels
61
62       mainMission.environmentBad();
63     }
64   }
65
```

```
66 }
```

3.3 ControlHandler

```
1  /** Spacecraft - Mode Change Example
2
3   *    Handler for the craft's controls
4
5   *    @author Matt Luckcuck <ml881@york.ac.uk>
6   */
7  package scjlevel2examples.spacecraft;
8
9  import javax.realtime.AperiodicParameters;
10 import javax.realtime.PriorityParameters;
11 import javax.safetycritical.AperiodicEventHandler;
12 import javax.safetycritical.StorageParameters;
13
14 import devices.Console;
15
16 public class ControlHandler extends AperiodicEventHandler
17 {
18   /**
19    * Class Constructor
20    *
21    * @param priorityParameters
22    *             the priority parameters for this handler
23    * @param periodicParameters
24    *             the periodic parameters for this handler
25    * @param storageConfigurationParameters
26    *             the storage parameters for this handler
27    * @param size
28    *             the size of the private memory for this handler
29    */
30   public ControlHandler(PriorityParameters priority,
31       AperiodicParameters release, StorageParameters storage)
32   {
33 //    super(priority, release, storage, "Control Handler");
34     super(priority, release, storage);
35   }
36
37   /**
38    * Called when the handler is fired
39    */
40   @Override
41   public void handleAsyncEvent()
42   {
43     Console.println("Handling Controls");
44     // Actually handle input from the controls
45   }
46 }
```

3.4 EnvironmentMonitor

```
1  /** Spacecraft - Mode Change Example
2   *
3   *  This class monitors the craft's environment -- Oxygen levels, internal pressure
        , fuel levels etc.
4   *
5   *  @author Matt Luckcuck <ml881@york.ac.uk>
6   */
7  package scjlevel2examples.spacecraft;
8
9  import javax.realtime.PeriodicParameters;
10 import javax.realtime.PriorityParameters;
11 import javax.safetycritical.PeriodicEventHandler;
12 import javax.safetycritical.StorageParameters;
13
14 import devices.Console;
15
16 public class EnvironmentMonitor extends PeriodicEventHandler
17 {
18
19   private final MainMission mainMission;
20
21   /**
22    * Class Constructor
23    *
24    * @param priority
25    *             priority parameters
26    * @param periodic
27    *             periodic parameters
28    * @param storage
29    *             storage parameters
30    * @param size
31    *             private memory size
32    */
33   public EnvironmentMonitor(PriorityParameters priority,
34       PeriodicParameters periodic, StorageParameters storage,
35       MainMission mainMission)
36   {
37     super(priority, periodic, storage);
38     this.mainMission = mainMission;
39   }
40
41   /**
42    * Called when the handler is fired
43    */
44   @SuppressWarnings("unused")
45   @Override
46   public void handleAsyncEvent()
47   {
48     Console.println("Checking Environment");
49
50     // **Obviously these is for testing purposes only toggle the true and
51     // false values to test behaviour
52
53     // if environment conditions are bad
54     // if(true)
55
56     // if environment conditions are fine
57     if (false)
58     {
59       // To get here the environment conditions should be below safe
60       // levels
61
62       mainMission.environmentBad();
63     }
64   }
65
66 }
```

3.5 AperiodicSimulator

```
1  /** Spacecraft - Mode Change Example
2   *
3   * This class is not a component of the pattern or the example application,
4   * it is plumbing only.
5   *
6   * This class simulates the aperiodic firing of an
7   * external event (e.g. a button press) by simply firing the event periodically
8   *
9   * @author Matt Luckcuck <ml881@york.ac.uk>
10  */
11 package scjlevel2examples.spacecraft;
12
13 import javax.realtime.PeriodicParameters;
14 import javax.realtime.PriorityParameters;
15 import javax.safetycritical.AperiodicEventHandler;
16 import javax.safetycritical.PeriodicEventHandler;
17 import javax.safetycritical.StorageParameters;
18
19 import devices.Console;
20
21 public class AperiodicSimulator extends PeriodicEventHandler
22 {
23
24   AperiodicEventHandler aperiodic;
25
26   /**
27    * Class constructor
28    *
29    * @param priority
30    *             the priority of the handler
31    * @param periodic
32    *             the periodic parameters of the handler
33    * @param storage
34    *             the storage parameters of the handler
35    * @param size
36    *             the size of the private memory of the handler
37    * @param aperiodicEvent
38    *             the aperiodic even to be fires each period
39    */
40   public AperiodicSimulator(PriorityParameters priority,
41       PeriodicParameters periodic, StorageParameters storage,
42       AperiodicEventHandler aperiodicEvent)
43   {
44     super(priority, periodic, storage);
45     aperiodic = aperiodicEvent;
46   }
47
48   /**
49    * The method the infrastructure calls when it is fired
50    *
51    * This method fires the <code>event</code>
52    */
53   @Override
54   public void handleAsyncEvent()
55   {
56     Console.println("Simulating AperiodicEvent: "+ aperiodic.toString());
57     aperiodic.release();
58   }
59
60 }
```

3.6 SPModeChanger

