

Espresso RNTL Project

Ludovic Gauthier, Marc Richard-Foy

2002/09/09

Contents

1	Package expresso	2
1.1	Interfaces	5
1.1.1	INTERFACE Schedulable	5
1.2	Classes	5
1.2.1	CLASS AbsoluteTime	5
1.2.2	CLASS AsyncEvent	11
1.2.3	CLASS AsyncEventHandler	12
1.2.4	CLASS BoundAsyncEventHandler	14
1.2.5	CLASS Clock	15
1.2.6	CLASS ExpressoIllegalMemoryAllocationException	16
1.2.7	CLASS ExpressoIllegalPhaseException	18
1.2.8	CLASS ExpressoNestedScopeException	20
1.2.9	CLASS ExpressoNotInImmortalMemoryException	22
1.2.10	CLASS ExpressoWrongThreadException	24
1.2.11	CLASS HighResolutionTime	26
1.2.12	CLASS ImmortalMemory	29
1.2.13	CLASS Initializer	30
1.2.14	CLASS Jrts	32
1.2.15	CLASS LTMemory	33
1.2.16	CLASS MemoryArea	35
1.2.17	CLASS MonitorControl	36
1.2.18	CLASS NoHeapRealtimeThread	37
1.2.19	CLASS PeriodicParameters	39
1.2.20	CLASS PeriodicThread	40
1.2.21	CLASS PriorityCeilingEmulation	42
1.2.22	CLASS PriorityParameters	43
1.2.23	CLASS PriorityScheduler	44
1.2.24	CLASS RealtimeClock	45
1.2.25	CLASS RealtimeThread	46
1.2.26	CLASS RelativeTime	48
1.2.27	CLASS ReleaseParameters	53
1.2.28	CLASS Scheduler	54
1.2.29	CLASS SchedulingParameters	55
1.2.30	CLASS ScopedMemory	55
1.2.31	CLASS SizeEstimator	57
1.2.32	CLASS SporadicEvent	59
1.2.33	CLASS SporadicEventHandler	60
1.2.34	CLASS SporadicInterrupt	61
1.2.35	CLASS SporadicParameters	62

Chapter 1

Package expresso

<i>Package Contents</i>	<i>Page</i>
<hr/>	
Interfaces	
Schedulable	5
<i>This interface is useless, but it has been introduced just for compatibility with the Java-Ravenscar profile.</i>	
Classes	
AbsoluteTime	5
<i>An object that represents a specific point in time given by milliseconds plus nanoseconds past the epoch (January 1, 1970, 00:00:00 GMT).</i>	
AsyncEvent	11
<i>An asynchronous event represents something that can happen, like a light turning red.</i>	
AsyncEventHandler	12
<i>An asynchronous event handler encapsulates code that gets run at some time after an AsyncEvent occurs.</i>	
BoundAsyncEventHandler	14
<i>A bound asynchronous event handler is an asynchronous event handler that is permanently bound to a thread.</i>	
Clock	15
<i>A clock advances from the past, through the present, into the future.</i>	
ExpressoIllegalMemoryAllocationException	16
<i>Exception thrown when a memory allocation is done in the ImmortalMemory area during the mission phase.</i>	
ExpressoIllegalPhaseException	18
<i>Exception thrown when the operation can not be performed in the current phase (initialization, mission, finalization).</i>	
ExpressoNestedScopeException	20
<i>Exception thrown by the method enter() of a given ScopedMemory object if the current thread is already in a ScopedMemory area. Expresso supports neither ScopedMemory shared between threads nor nested ScopedMemory.</i>	
ExpressoNotInImmortalMemoryException	22
<i>Exception thrown when an object is allocated in the ImmortalMemory area whereas it should be allocated in a ScopedMemory area. Expresso supports neither ScopedMemory shared between threads nor nested ScopedMemory.</i>	
ExpressoWrongThreadException	24
<i>Exception thrown when the calling thread is not the good one.</i>	

HighResolutionTime	26
<i>Abstract class that represent high resolution time value.</i>	
ImmortalMemory	29
<i>ImmortalMemory is a memory resource that is shared among all threads.</i>	
Initializer	30
<i>Initializer is a thread class used to encapsulate the initialization phase as defined in the Ravenscar-Java profile YCS 342.</i>	
Jrts	32
<i>Expresso Java RunTime System suport class.</i>	
LTMemory	33
<i>LTMemory represents a memory area, allocated per RealtimeThread, guaranteed by the system to have linear time allocation.</i>	
MemoryArea	35
<i>MemoryArea is the abstract base class of all classes dealing with representations of allocatable memory areas, including the immortal memory area and scoped memory areas.</i>	
MonitorControl	36
<i>Class for monitor control policy of some given objects.</i>	
NoHeapRealtimeThread	37
<i>A NoHeapRealtimeThread is a specialized form of RealtimeThread.</i>	
PeriodicParameters	39
<i>Expresso schedulable objects are threads.</i>	
PeriodicThread	40
<i>Expresso periodic threads are instances of PeriodicThread.</i>	
PriorityCeilingEmulation	42
<i>Monitor control class specifying use of the priority ceiling emulation protocol for monitor objects.</i>	
PriorityParameters	43
<i>Instances of this class should be assigned to threads that are managed by the fixed-priority preemptive scheduler which uses a single integer to determine execution order.</i>	
PriorityScheduler	44
<i>Fixed priority preemptive scheduling policy is assumed.</i>	
RealtimeClock	45
<i>The clock that represents the realtime clock.</i>	
RealtimeThread	46
<i>RealtimeThread extends java.lang.Thread and includes classes and methods to get and set parameter objects, manage the execution of those threads with scheduling parameters.</i>	
RelativeTime	48
<i>An object that represents a time interval millis/1E3+nanos/1E9 seconds long.</i>	
ReleaseParameters	53
<i>The abstract top-level class for release characteristics of threads.</i>	
Scheduler	54
<i>An instance of Scheduler manages the execution of schedulable objects.</i>	
SchedulingParameters	55
<i>The subclass PriorityParameters provide the parameters to be used by the scheduler.</i>	
ScopedMemory	55

ScopedMemory is the abstract base class of all classes dealing with representations of memory spaces with a limited lifetime.

SizeEstimator57
This is a convenient class to help people figure out how much memory they need.