```
1  /** Spacecraft - Mode Change Example
2   *
3   *  This is the mode changer for the Spacecraft application,
4   *  it controls which mode the application is in
5   *
6   *   @author Matt Luckcuck <ml881@york.ac.uk>
7   */
8  package scjlevel2examples.spacecraft;
9
10 import javax.realtime.PriorityParameters;
11 import javax.safetycritical.Mission;
12 import javax.safetycritical.MissionSequencer;
13 import javax.safetycritical.StorageParameters;
14
15 import devices.Console;
16
17 public class SPModeChanger extends MissionSequencer<Mission> implements
18     ModeChanger
19 {
20   /**
21    * This variable represents the number of modes this ModeChanger has to deal
22    * with
23    */
24   private int modesLeft = 3;
25   /**
26    * A reference to a mode
27    */
28   private Mode currentMode, launchMode, cruiseMode, landMode;
29
30   /**
31    * The controlling mission
32    */
33   private MainMission controllingMission;
34
35   /**
36    * Class constructor
37    *
38    * @param priority
39    *            the priority parameters for this mission sequencer
40    * @param storage
41    *            the storage parameters for this mission sequencer
42    */
43   public SPModeChanger(PriorityParameters priority, StorageParameters storage,
       MainMission controllingMission)
44   {
45     super(priority, storage);
46     Console.println("Mode Changer: Construct ");
47
48     launchMode = new LaunchMission();
49     cruiseMode = new CruiseMission();
50     landMode = new LandMission();
51
52     this.controllingMission = controllingMission;
53   }
54
55   /**
56    * Change the mode to given mode
57    */
58   @Override
59   public synchronized void changeTo(Mode newMode)
60   {
61     currentMode = newMode;
62
63   }
64
65   /**
66    * Advance the mode to the next mode
67    */
```

```
68    @Override
69    public synchronized void advanceMode()
70    {
71      Console.println("Mode Changer: Advance To Next Mode");
72      // check the value of the modes variable and changeTo the associated
73      // mode
74      // once all the missions have been run, changeTo null to terminate the
75      // sequencer
76      if (modesLeft == 3)
77      {
78        modesLeft--;
79        Console.println("Mode Changer: Advance To Launch Mode");
80        changeTo(launchMode);
81      }
82      else if (modesLeft == 2)
83      {
84        modesLeft--;
85        Console.println("Mode Changer: Advance To Cruise Mode");
86        changeTo(cruiseMode);
87      }
88      else if (modesLeft == 1)
89      {
90        modesLeft--;
91        Console.println("Mode Changer: Advance To Land Mode");
92        changeTo(landMode);
93      }
94      else
95      {
96        changeTo(null);
97        Console.println("Mode Changer: FINISHED");
98        controllingMission.requestTermination();
99      }
100   }
101
102   /**
103    * return the <code>currentMode</code> which has been set by either
104    * <code>advanceMode</code> or <code>changeTo</code>
105    */
106   @Override
107   protected Mission getNextMission()
108   {
109     Console.println("Mode Changer: getNextMission");
110     if (modesLeft == 3)
111     {
112       modesLeft--;
113       Console.println("Mode Changer: Advance To Launch Mode");
114       return (Mission) launchMode;
115     }
116     else if (modesLeft == 2)
117     {
118       modesLeft--;
119       Console.println("Mode Changer: Advance To Cruise Mode");
120       return (Mission) cruiseMode;
121     }
122     else if (modesLeft == 1)
123     {
124       modesLeft--;
125       Console.println("Mode Changer: Advance To Land Mode");
126       return (Mission) landMode;
127     }
128     else
129     {
130       Console.println("Mode Changer: FINISHED");
131       controllingMission.requestTermination();
132       return null;
133
134     }
135   }
136
137 }
```

4 LaunchMission