SporadicEvent59
Sporadic events are minimal inter-arrival time-bounded asynchronous events.

SporadicEventHandler60
SporadicEventHandler are BoundAsyncEventHandler with ReleaseParameters of type SporadicParameters.

SporadicInterrupt61
Sporadic interrupts like sporadic events which are minimal inter-arrival time-bounded asynchronous events.

SporadicParameters62
Expresso schedulable objects are threads.

1.1 Interfaces

1.1.1 INTERFACE `Schedulable`

This interface is useless, but it has been introduced just for compatibility with the Java-Ravenscar profile.

Ravenscar-Java (YCS 342) non-compliance:

- None.

Expresso Software Requirements Traceability:

- No link.

DECLARATION

```
public interface Schedulable
implements java.lang Runnable
```

1.2 Classes

1.2.1 CLASS `AbsoluteTime`

An object that represents a specific point in time given by milliseconds plus nanoseconds past the epoch (January 1, 1970, 00:00:00 GMT).

Ravenscar-Java (YCS 342) non-compliance:

- NC019: the following Ravenscar-Java methods of `HighResolutionTime` are not supported:
`public AbsoluteTime(java.util.Date date)`
`public java.util.Date getDate()`
`public void set(java.util.Date date)`

Expresso Software Requirements Traceability:

- Thread05

DECLARATION

```
public class AbsoluteTime
extends expresso.HighResolutionTime
```

CONSTRUCTORS

- *AbsoluteTime*
public AbsoluteTime()
 - **Usage**
 - * Make a new `AbsoluteTime` representing the epoch (January 1, 1970, 00:00:00 GMT).

- *AbsoluteTime*
public AbsoluteTime(expresso.AbsoluteTime time)
 - **Usage**
 - * Make a new `AbsoluteTime` object from the given `AbsoluteTime` object.
 - **Parameters**
 - * `time` - The `AbsoluteTime` object as the source for the copy.

- *AbsoluteTime*
public AbsoluteTime(long millis, int nanos)
 - **Usage**
 - * Construct an `AbsoluteTime` object which means a time `millis` milliseconds plus `nanos` nanoseconds past the epoch (January 1, 1970, 00:00:00 GMT).

METHODS

- *absolute*
public AbsoluteTime absolute(expresso.Clock clock)
 - **Usage**
 - * Return this.
 - **Parameters**
 - * `clock` - this parameter is ignored.
 - **Returns** - this

- *absolute*
public AbsoluteTime absolute(expresso.Clock clock, expresso.AbsoluteTime destination)
 - **Usage**
 - * Return this, and copy it into the `destination` parameter if it is not null.
 - **Parameters**

- * `clock` - this parameter is ignored.
 - * `destination` - if not null, set it to the same value as this, else do nothing.
 - **Returns** - this
-

- *add*

```
public AbsoluteTime add( long millis, int nanos )
```

- **Usage**
 - * Add millis and nanos to this. A new object is allocated for the result. This is not modified.
 - **Parameters**
 - * `millis` - the milliseconds value to be added to this
 - * `nanos` - the nanoseconds value to be added to this
 - **Returns** - the newly allocated object with the result of adding this with millis and nanos.
-

- *add*

```
public AbsoluteTime add( long millis, int nanos, espresso.AbsoluteTime destination )
```

- **Usage**
 - * Add millis and nanos to this. If the destination parameter is not null, set it to the result, else a new object is allocated for the result. This is not modified.
 - **Parameters**
 - * `millis` - the milliseconds value to be added to this
 - * `nanos` - the nanoseconds value to be added to this
 - * `destination` - the `AbsoluteTime` that will be set to the result.
 - **Returns** - destination if not null, else the newly allocated object with the result of adding this with millis and nanos.
-

- *add*

```
public AbsoluteTime add( espresso.RelativeTime time )
```

- **Usage**
 - * Return this + time. A new object is allocated for the result. This is not modified.
 - **Parameters**
 - * `time` - the time to add to this.
 - **Returns** - the result.
-

- *add*

```
public AbsoluteTime add( espresso.RelativeTime time, espresso.AbsoluteTime destination )
```

- **Usage**
 - * Add time to this. If the destination parameter is not null, set it to the result, else a new object is allocated for the result. This is not modified.
- **Parameters**
 - * `time` - the time to add to this.
 - * `destination` - the `AbsoluteTime` that will be set to the result.
- **Returns** - destination if not null, else the newly allocated object with the result of adding time to this.

- *relative*

```
public RelativeTime relative( espresso.Clock clock )
```

- **Usage**

- * Return a newly allocated RelativeTime object that represent the time between this and the current time of the specified clock (or the realtime clock if the clock parameter is null).

- **Parameters**

- * **clock** - the clock to get the current time from.

- **Returns** - the RelativeTime representing the time between this and the current time.

- *relative*

```
public RelativeTime relative( espresso.Clock clock, espresso.RelativeTime destination )
```

- **Usage**

- * Compute the time between this and the specified clock (or the realtime clock if the clock parameter is null). If the destination parameter is not null, set it to the result, else allocate a new object for the result.

- Warning: this method allways allocate memory.

- **Parameters**

- * **clock** - the clock to get the current time from.

- **Returns** - destination if not null or the newly allocated object.

- *subtract*

```
public final RelativeTime subtract( espresso.AbsoluteTime time )
```

- **Usage**

- * Return this - time. A new object is allocated for the result. This is not modified.

- **Parameters**

- * **time** - the time to subtract to this.

- **Returns** - the result.

- *subtract*

```
public final RelativeTime subtract( espresso.AbsoluteTime time, espresso.RelativeTime destination )
```

- **Usage**

- * Subtract time to this. If the destination parameter is not null, set it to the result, else a new object is allocated for the result. This is not modified.

- **Parameters**

- * **time** - the time to subtract to this.

- * **destination** - the RelativeTime that will be set to the result.

- **Returns** - destination if not null, else the newly allocated object with the result of subtracting time to this.

- *subtract*

```
public final AbsoluteTime subtract( espresso.RelativeTime time )
```

- **Usage**
 - * Return this - time. A new object is allocated for the result. This is not modified.
- **Parameters**
 - * `time` - the time to subtract to this.
- **Returns** - the result.