```
1  /** Spacecraft - Mode Change Example
2   *
3   *    This mission deals with launching the craft
4   *
5   *    @author Matt Luckcuck <ml881@york.ac.uk>
6   */
7  package scjlevel2examples.spacecraft;
8
9  import javax.realtime.AperiodicParameters;
10 import javax.realtime.PeriodicParameters;
11 import javax.realtime.PriorityParameters;
12 import javax.realtime.RelativeTime;
13 import javax.safetycritical.Mission;
14 import javax.scj.util.Const;
15
16 import devices.Console;
17
18 public class LaunchMission extends Mission implements Mode
19 {
20   /**
21    * This variable represents if the craft is able to launch, when the
22    * countdown reaches 0
23    */
24   private volatile boolean launch = true;
25
26   /**
27    * Called when the craft is ok to launch sets <code>launch</code> to
28    * <code>true</code>
29    */
30   public void goodToLaunch()
31   {
32     launch = true;
33   }
34
35   /**
36    * Returns the <code>launch</code> variable
37    *
38    * @return <code>launch</code>
39    */
40   public boolean canLaunch()
41   {
42     return launch;
43   }
44
45   /**
46    * initialises the mission
47    */
48   @Override
49   protected void initialize()
50   {
51     Console.println("Launch Mission: Init ");
52
53     // Initially false because the conditions haven't been checked yet
54
55     launch = true;
56
57     // Load the handlers for this mission
58     // Note these handlers are passed a reference to this mission so they
59     // can update the
60     // ready to launch variable with the two methods above
61
62 //    LaunchConditionsMonitor launchConditionsMonitor = new LaunchConditionsMonitor
       (
63 //        new PriorityParameters(5), new PeriodicParameters(
64 //            new RelativeTime(0, 0), new RelativeTime(500, 0)),
65 //        SPSafelet.storageParameters_Schedulable, this);
66 //    launchConditionsMonitor.register();
67
```

```
68      LaunchHandler launchHandler = new LaunchHandler(new PriorityParameters(
69          5), new AperiodicParameters(),
70          SPSafelet.storageParameters_Schedulable, this);
71      launchHandler.register();
72
73      LaunchCountdown launchCountdown = new LaunchCountdown(
74          new PriorityParameters(5), new PeriodicParameters(
75              new RelativeTime(0, 0), new RelativeTime(1000, 0)),
76          SPSafelet.storageParameters_Schedulable, 5, launchHandler);
77      launchCountdown.register();
78
79      Console.println("Launch Mission: Begin ");
80    }
81
82    /**
83     * Returns the size of the mission's memory
84     */
85    @Override
86    public long missionMemorySize()
87    {
88      return Const.MISSION_MEM_DEFAULT;
89    }
90  }
```

4.1 Schedulables of LaunchMission

4.2 LaunchHandler

```
1  /** Spacecraft - Mode Change Example
2   *
3   *  Handler for launching the craft
4   *
5   *   @author Matt Luckcuck <ml881@york.ac.uk>
6   */
7  package scjlevel2examples.spacecraft;
8
9  import javax.realtime.AperiodicParameters;
10 import javax.realtime.PriorityParameters;
11 import javax.safetycritical.AperiodicEventHandler;
12 import javax.safetycritical.StorageParameters;
13
14 import devices.Console;
15
16 public class LaunchHandler extends AperiodicEventHandler
17 {
18   /**
19    * The controlling mission
20    */
21   private final LaunchMission launchMission;
22
23   /**
24    * Class Constructor
25    *
26    * @param priorityParameters
27    *             the priority parameters for this handler
28    * @param periodicParameters
29    *             the periodic parameters for this handler
30    * @param storageConfigurationParameters
31    *             the storage parameters for this handler
32    * @param size
33    *             the size of the private memory for this handler
34    * @param launchMission
35    *             the controlling mission
36    */
37   public LaunchHandler(PriorityParameters priority,
38       AperiodicParameters release, StorageParameters storage,
39       LaunchMission launchMission)
40   {
41     // super(priority, release, storage);
42 //    super(priority, release, storage, "Laucnh Handler");
43     super(priority, release, storage);
44
45     Console.println("LaunchHandler: Construct");
46     this.launchMission = launchMission;
47   }
48
49   /**
50    * Called when the handler is fired Launches the craft
51    */
52   @Override
53   public void handleAsyncEvent()
54   { // if the launch mission says that the launch can go ahead then the craft
55     // launches
56     // for testing, here the launch mission is just terminated
57     // else the handler waits for it's next release and checks again
58     Console.println("LaunchHandler: LaunchHandler");
59
60     if (launchMission.canLaunch())
61     {
62       Console.println("LaunchHandler: Launching!");
63
64       launchMission.requestTermination();
65     } else
66     {
```

```
67            Console.println("LaunchHandler: Launch Blocked!");
68        }
69    }
70 }
```

4.3 LaunchCountdown