- *subtract*

```
public final AbsoluteTime subtract( espresso.RelativeTime time,
    espresso.AbsoluteTime destination )
```

- **Usage**
 - * Subtract time to this. If the destination parameter is not null, set it to the result, else a new object is allocated for the result. This is not modified.
- **Parameters**
 - * `time` - the time to subtract to this.
 - * `destination` - the `AbsoluteTime` that will be set to the result.
- **Returns** - destination if not null, else the newly allocated object with the result of subtracting time to this.

- *toString*

```
public String toString( )
```

- **Usage**
 - * Return a printable version of this time, in a format that matches `java.util.Date.toString()` with a postfix to the detail the sub-second value.

FIXME: the returned string is not at all the the same format as `java.util.Date.toString()`;

- **Returns** - a newly allocated `String` object representing this time.

METHODS INHERITED FROM CLASS `expresso.HighResolutionTime`

(in 1.2.11, page 26)

- *absolute*

```
public abstract AbsoluteTime absolute( espresso.Clock clock )
```

- **Usage**
 - * Convert this time to an absolute time, relative to some clock. Convenient for situations where you really need an absolute time. Allocates a destination object if necessary. See the derived class comments for more specific information.
- **Parameters**
 - * `clock` - This clock is used to convert this time into absolute time.

- *absolute*

```
public abstract AbsoluteTime absolute( espresso.Clock clock, espresso.AbsoluteTime dest )
```

- **Usage**

- * Convert this time to an absolute time, relative to some clock. Convenient for situations where you really need an absolute time. Allocates a destination object if necessary. See the derived class comments for more specific information.

– **Parameters**

- * `clock` - This clock is used to convert this time into absolute time.
- * `dest` - if null, a new object is created and returned as result, else `dest` is returned.

- *compareTo*

```
public int compareTo( espresso.HighResolutionTime time )
```

– **Usage**

- * Compares this `HighResolutionTime` with the specified `HighResolutionTime`.

– **Parameters**

- * `time` - compares with this time.

- *compareTo*

```
public int compareTo( java.lang.Object time )
```

– **Usage**

- * For the `Comparable` interface. Specified By:
`java.lang.Comparable.compareTo(java.lang.Object)` in interface `java.lang.Comparable`

- *equals*

```
public boolean equals( espresso.HighResolutionTime time )
```

– **Usage**

- * Returns true if the argument object has the same values as this.

– **Parameters**

- * `time` - Values are compared to this.

- *equals*

```
public boolean equals( java.lang.Object object )
```

– **Usage**

- * Returns true if the argument is a `HighResolutionTime` reference and has the same values as this.

– **Parameters**

- * `object` - Values are compared to this.

- *getMilliseconds*

```
public final long getMilliseconds( )
```

– **Usage**

- * Returns the milliseconds component of this.

– **Parameters**

- * `The` - milliseconds component of the time represented by this.

- *getNanoseconds*

```
public final int getNanoseconds( )
```

– **Usage**

- * Returns the nanoseconds component of this.

– **Parameters**

- * `The` - nanoseconds component of the time represented by this.

- *hashCode*

```
public int hashCode( )
```

- *relative*

```
public abstract RelativeTime relative( espresso.Clock clock )
```

- **Usage**
 - * Convert this time to a relative time, relative to the specified clock. Convenient for situations where you really need an relative time. Allocates a destination object if necessary. See the derived class comments for more specific information.
 - **Parameters**
 - * `clock` - This clock is used to convert this time into relative time.
-
- *relative*

```
public abstract RelativeTime relative( espresso.Clock clock, espresso.RelativeTime dest )
```

 - **Usage**
 - * Convert this time to a relative time, relative to the specified clock. Convenient for situations where you really need an relative time. Allocates a destination object if necessary. See the derived class comments for more specific information.
 - **Parameters**
 - * `clock` - This clock is used to convert this time into relative time.
 - * `dest` - if null, a new object is created and returned as result, else dest is returned.
-
- *set*

```
public void set( espresso.HighResolutionTime time )
```

 - **Usage**
 - * Changes the time represented by the argument to the same value as the one given in parameter.
 - **Parameters**
 - * `time` - The HighResolutionTime that will be used to set the time of this object.
-
- *set*

```
public void set( long millis )
```

 - **Usage**
 - * Sets the millisecond component of this to the given argument.
-
- *set*

```
public void set( long millis, int nanos )
```

 - **Usage**
 - * Sets the millisecond and nanosecond components of this.
 - **Parameters**
 - * `millis` - Value to set millisecond part of this. If millis is negative the millisecond value of this is set to negative value. Although logically this may represent time before the epoch, invalid results may occur if a HighResolutionTime representing time before the epoch is given as a parameter to the methods.
 - * `nanos` - Value to set nanosecond part of this. If nanos is negative the millisecond value of this is set to negative value. Although logically this may represent time before the epoch, invalid results may occur if a HighResolutionTime representing time before the epoch is given as a parameter to the methods.

1.2.2 CLASS AsyncEvent

An asynchronous event represents something that can happen, like a light turning red. It can have a set of handlers associated with it, and when the event occurs, the handler is scheduled.

This class is abstract since only the new SporadicEvent and SporadicInterrupt are supported by this profile.

Ravenscar-Java (YCS 342) non-compliance:

- NC013: Ravenscar-Java AsyncEvent class is defined as follows:

```
public class AsyncEvent
{
    AsyncEvent();
    void addHandler;
    void fire();
    void bindTo();
}
```

Espresso Software Requirements Traceability:

- Thread08

DECLARATION

```
public abstract class AsyncEvent
extends java.lang.Object
```

METHODS

- *fire*

```
public void fire( )
```

 - **Usage**
 - * Fire (schedule the run() methods of) the handlers associated with this event.

1.2.3 CLASS AsyncEventHandler

An asynchronous event handler encapsulates code that gets run at some time after an AsyncEvent occurs. It is essentially a java.lang.Runnable with a set of parameter objects, making it very much like a RealtimeThread.

This class is abstract since only SporadicEventHandler is supported by this profile.