```
1  /** Spacecraft - Mode Change Example
2   *
3   *  This handler counts down from a given value to zero
4   *
5   *   @author Matt Luckcuck <ml881@york.ac.uk>
6   */
7  package scjlevel2examples.spacecraft;
8
9  import javax.realtime.PeriodicParameters;
10 import javax.realtime.PriorityParameters;
11 import javax.safetycritical.AperiodicEventHandler;
12 import javax.safetycritical.PeriodicEventHandler;
13 import javax.safetycritical.StorageParameters;
14
15 import devices.Console;
16
17 public class LaunchCountdown extends PeriodicEventHandler
18 {
19   /**
20    * The Aperiodic Event Handler to be released
21    */
22   private final AperiodicEventHandler aperiodic;
23
24   /**
25    * The starting value of the countdown
26    */
27   private int countdown;
28
29   /**
30    * Class Constructor
31    *
32    * @param priority
33    *             priority parameters
34    * @param periodic
35    *             periodic parameters
36    * @param storage
37    *             storage parameters
38    * @param size
39    *             private memory size
40    * @param countdown
41    *             starting value of countdown
42    * @param ae
43    *             the event to be fired
44    */
45   public LaunchCountdown(PriorityParameters priority,
46       PeriodicParameters periodic, StorageParameters storage,
47       int countdown, AperiodicEventHandler ae)
48   {
49     super(priority, periodic, storage);
50     Console.println("LaunchCountdown: Construct");
51
52     aperiodic = ae;
53     this.countdown = countdown;
54   }
55
56   /**
57    * Called when this event is fired, if the countdown is 0 then fire the
58    * launch event
59    */
60   @Override
61   public void handleAsyncEvent()
62   {
63     Console.println("***LaunchCountdown***");
64     // if the count has reached 0 then fire the launch event
65     // else decrement the count
66     if (countdown == 0)
67     {
68       Console.println("" + countdown);
```

```
69          Console.println("Launching");
70          aperiodic.release();
71        } else
72        {
73          Console.println("" + countdown);
74          countdown--;
75        }
76    }
77
78 }
```

5 CruiseMission

```
1  /** Spacecraft - Mode Change Example
2   *
3   * This mission handles events when the craft is cruising -- not launching,
4   * orbiting, or landing.
5   *
6   *    @author Matt Luckcuck <ml881@york.ac.uk>
7   */
8  package scjlevel2examples.spacecraft;
9
10 import javax.realtime.AperiodicParameters;
11
12 import javax.realtime.PeriodicParameters;
13 import javax.realtime.PriorityParameters;
14 import javax.realtime.RelativeTime;
15 import javax.safetycritical.Mission;
16
17 import javax.scj.util.Const;
18
19 import devices.Console;
20
21 public class CruiseMission extends Mission implements Mode
22 {
23   /**
24    * Boolean representing if it is safe to burn the engines
25    */
26   private boolean okToCruise = true;
27
28   /**
29    * Desired duration of the burn
30    */
31   private RelativeTime burnDuration;
32
33   @Override
34   protected void initialize()
35   {
36
37     Console.println("Cruise Mission: Init ");
38
39     /**
40      * Then length of time to burn the engines for
41      */
42     burnDuration = new RelativeTime();
43
44     /**
45      * Handler for monitoring the cruising conditions and updating
46      * <code>okToCruise</code>
47      */
48     CruiseConditionsMonitor crusieConditionsMonitor = new CruiseConditionsMonitor(
49         new PriorityParameters(5), new PeriodicParameters(
50             new RelativeTime(0, 0), new RelativeTime(500, 0)),
51         SPSafelet.storageParameters_Schedulable, this);
52     crusieConditionsMonitor.register();
53
54     /**
55      * Handler for responding to the burn being activated
56      */
57     BurnActivationHandler burnActivationHandler = new BurnActivationHandler(
58         new PriorityParameters(5), new AperiodicParameters(
59             new RelativeTime(0, 0), null),
60         SPSafelet.storageParameters_Schedulable, this);
61     burnActivationHandler.register();
62
63     /**
64      * Handler for activating the engine burn when requested
65      */
66     BurnDurationHandler burnDurationHandler = new BurnDurationHandler(
67         new PriorityParameters(5), new AperiodicParameters(
68             new RelativeTime(0, 0), null),
```

```
69            SPSafelet.storageParameters_Schedulable, this);
70        burnDurationHandler.register();
71
72        /**
73         * Handler simulating a button push to activate the burn
74         */
75        AperiodicSimulator cruiseSim = new AperiodicSimulator(
76            new PriorityParameters(5), new PeriodicParameters(
77                new RelativeTime(0, 0), new RelativeTime(2000, 0)),
78            SPSafelet.storageParameters_Schedulable, burnActivationHandler);
79        cruiseSim.register();
80
81        Console.println("Cruise Mission: Begin ");
82    }
83
84    /**
85     * returns the mission's private memory size
86     */
87    @Override
88    public long missionMemorySize()
89    {
90        return Const.MISSION_MEM_DEFAULT;
91    }
92
93    /**
94     * returns <code> oKToCruise</code>
95     *
96     * @return true if it is ok to activate the burn, false if it is not
97     */
98    public boolean isOkToCruise()
99    {
100       return okToCruise;
101   }
102
103   /**
104    * Sets <code>okToCruise</code>
105    *
106    * @param okToCruise
107    *            new boolean value for <code>okToCruise</code>
108    */
109   public void setOkToCruise(boolean okToCruise)
110   {
111       this.okToCruise = okToCruise;
112   }
113
114   /**
115    * Sets the duration of the burn
116    *
117    * @param millis
118    *            burn duration millisecond part
119    * @param nanos
120    *            burn duration nanosecond part
121    */
122   public synchronized void setBurnDuration(long millis, int nanos)
123   {
124       burnDuration.set(millis, nanos);
125   }
126
127   /**
128    * activates the engine burn
129    */
130   public void activateBurn()
131   {
132       Console.println("Burning Engines!");
133       // actually activate the engines here
134   }
135 }
```

5.1 Schedulables of CruiseMission

5.2 BurnActivationHandler

```
1  /** Spacecraft - Mode Change Example
2   *
3   * Handler for responding to the pilot starting an engine burn
4   *
5   *    @author Matt Luckcuck <ml881@york.ac.uk>
6   */
7  package scjlevel2examples.spacecraft;
8
9  import javax.realtime.AperiodicParameters;
10 import javax.realtime.PriorityParameters;
11 import javax.safetycritical.AperiodicEventHandler;
12 import javax.safetycritical.StorageParameters;
13
14 import devices.Console;
15
16 public class BurnActivationHandler extends AperiodicEventHandler
17 {
18   /**
19    * The controlling mission
20    */
21   private final CruiseMission mission;
22
23   /**
24    * Class Constructor
25    *
26    * @param priority
27    *             PriportiyParamters for this handler
28    * @param release
29    *             ReleaseParameters for this handler
30    * @param storage
31    *             StorageParamters for this handler
32    * @param size
33    *             size if this handler's private memory
34    * @param mission
35    *             this handler's controlling mission
36    */
37   public BurnActivationHandler(PriorityParameters priority,
38       AperiodicParameters release, StorageParameters storage,
39       CruiseMission mission)
40   {
41 //    super(priority, release, storage, "Burn Activation Handler");
42     super(priority, release, storage);
43     this.mission = mission;
44   }
45
46   /**
47    * Called when the handler is fired
48    *
49    * Checks with the mission to see if it is ok to activate the burn.
50    * Activates if <code>mission.isOkToCruise()</code> returns
51    * <code>true</code>
52    */
53   @Override
54   public void handleAsyncEvent()
55   {
56     if (mission.isOkToCruise())
57     {
58       Console.println("Activate Burn");
59       mission.activateBurn();
60
61     } else
62     {
63       Console.println("Burn Blocked");
64     }
65
66   }
```

```
67
68 }
```

5.3 BurnDurationHandler

```
1  /** Spacecraft - Mode Change Example
2   *
3   * Handler that sets the required burn duration based on user input (simulated in
        this example)
4   *
5   *    @author Matt Luckcuck <ml881@york.ac.uk>
6   */
7  package scjlevel2examples.spacecraft;
8
9  import javax.realtime.AperiodicParameters;
10 import javax.realtime.PriorityParameters;
11 import javax.safetycritical.AperiodicEventHandler;
12 import javax.safetycritical.StorageParameters;
13
14 import devices.Console;
15
16 public class BurnDurationHandler extends AperiodicEventHandler
17 {
18   /**
19    * This handler's controlling mission
20    */
21   private final CruiseMission mission;
22   /**
23    * The burn duration's nanosecond component
24    */
25   private int nanos;
26   /**
27    * the burn duration's millisecond component
28    */
29   private long millis;
30
31   /**
32    * Class Constructor
33    *
34    * @param priorityParameters
35    *             the priority parameters for this handler
36    * @param periodicParameters
37    *             the periodic parameters for this handler
38    * @param storageConfigurationParameters
39    *             the storage parameters for this handler
40    * @param size
41    *             the size of the private memory for this handler
42    * @param mission
43    *             this handler's controlling mission
44    */
45   public BurnDurationHandler(PriorityParameters priority,
46       AperiodicParameters aperiodic, StorageParameters storage,
47       CruiseMission mission)
48   {
49 //     super(priority, aperiodic, storage, "Burn Duration Handler");
50     super(priority, aperiodic, storage);
51     this.mission = mission;
52     nanos = 0;
53     millis = 0;
54   }
55
56   /**
57    * Sets the desired duration of the burn
58    *
59    * @param millis
60    *             burn duration, millisecond part
61    * @param nanos
62    *             burn duration, nanosecond part
63    */
64   public void setBurnDuration(int millis, int nanos)
65   {
66     this.millis = millis;
67     this.nanos = nanos;
```