Ravenscar-Java (YCS 342) non-compliance:

- NC015: Ravenscar-Java AsyncEventHandler class is defined as follows:

```
public class AsyncEventHandler implements Schedulable
{
    AsyncEventHandler(PriorityParameters pp,
                    ReleaseParameters p, MemoryArea ma);
    AsyncEventHandler(PriorityParameters pp,
                    ReleaseParameters p, MemoryArea ma,
                    java.lang.Runnable logic);
    public MemoryArea getCurrentMemoryArea();
    protected void handleAsyncEvent();
    public final void run();
}
```

Espresso Software Requirements Traceability:

- Thread06
- Thread08

DECLARATION

```
public abstract class AsyncEventHandler
extends java.lang.Object
implements Schedulable
```

METHODS

- *handleAsyncEvent*

```
protected void handleAsyncEvent( )
```

– **Usage**

- * If this handler was constructed using a separate Runnable logic object, then that Runnable object's run method is called; This method will be invoked repeatedly while fireCount is greater than zero.

- *run*

```
public final void run( )
```

– **Usage**

- * This method invokes handleAsyncEvent() repeatedly while the fire count is greater than zero. Applications cannot override this method and should thus override handleAsyncEvent() in subclasses with the logic of the handler.

1.2.4 CLASS BoundAsyncEventHandler

A bound asynchronous event handler is an asynchronous event handler that is permanently bound to a thread. Bound asynchronous event handlers are meant for use in situations where the added timeliness is worth the overhead of binding the handler to a thread.

This class is abstract since only SporadicEventHandler is supported by this profile.

Ravenscar-Java (YCS 342) non-compliance:

- NC016: Ravenscar-Java BoundAsyncEventHandler class is defined as follows:

```
public class BoundAsyncEventHandler extends AsyncEventHandler
{
    BoundAsyncEventHandler(PriorityParameters pp,
                          ReleaseParameters p, MemoryArea ma);
    BoundAsyncEventHandler(PriorityParameters pp,
                          ReleaseParameters p, MemoryArea ma,
                          java.lang.Runnable logic);
    protected void handleAsyncEvent();
}
```

Espresso Software Requirements Traceability:

- Thread06
- Thread08

DECLARATION

<pre>public abstract class BoundAsyncEventHandler extends espresso.AsyncEventHandler</pre>

METHODS INHERITED FROM CLASS espresso.AsyncEventHandler

(in 1.2.3, page 12)

- *handleAsyncEvent*
protected void **handleAsyncEvent**()

– Usage

- * If this handler was constructed using a separate Runnable logic object, then that Runnable object's run method is called; This method will be invoked repeatedly while fireCount is greater than zero.

- *run*

```
public final void run( )
```

- **Usage**

- * This method invokes handleAsyncEvent() repeatedly while the fire count is greater than zero. Applications cannot override this method and should thus override handleAsyncEvent() in subclasses with the logic of the handler.

1.2.5 CLASS Clock

A clock advances from the past, through the present, into the future. It has a concept of now that can be queried through Clock.getTime(), and it can have events queued on it which will be fired when their appointed time is reached. Only one instance of clock is supported by this profile: the realtime clock.

Ravenscar-Java (YCS 342) non-compliance:

- NC022: The Ravenscar-Java profile does not require the class Clock.

Espresso Software Requirements Traceability:

- Thread05

DECLARATION

<pre>public abstract class Clock extends java.lang.Object</pre>
--

METHODS

- *getRealtimeClock*

```
public static Clock getRealtimeClock( )
```

- **Usage**

- * There is always one clock object available: a realtime clock that advances in sync with the external world. This is the default Clock and the only one supported by this profile.

-
- *getResolution*

```
public abstract RelativeTime getResolution( )
```

- **Usage**

- * Return the resolution of the clock (the interval between ticks).

- **Returns** - : A newly allocated RelativeTime object representing the resolution of this

- *getTime*

```
public AbsoluteTime getTime( )
```

- **Usage**

- * Return the current time in a freshly allocated object.

- **Returns** - An AbsoluteTime object representing the current time.

- *getTime*

```
public abstract void getTime( expresso.AbsoluteTime time )
```

- **Usage**

- * Return the current time in an existing object. The time represented by the given AbsoluteTime is changed some time between the invocation of the method and the return of the method.

- **Parameters**

- * **time** - The AbsoluteTime object which will have its time changed (if null then nothing happens).

1.2.6 CLASS `ExpressoIllegalMemoryAllocationException`

Exception thrown when a memory allocation is done in the ImmortalMemory area during the mission phase. In this phase, memory allocations are only allowed in a ScopedMemory area.

Ravenscar-Java (YCS 342) non-compliance:

- Not in Ravenscar-Java.

Expresso Software Requirements Traceability:

- Mem05

DECLARATION

```
public class ExpressoIllegalMemoryAllocationException
extends java.lang.RuntimeException
```

CONSTRUCTORS

- *ExpressoIllegalMemoryAllocationException*
`public ExpressoIllegalMemoryAllocationException()`
- *ExpressoIllegalMemoryAllocationException*
`public ExpressoIllegalMemoryAllocationException(java.lang.String s)`

METHODS INHERITED FROM CLASS `java.lang.RuntimeException`

METHODS INHERITED FROM CLASS `java.lang.Exception`

METHODS INHERITED FROM CLASS `java.lang.Throwable`

- *fillInStackTrace*
`public native Throwable fillInStackTrace()`
 - **Usage**
 - * Fills in the execution stack trace. This method records within this Throwable object information about the current state of the stack frames for the current thread. This method is useful when an application is re-throwing an error or exception. For example:

```

try {
    a = b / c;
} catch(ArithmeticThrowable e) {
    a = Double.MAX_VALUE;
    throw e.fillInStackTrace();
}

```
 - **Returns** - this Throwable object.
 - **See Also**
 - * `java.lang.Throwable.printStackTrace()`
- *getLocalizedMessage*
`public String getLocalizedMessage()`
- *getMessage*
`public String getMessage()`
 - **Usage**
 - * Returns the error message string of this throwable object.
 - **Returns** - the error message string of this Throwable object if it was created with an error message string; or null if it was created with no error message.
- *printStackTrace*
`public void printStackTrace()`
 - **Usage**

- * Prints this Throwable and its backtrace to the standard error stream. This method prints a stack trace for this Throwable object on the error output stream that is the value of the field System.err. The first line of output contains the result of the toString() method for this object.

The format of the backtrace information depends on the implementation.

- *toString*

```
public String toString( )
```

- **Usage**

- * Returns a short description of this throwable object. If this Throwable object was created with an error message string, then the result is the concatenation of three strings:
 - The name of the actual class of this object
 - : (a colon and a space)
 - The result of the getMessage() method for this object

If this Throwable object was created with no error message string, then the name of the actual class of this object is returned.

- **Returns** - a string representation of this Throwable.

1.2.7 CLASS `ExpressoIllegalPhaseException`

Exception thrown when the operation can not be performed in the current phase (initialization, mission, finalization). Expresso requires static objects to be allocated in the ImmortalMemory area during the initialization phase and dynamic objects to be allocated within ScopedMemory areas (per thread) during the mission phase.

Ravenscar-Java (YCS 342) non-compliance:

- Not in Ravenscar-Java.

Expresso Software Requirements Traceability:

- Mem05
- Mem06

DECLARATION

<pre>public class ExpressoIllegalPhaseException extends java.lang.RuntimeException</pre>

CONSTRUCTORS

- *ExpressoIllegalPhaseException*
`public ExpressoIllegalPhaseException()`
- *ExpressoIllegalPhaseException*
`public ExpressoIllegalPhaseException(java.lang.String s)`

METHODS INHERITED FROM CLASS `java.lang.RuntimeException`

METHODS INHERITED FROM CLASS `java.lang.Exception`

METHODS INHERITED FROM CLASS `java.lang.Throwable`

- *fillInStackTrace*
`public native Throwable fillInStackTrace()`
 - **Usage**
 - * Fills in the execution stack trace. This method records within this Throwable object information about the current state of the stack frames for the current thread. This method is useful when an application is re-throwing an error or exception. For example:

```

try {
    a = b / c;
} catch(ArithmeticThrowable e) {
    a = Double.MAX_VALUE;
    throw e.fillInStackTrace();
}

```
 - **Returns** - this Throwable object.
 - **See Also**
 - * `java.lang.Throwable.printStackTrace()`
- *getLocalizedMessage*
`public String getLocalizedMessage()`
- *getMessage*
`public String getMessage()`
 - **Usage**
 - * Returns the error message string of this throwable object.
 - **Returns** - the error message string of this Throwable object if it was created with an error message string; or null if it was created with no error message.
- *printStackTrace*
`public void printStackTrace()`
 - **Usage**

- * Prints this Throwable and its backtrace to the standard error stream. This method prints a stack trace for this Throwable object on the error output stream that is the value of the field System.err. The first line of output contains the result of the toString() method for this object.

The format of the backtrace information depends on the implementation.

- *toString*

`public String toString()`

- **Usage**

- * Returns a short description of this throwable object. If this Throwable object was created with an error message string, then the result is the concatenation of three strings:

- The name of the actual class of this object
- (a colon and a space)
- The result of the `getMessage()` method for this object

If this Throwable object was created with no error message string, then the name of the actual class of this object is returned.

- **Returns** - a string representation of this Throwable.

1.2.8 CLASS `ExpressoNestedScopeException`

Exception thrown by the method `enter()` of a given `ScopedMemory` object if the current thread is already in a `ScopedMemory` area. Expresso supports neither `ScopedMemory` shared between threads nor nested `ScopedMemory`.

Ravenscar-Java (YCS 342) non-compliance:

- Not in Ravenscar-Java.

Expresso Software Requirements Traceability:

- Mem05

DECLARATION

<pre>public class ExpressoNestedScopeException extends java.lang.RuntimeException</pre>
--

CONSTRUCTORS

- *ExpressoNestedScopeException*
public **ExpressoNestedScopeException**()
- *ExpressoNestedScopeException*
public **ExpressoNestedScopeException**(java.lang.String s)

METHODS INHERITED FROM CLASS `java.lang.RuntimeException`

METHODS INHERITED FROM CLASS `java.lang.Exception`

METHODS INHERITED FROM CLASS `java.lang.Throwable`

- *fillInStackTrace*
public native **Throwable fillInStackTrace**()
 - **Usage**
 - * Fills in the execution stack trace. This method records within this Throwable object information about the current state of the stack frames for the current thread. This method is useful when an application is re-throwing an error or exception. For example:

```

try {
    a = b / c;
} catch(ArithmeticThrowable e) {
    a = Double.MAX_VALUE;
    throw e.fillInStackTrace();
}

```
 - **Returns** - this Throwable object.
 - **See Also**
 - * `java.lang.Throwable.printStackTrace()`
- *getLocalizedMessage*
public **String getLocalizedMessage**()
- *getMessage*
public **String getMessage**()
 - **Usage**
 - * Returns the error message string of this throwable object.
 - **Returns** - the error message string of this Throwable object if it was created with an error message string; or null if it was created with no error message.
- *printStackTrace*
public void **printStackTrace**()
 - **Usage**
 - * Prints this Throwable and its backtrace to the standard error stream. This method prints a stack trace for this Throwable object on the error output stream that is the value of the field `System.err`. The first line of output contains the result of the `toString()` method for this object.

The format of the backtrace information depends on the implementation.

-
- *toString*
`public String toString()`
 - **Usage**
 - * Returns a short description of this throwable object. If this Throwable object was created with an error message string, then the result is the concatenation of three strings:
 - The name of the actual class of this object
 - (a colon and a space)
 - The result of the `getMessage()` method for this object
 - If this Throwable object was created with no error message string, then the name of the actual class of this object is returned.
 - **Returns** - a string representation of this Throwable.

1.2.9 CLASS *ExpressoNotInImmortalMemoryException*

Exception thrown when an object is allocated in the *ImmortalMemory* area whereas it should be allocated in a *ScopedMemory* area. *Expresso* supports neither *ScopedMemory* shared between threads nor nested *ScopedMemory*.

Ravenscar-Java (YCS 342) non-compliance:

- Not in Ravenscar-Java.

Expresso Software Requirements Traceability:

- Mem05

DECLARATION

```
public class ExpressoNotInImmortalMemoryException
extends java.lang.RuntimeException
```

CONSTRUCTORS

- *ExpressoNotInImmortalMemoryException*
`public ExpressoNotInImmortalMemoryException()`
- *ExpressoNotInImmortalMemoryException*
`public ExpressoNotInImmortalMemoryException(java.lang.String s)`

METHODS INHERITED FROM CLASS *java.lang.RuntimeException*

METHODS INHERITED FROM CLASS `java.lang.Exception`

METHODS INHERITED FROM CLASS `java.lang.Throwable`

• *fillInStackTrace*

```
public native Throwable fillInStackTrace( )
```

– Usage

- * Fills in the execution stack trace. This method records within this Throwable object information about the current state of the stack frames for the current thread. This method is useful when an application is re-throwing an error or exception. For example:

```
try {
    a = b / c;
} catch(ArithmeticThrowable e) {
    a = Double.MAX_VALUE;
    throw e.fillInStackTrace();
}
```

- **Returns** - this Throwable object.