```
68      }
69
70      /**
71       * This method is called when the handler is fired.
72       *
73       * sets the <code>burnDuration</code> in the controlling mission to the
74       * values stored in this handler
75       */
76      @Override
77      public void handleAsyncEvent()
78      {
79        Console.println("Burn Duration Handler");
80        mission.setBurnDuration(millis, nanos);
81      }
82  }
```

5.4 CruiseConditionsMonitor

```
1  /** Spacecraft - Mode Change Example
2   *
3   *    @author Matt Luckcuck <ml881@york.ac.uk>
4   */
5  package scjlevel2examples.spacecraft;
6
7  import javax.realtime.PeriodicParameters;
8  import javax.realtime.PriorityParameters;
9  import javax.safetycritical.PeriodicEventHandler;
10 import javax.safetycritical.StorageParameters;
11
12 import devices.Console;
13
14 /**
15  * Handler for monitoring the conditions which have to be true for the craft to
16  * begin cruising
17  *
18  * @author Matt Luckcuck
19  *
20  */
21 public class CruiseConditionsMonitor extends PeriodicEventHandler
22 {
23   /**
24    * The controlling mission
25    */
26   private final CruiseMission mission;
27
28   /**
29    * The count of times this handler will be released before terminating the
30    * controlling mission
31    */
32   private int count = 10;
33
34   public CruiseConditionsMonitor(PriorityParameters priority,
35       PeriodicParameters periodic, StorageParameters storage,
36       CruiseMission mission)
37   {
38     super(priority, periodic, storage);
39     this.mission = mission;
40   }
41
42   /**
43    * Called when the handler is fired
44    */
45   @Override
46   public void handleAsyncEvent()
47   {
48     Console.println("Checking Cruise Conditions");
49     // Check sensors to make sure an engine burn is safe
50
51     if (count == 0)
52     {
53       Console.println("CruiseConditionsMonitor: Terminating");
54       mission.requestTermination();
55     } else
56     {
57       Console.println("CruiseConditionsMonitor: " + count);
58       mission.setOkToCruise(true);
59       count--;
60     }
61   }
62 }
```

5.5 AperiodicSimulator

```
1  /** Spacecraft - Mode Change Example
2   *
3   * This class is not a component of the pattern or the example application,
4   * it is plumbing only.
5   *
6   * This class simulates the aperiodic firing of an
7   * external event (e.g. a button press) by simply firing the event periodically
8   *
9   * @author Matt Luckcuck <ml881@york.ac.uk>
10  */
11 package scjlevel2examples.spacecraft;
12
13 import javax.realtime.PeriodicParameters;
14 import javax.realtime.PriorityParameters;
15 import javax.safetycritical.AperiodicEventHandler;
16 import javax.safetycritical.PeriodicEventHandler;
17 import javax.safetycritical.StorageParameters;
18
19 import devices.Console;
20
21 public class AperiodicSimulator extends PeriodicEventHandler
22 {
23
24   AperiodicEventHandler aperiodic;
25
26   /**
27    * Class constructor
28    *
29    * @param priority
30    *             the priority of the handler
31    * @param periodic
32    *             the periodic parameters of the handler
33    * @param storage
34    *             the storage parameters of the handler
35    * @param size
36    *             the size of the private memory of the handler
37    * @param aperiodicEvent
38    *             the aperiodic even to be fires each period
39    */
40   public AperiodicSimulator(PriorityParameters priority,
41       PeriodicParameters periodic, StorageParameters storage,
42       AperiodicEventHandler aperiodicEvent)
43   {
44     super(priority, periodic, storage);
45     aperiodic = aperiodicEvent;
46   }
47
48   /**
49    * The method the infrastructure calls when it is fired
50    *
51    * This method fires the <code>event</code>
52    */
53   @Override
54   public void handleAsyncEvent()
55   {
56     Console.println("Simulating AperiodicEvent: "+ aperiodic.toString());
57     aperiodic.release();
58   }
59
60 }
```

6 LandMission

```
1  /** Spacecraft - Mode Change Example
2   *
3   *  This mission handles events when the craft is landing
4   *
5   *   @author Matt Luckcuck <ml881@york.ac.uk>
6   */
7  package scjlevel2examples.spacecraft;
8
9  import javax.realtime.AperiodicParameters;
10 import javax.realtime.PeriodicParameters;
11 import javax.realtime.PriorityParameters;
12 import javax.realtime.RelativeTime;
13 import javax.safetycritical.Mission;
14 import javax.scj.util.Const;
15
16 import devices.Console;
17
18 public class LandMission extends Mission implements Mode
19 {
20
21
22   /**
23    * Initialise the mission
24    */
25   @Override
26   protected void initialize()
27   {
28     Console.println("Land Mission: Init ");
29
30     /* ***Start this mission's handlers */
31
32     AirSpeedMonitor airSpeedMonitor = new AirSpeedMonitor(
33         new PriorityParameters(5), new PeriodicParameters(
34             new RelativeTime(0, 0), new RelativeTime(500, 0)),
35         SPSafelet.storageParameters_Schedulable, this);
36     airSpeedMonitor.register();
37
38      LandingGearHandler landingHandler = new LandingGearHandler(
39      new PriorityParameters(5), new AperiodicParameters(new RelativeTime(0,0), null
            ),
40      SPSafelet.storageParameters_Schedulable, this);
41
42      landingHandler.register();
43
44      ParachuteHandler parachuteHandler = new ParachuteHandler(
45      new PriorityParameters(5), new AperiodicParameters(new RelativeTime(0,0), null
            ),
46      SPSafelet.storageParameters_Schedulable, this);
47
48      parachuteHandler.register();
49
50     GroundDistanceMonitor groundDistanceMonitor = new GroundDistanceMonitor(
51           new PriorityParameters(5), new PeriodicParameters(
52               new RelativeTime(0, 0), new RelativeTime(500, 0)),
53           SPSafelet.storageParameters_Schedulable, this, landingHandler,
                 parachuteHandler);
54     groundDistanceMonitor.register();
55
56
57     Console.println("Land Mission: Begin ");
58   }
59
60   /**
61    * Returns the size of this mission's memory
62    */
63   @Override
64   public long missionMemorySize()
65   {
```

```
66         return Const.MISSION_MEM_DEFAULT;
67     }
68
69
70
71 }
```

6.1 Schedulables of LandMission

6.2 LandingGearHandler

```
1  /** Spacecraft - Mode Change Example
2  *
3  *   Handler for dealing with the craft's landing gear
4  *
5  *   @author Matt Luckcuck <ml881@york.ac.uk>
6  */
7  package scjlevel2examples.spacecraft;
8
9  import javax.realtime.AperiodicParameters;
10 import javax.realtime.PriorityParameters;
11 import javax.safetycritical.AperiodicEventHandler;
12 import javax.safetycritical.StorageParameters;
13
14 import devices.Console;
15
16
17 public class LandingGearHandler extends AperiodicEventHandler
18 {
19   /**
20    * The controlling mission of this handler
21    */
22   private final LandMission mission;
23
24   /**
25    * Class Constructor
26    * @param priority the priority parameters for this handler
27    * @param release the release parameters for this handler
28    * @param storage the storage parameters for this handler
29    * @param size the private memory size of this handler
30    * @param landMission the controlling mission of this handler
31    */
32   public LandingGearHandler(PriorityParameters priority, AperiodicParameters
         release,
33       StorageParameters storage, LandMission landMission)
34   {
35 //    super(priority, release, storage, "Landing Gear Handler");
36     super(priority, release, storage);
37
38     mission = landMission;
39   }
40
41   /**
42    * Called when the handler is fired, deploys the landing gear
43    */
44   @Override
45   public void handleAsyncEvent()
46   {
47     Console.println("Deploying Landing Gear");
48
49 //    mission.deployLandingGear();
50   }
51 }
```

6.3 GroundDistanceMonitor

```
1  /** Spacecraft - Mode Change Example
2   *
3   *    @author Matt Luckcuck <ml881@york.ac.uk>
4   */
5  package scjlevel2examples.spacecraft;
6
7  import javax.realtime.PeriodicParameters;
8  import javax.realtime.PriorityParameters;
9  import javax.safetycritical.AperiodicEventHandler;
10 import javax.safetycritical.PeriodicEventHandler;
11 import javax.safetycritical.StorageParameters;
12
13 import devices.Console;
14
15 //Handler for monitoring the conditions which must be true for the craft to start
       landing
16 public class GroundDistanceMonitor extends PeriodicEventHandler
17 {
18   /**
19    * The controlling mission of this handler
20    */
21   private final LandMission mission;
22   private final AperiodicEventHandler landingHandler;
23   private final AperiodicEventHandler parachuteHandler;
24   /**
25    * Distance from the ground, dictates the amount of releases this handler will
          have before termination
26    */
27   private int groundDistance = 10;
28
29   /**
30    * Class Constructor
31    *
32    * @param priorityParameters
33    *             the priority parameters for this handler
34    * @param periodicParameters
35    *             the periodic parameters for this handler
36    * @param storageConfigurationParameters
37    *             the storage parameters for this handler
38    * @param size
39    *             the size of the private memory for this handler
40    * @param landMission
41    *             the controlling mission of this mission
42    */
43   public GroundDistanceMonitor(PriorityParameters priority,
44       PeriodicParameters periodic, StorageParameters storage,
45       LandMission landMission, AperiodicEventHandler landingHandler,
              AperiodicEventHandler parachuteHandler)
46   {
47     super(priority, periodic, storage);
48
49     mission = landMission;
50     this.landingHandler = landingHandler;
51     this.parachuteHandler = parachuteHandler;
52   }
53
54   /**
55    * Called when the handler is fired
56    */
57   @Override
58   public void handleAsyncEvent()
59   {
60     Console.println("GroundDistanceMonitor: Checking Ground Distance");
61     // read this value from sensors
62
63
64     if (groundDistance <= 0.0)
65     {
```