– See Also

- * `java.lang.Throwable.printStackTrace()`

• *getLocalizedMessage*

```
public String getLocalizedMessage( )
```

• *getMessage*

```
public String getMessage( )
```

– Usage

- * Returns the error message string of this throwable object.
- **Returns** - the error message string of this Throwable object if it was created with an error message string; or null if it was created with no error message.

• *printStackTrace*

```
public void printStackTrace( )
```

– Usage

- * Prints this Throwable and its backtrace to the standard error stream. This method prints a stack trace for this Throwable object on the error output stream that is the value of the field `System.err`. The first line of output contains the result of the `toString()` method for this object.

The format of the backtrace information depends on the implementation.

• *toString*

```
public String toString( )
```

– Usage

- * Returns a short description of this throwable object. If this Throwable object was created with an error message string, then the result is the concatenation of three strings:
 - The name of the actual class of this object
 - " " (a colon and a space)
 - The result of the `getMessage()` method for this object

If this Throwable object was created with no error message string, then the name of the actual class of this object is returned.

- **Returns** - a string representation of this Throwable.

1.2.10 CLASS **ExpressoWrongThreadException**

Exception thrown when the calling thread is not the good one. This exception is thrown by the following methods:

- by the `enter()` method of the `ScopedMemory` class if the current thread is not the one associated with the specified `ScopedMemory` object (see `ScopedMemory.setAssociatedThread(RealtimeThread)` (in 1.2.30, page 57))
- by the `waitForStart()` and the `waitForNextPeriod()` method of the `PeriodicThread` class if the current thread is not the one on which the method is called.

Ravenscar-Java (YCS 342) non-compliance:

- Not in Ravenscar-Java.

Expresso Software Requirements Traceability:

- Mem04

DECLARATION

```
public class ExpressoWrongThreadException
extends java.lang.RuntimeException
```

CONSTRUCTORS

- *ExpressoWrongThreadException*
`public ExpressoWrongThreadException()`
- *ExpressoWrongThreadException*
`public ExpressoWrongThreadException(java.lang.String s)`

METHODS INHERITED FROM CLASS `java.lang.RuntimeException`

METHODS INHERITED FROM CLASS `java.lang.Exception`

METHODS INHERITED FROM CLASS `java.lang.Throwable`

• *fillInStackTrace*

```
public native Throwable fillInStackTrace( )
```

– Usage

- * Fills in the execution stack trace. This method records within this `Throwable` object information about the current state of the stack frames for the current thread. This method is useful when an application is re-throwing an error or exception. For example:

```
try {
    a = b / c;
} catch(ArithmeticThrowable e) {
    a = Double.MAX_VALUE;
    throw e.fillInStackTrace();
}
```

– **Returns** - this `Throwable` object.

– **See Also**

- * `java.lang.Throwable.printStackTrace()`

• *getLocalizedMessage*

```
public String getLocalizedMessage( )
```

• *getMessage*

```
public String getMessage( )
```

– Usage

- * Returns the error message string of this throwable object.
- **Returns** - the error message string of this `Throwable` object if it was created with an error message string; or null if it was created with no error message.

• *printStackTrace*

```
public void printStackTrace( )
```

– Usage

- * Prints this `Throwable` and its backtrace to the standard error stream. This method prints a stack trace for this `Throwable` object on the error output stream that is the value of the field `System.err`. The first line of output contains the result of the `toString()` method for this object.

The format of the backtrace information depends on the implementation.

• *toString*

```
public String toString( )
```

– Usage

- * Returns a short description of this throwable object. If this `Throwable` object was created with an error message string, then the result is the concatenation of three strings:
 - The name of the actual class of this object
 - ":" (a colon and a space)
 - The result of the `getMessage()` method for this object

If this `Throwable` object was created with no error message string, then the name of the actual class of this object is returned.

– **Returns** - a string representation of this `Throwable`.

1.2.11 CLASS **HighResolutionTime**

Abstract class that represent high resolution time value. It is the base class for `AbsoluteTime` and `RelativeTime` (`RationalTime` is not supported by this profile).

Ravenscar-Java (YCS 342) non-compliance:

- NC018: the following Ravenscar-Java methods of `HighResolutionTime` are not supported:

```
public static void waitForObject(java.lang.Object target,
                               HighResolutionTime time)
    throws InterruptedException
```

Espresso Software Requirements Traceability:

- Thread05

DECLARATION

```
public abstract class HighResolutionTime
extends java.lang.Object
implements java.lang.Comparable
```

METHODS

- *absolute*

```
public abstract AbsoluteTime absolute( espresso.Clock clock )
```

- **Usage**

- * Convert this time to an absolute time, relative to some clock. Convenient for situations where you really need an absolute time. Allocates a destination object if necessary. See the derived class comments for more specific information.

- **Parameters**

- * `clock` - This clock is used to convert this time into absolute time.

- *absolute*

```
public abstract AbsoluteTime absolute( espresso.Clock clock,
                                       espresso.AbsoluteTime dest )
```

- **Usage**

- * Convert this time to an absolute time, relative to some clock. Convenient for situations where you really need an absolute time. Allocates a destination object if necessary. See the derived class comments for more specific information.

– **Parameters**

- * **clock** - This clock is used to convert this time into absolute time.
- * **dest** - if null, a new object is created and returned as result, else dest is returned.

- *compareTo*

```
public int compareTo( expresso.HighResolutionTime time )
```

– **Usage**

- * Compares this HighResolutionTime with the specified HighResolutionTime.

– **Parameters**

- * **time** - compares with this time.

- *compareTo*

```
public int compareTo( java.lang.Object time )
```

– **Usage**

- * For the Comparable interface. Specified By:
java.lang.Comparable.compareTo(java.lang.Object) in interface
java.lang.Comparable

- *equals*

```
public boolean equals( expresso.HighResolutionTime time )
```

– **Usage**

- * Returns true if the argument object has the same values as this.