```
66          Console.println("GroundDistanceMonitor: Landed!");
67          mission.requestTermination();
68        }
69        else
70          if (groundDistance == 10)
71          {
72            landingHandler.release();
73            Console.println("GroundDistanceMonitor: ground distance is " +
                  groundDistance);
74            groundDistance = groundDistance - 2;
75          }
76          else if(groundDistance == 2)
77          {
78            parachuteHandler.release();
79            Console.println("GroundDistanceMonitor: ground distance is " +
                  groundDistance);
80            groundDistance = groundDistance - 2;
81          }
82          else
83
84          {
85            Console.println("GroundDistanceMonitor: ground distance is " +
                  groundDistance);
86            groundDistance = groundDistance - 2;
87          }
88    }
89  }
```

6.4 ParachuteHandler

```
1  /** Spacecraft - Mode Change Example
2   *
3   *  Handler that deploys a parachute to slow the craft on landing
4   *
5   *   @author Matt Luckcuck <ml881@york.ac.uk>
6   */
7  package scjlevel2examples.spacecraft;
8
9  import javax.realtime.AperiodicParameters;
10 import javax.realtime.PriorityParameters;
11 import javax.safetycritical.AperiodicEventHandler;
12 import javax.safetycritical.StorageParameters;
13
14 import devices.Console;
15
16 public class ParachuteHandler extends AperiodicEventHandler
17 {
18   /**
19    * The controlling mission
20    */
21   private final LandMission mission;
22
23   /**
24    * Class Constructor
25    *
26    * @param priorityParameters
27    *             the priority parameters for this handler
28    * @param periodicParameters
29    *             the periodic parameters for this handler
30    * @param storageConfigurationParameters
31    *             the storage parameters for this handler
32    * @param size
33    *             the size of the private memory for this handler
34    * @param landMission
35    *             the controlling mission
36    */
37   public ParachuteHandler(PriorityParameters priorityParameters,
38       AperiodicParameters aperiodicParameters,
39       StorageParameters storageParameters, LandMission landMission)
40   {
41 //    super(priorityParameters, aperiodicParameters, storageParameters,
42 //        "Parachute Handler");
43
44     super(priorityParameters, aperiodicParameters, storageParameters);
45
46     mission = landMission;
47   }
48
49   /**
50    * Called when the handler fires, deploys the parachute
51    */
52   @Override
53   public void handleAsyncEvent()
54   {
55     Console.println("Parachute Handler");
56 //    mission.deployParachute();
57   }
58 }
```

6.5 AirSpeedMonitor

```
1  /** Spacecraft - Mode Change Example
2   *
3   *    This Handler monitors the air speed of the Spacecraft
4   *
5   *   @author Matt Luckcuck <ml881@york.ac.uk>
6   */
7  package scjlevel2examples.spacecraft;
8
9  import javax.realtime.PeriodicParameters;
10 import javax.realtime.PriorityParameters;
11 import javax.safetycritical.PeriodicEventHandler;
12 import javax.safetycritical.StorageParameters;
13
14 import devices.Console;
15
16 public class AirSpeedMonitor extends PeriodicEventHandler
17 {
18   /**
19    * A reference to this handler's controlling mission
20    */
21   @SuppressWarnings("unused")
22   private final LandMission mission;
23
24   /**
25    * Class constructor
26    *
27    * @param priorityParameters
28    *            the priority parameters for this handler
29    * @param periodicParameters
30    *            the periodic parameters for this handler
31    * @param storageConfigurationParameters
32    *            the storage parameters for this handler
33    * @param size
34    *            the size of the private memory for this handler
35    * @param landMission
36    *            the controlling mission of this handler
37    *
38    */
39   public AirSpeedMonitor(PriorityParameters priorityParameters,
40       PeriodicParameters periodicParameters,
41       StorageParameters storageParameters, LandMission landMission)
42   {
43     super(priorityParameters, periodicParameters, storageParameters);
44
45     mission = landMission;
46   }
47
48   /**
49    * The method the infrastructure calls when the handler is released
50    */
51   @Override
52   public void handleAsyncEvent()
53   {
54     Console.println("AirSpeedMonitor: check Air Speed");
55     //Actually check the air speed sensor and update the main mission
56   }
57
58 }
```