– **Parameters**

- * **time** - Values are compared to this.

- *equals*

```
public boolean equals( java.lang.Object object )
```

– **Usage**

- * Returns true if the argument is a HighResolutionTime reference and has the same values as this.

– **Parameters**

- * **object** - Values are compared to this.

- *getMilliseconds*

```
public final long getMilliseconds( )
```

– **Usage**

- * Returns the milliseconds component of this.

– **Parameters**

- * **The** - milliseconds component of the time represented by this.

- *getNanoseconds*

```
public final int getNanoseconds( )
```

- **Usage**
 - * Returns the nanoseconds component of this.
- **Parameters**
 - * The - nanoseconds component of the time represented by this.

- *hashCode*

```
public int hashCode( )
```

- *relative*

```
public abstract RelativeTime relative( espresso.Clock clock )
```

- **Usage**
 - * Convert this time to a relative time, relative to the specified clock. Convenient for situations where you really need an relative time. Allocates a destination object if necessary. See the derived class comments for more specific information.
- **Parameters**
 - * `clock` - This clock is used to convert this time into relative time.

- *relative*

```
public abstract RelativeTime relative( espresso.Clock clock,
    espresso.RelativeTime dest )
```

- **Usage**
 - * Convert this time to a relative time, relative to the specified clock. Convenient for situations where you really need an relative time. Allocates a destination object if necessary. See the derived class comments for more specific information.
- **Parameters**
 - * `clock` - This clock is used to convert this time into relative time.
 - * `dest` - if null, a new object is created and returned as result, else dest is returned.

- *set*

```
public void set( espresso.HighResolutionTime time )
```

- **Usage**
 - * Changes the time represented by the argument to the same value as the one given in parameter.
- **Parameters**
 - * `time` - The HighResolutionTime that will be used to set the time of this object.

- *set*

```
public void set( long millis )
```

- **Usage**
 - * Sets the millisecond component of this to the given argument.

- *set*

```
public void set( long millis, int nanos )
```

- **Usage**
 - * Sets the millisecond and nanosecond components of this.
- **Parameters**

- * `millis` - Value to set millisecond part of this. If `millis` is negative the millisecond value of this is set to negative value. Although logically this may represent time before the epoch, invalid results may occur if a `HighResolutionTime` representing time before the epoch is given as a parameter to the methods.
- * `nanos` - Value to set nanosecond part of this. If `nanos` is negative the millisecond value of this is set to negative value. Although logically this may represent time before the epoch, invalid results may occur if a `HighResolutionTime` representing time before the epoch is given as a parameter to the methods.

1.2.12 CLASS `ImmortalMemory`

`ImmortalMemory` is a memory resource that is shared among all threads. Objects allocated in the immortal memory live until the end of the application and are never subject to garbage collection. Unlike standard Java heap objects, immortal objects continue to exist even after there are no other references to them.

Ravenscar-Java (YCS 342) non-compliance:

- NC004: Ravenscar-Java `ImmortalMemory` class is defined as follows:

```
public final class ImmortalMemory extends ScopedMemory
{
    public static ImmortalMemory instance();
}
```

Expresso Software Requirements Traceability:

- Mem01
- Mem03
- Mem06

DECLARATION

```
public final class ImmortalMemory
extends expresso.MemoryArea
```

METHODS

- *getInstance*

```
public static ImmortalMemory getInstance( )
```

– Usage

- * Returns a pointer to the singleton ImmortalMemory space.
- **Returns** - The singleton ImmortalMemory object.

- *memoryConsumed*

public final long **memoryConsumed**()

- **Usage**

- * returns the number of bytes in use of the ImmortalMemory area

- *memoryRemaining*

public final long **memoryRemaining**()

- **Usage**

- * returns the number of not used bytes of the ImmortalMemory area

- *size*

public final long **size**()

- **Usage**

- * returns the size of the ImmortalMemory area in bytes

METHODS INHERITED FROM CLASS `expresso.MemoryArea`

(in 1.2.16, page 35)

- *memoryConsumed*

public abstract long **memoryConsumed**()

- *memoryRemaining*

public abstract long **memoryRemaining**()

- *size*

public abstract long **size**()

- **Returns** - The size of the memory area in bytes.

1.2.13 CLASS **Initializer**

Initializer is a thread class used to encapsulate the initialization phase as defined in the Ravenscar-Java profile YCS 342.

Ravenscar-Java (YCS 342) non-compliance:

- None.

Espresso Software Requirements Traceability:

- Mem01
- Mem03

- Mem06

DECLARATION

```
public class Initializer
extends espresso.RealtimeThread
```

CONSTRUCTORS

- *Initializer*
public **Initializer**()

METHODS

- *start*
public void **start**()

METHODS INHERITED FROM CLASS espresso.RealtimeThread

(in 1.2.25, page 46)

- *currentRealtimeThread*
public static native RealtimeThread currentRealtimeThread()
- *getCurrentMemoryArea*
public static MemoryArea **getCurrentMemoryArea**()

– **Usage**

* Return the instance of MemoryArea which is the current memory area for this.

- *run*
public void **run**()
- *sleep*
public static void **sleep**(espresso.AbsoluteTime time)
- *start*
public void **start**()

METHODS INHERITED FROM CLASS java.lang.Thread

- *run*
public abstract void **run**()
– **Usage**
* The method executed by this thread. Subclasses of Thread should override this method.
- *start*
public abstract void **start**()

- **Usage**
 - * Activate this thread. All the activated thread will start at the beginning of the mission phase, after the end of the main method of the application.
- **Exceptions**
 - * `java.lang.IllegalThreadStateException` - if the thread was already started.
- **See Also**
 - * `java.lang.Thread.run()`

1.2.14 CLASS Jrts

Expresso Java RunTime System suport class.

Ravenscar-Java (YCS 342) non-compliance:

- Not in Java-Ravenscar: Not applicable

Expresso Software Requirements Traceability:

- All: (Thread01 to Thread10, Mem01 to Mem02 and Exc01, Exc02)

DECLARATION

```
public final class Jrts
extends java.lang.Object
```

FIELDS

- public static final int STATE_INIT
 -
- public static final int STATE_MISSION
 -
- public static final int STATE_FINISH
 -
- public static volatile int state
 -
- public static final int NO_ERROR
 -

METHODS

- *refToInt*

```
public static native int refToInt( java.lang.Object o )
```

1.2.15 CLASS *L*Memory

*L*Memory represents a memory area, allocated per *RealtimeThread*, guaranteed by the system to have linear time allocation.

Ravenscar-Java (YCS 342) non-compliance:

- None

Expresso Software Requirements Traceability:

- Mem01
- Mem03
- Mem04
- Mem05

DECLARATION

```
public class LMemory
extends expresso.ScopedMemory
```

CONSTRUCTORS

- *L*Memory

```
public LMemory( long size )
```

 - **Usage**
 - * Public constructor
 - **Parameters**
 - * *size* - a long
-
- *L*Memory

```
public LMemory( expresso.SizeEstimator size )
```

 - **Usage**
 - * Public constructor
 - **Parameters**
 - * *size* - a *SizeEstimator*

METHODS INHERITED FROM CLASS `expresso.ScopedMemory`

(in 1.2.30, page 55)

- *enter*

```
public void enter( java.lang Runnable logic )
```

 - **Usage**
 - * Associate this memory area to the current real-time thread for the duration of the execution of the `run()` method of the given `java.lang Runnable`. During this bound period of execution, all objects are allocated from this `ScopedMemory` area until another one takes effect, or the `enter()` method is exited. `ScopedMemory` areas are not sharable between threads.
 - **Parameters**
 - * `logic` - a `Runnable` object.
 - **Exceptions**
 - * `expresso.ExpressoNestedScopeException` - if the current thread is already in a `ScopedMemory` area.
 - * `expresso.ExpressoWrongThreadException` - if this `ScopedMemory` object is not associated to the current thread (see `setAssociatedThread(RealtimeThread)`).

- *memoryConsumed*

```
public final long memoryConsumed( )
```

 - **Returns** - the number of bytes in use of this `ScopedMemory` area.

- *memoryRemaining*

```
public final long memoryRemaining( )
```

 - **Returns** - the number of not used bytes of this `ScopedMemory` area.

- *setAssociatedThread*

```
public void setAssociatedThread( expresso.RealtimeThread thread )
```

 - **Usage**
 - * Set the associated thread of this `ScopedMemory`. This is the only one thread that will be allowed to enter this `ScopedMemory`.

FIXME: this method is not defined in the RTSJ !
 - **Exceptions**
 - * `java.lang.IllegalArgumentException` - if this `ScopedMemory` object is already associated to a thread.
 - * `expresso.ExpressoIllegalPhaseException` - if this is not the initialization phase of the application.

- *size*

```
public final long size( )
```

 - **Returns** - the size of this `ScopedMemory` area in bytes.

METHODS INHERITED FROM CLASS `expresso.MemoryArea`

(in 1.2.16, page 35)

- *memoryConsumed*

```
public abstract long memoryConsumed( )
```
- *memoryRemaining*

```
public abstract long memoryRemaining( )
```
- *size*

```
public abstract long size( )
```

 - **Returns** - The size of the memory area in bytes.

1.2.16 CLASS *MemoryArea*

MemoryArea is the abstract base class of all classes dealing with representations of allocatable memory areas, including the immortal memory area and scoped memory areas.

Ravenscar-Java (YCS 342) non-compliance:

- NC001: Ravenscar-Java constructors are defined as follows:

```
protected MemoryArea(long sizeInBytes);
protected MemoryArea(javax.realtime.SizeEstimator size);
```
- NC002: Ravenscar-Java methods are defined as follows:

```
public void enter(java.lang.Runnable logic)
throws InaccessibleAreaException;
public long memoryConsumed();
public long memoryRemaining();
public long size();
```
- NC003: Ravenscar-Java method not supported by Espresso profile:

```
public java.lang.Object newInstance(
    java.lang.reflect.Constructor constructor,
    java.lang.Object[] args)
throws java.lang.IllegalAccessException
    java.lang.InstantiationException;
```

Espresso Software Requirements Traceability:

- Mem01
- Mem03

DECLARATION

```
public abstract class MemoryArea
extends java.lang.Object
```

METHODS

- *memoryConsumed*
`public abstract long memoryConsumed()`
- *memoryRemaining*
`public abstract long memoryRemaining()`
- *size*
`public abstract long size()`
– **Returns** - The size of the memory area in bytes.

1.2.17 CLASS MonitorControl

Class for monitor control policy of some given objects. Monitor control policies aim at priority inversion control. Espresso only supports the priority ceiling emulation policy.

Ravenscar-Java (YCS 342) non-compliance:

- NC024: The Ravenscar-Java profile apparently supports the following MonitorControl:

```
public abstract class MonitorControl {
    public MonitorControl();
    public static MonitorControl getMonitorControl();
    public static MonitorControl getMonitorControl(java.lang.Object monitor);
    public static void setMonitorControl(MonitorControl policy);
    public static void setMonitorControl(java.lang.Object MonitorControl monCtl);
}
```

Espresso Software Requirements Traceability:

- Thread07

DECLARATION

```
public abstract class MonitorControl
extends java.lang.Object
```

METHODS

- *setMonitorControl*
`public static void setMonitorControl(java.lang.Object monitor,
expresso.PriorityCeilingEmulation monitorControl)`

1.2.18 CLASS `NoHeapRealtimeThread`

A `NoHeapRealtimeThread` is a specialized form of `RealtimeThread`. The Espresso memory model assumes that neither heap nor related garbage collector mechanism are supported. All memory allocations for an Espresso program are performed in the `ImmortalMemory` area at initialization phase or in `LTMemory` areas during the mission phase. Consequently all Espresso thread objects are instances of the class `NoHeapRealtimeThread`.

Ravenscar-Java (YCS 342) non-compliance:

- NC011: Ravenscar-Java `NoHeapRealtimeThread` class is defined as follows:

```
public class NoHeapRealtimeThread extends RealtimeThread
{
    NoHeapRealtimeThread( PriorityParameters pp,
        MemoryArea ma);
    NoHeapRealtimeThread( PriorityParameters pp,
        PeriodicParameters p, MemoryArea ma);
    void start();
}
```

Espresso Software Requirements Traceability:

- Thread01
- Thread02

DECLARATION

```
public class NoHeapRealtimeThread
extends espresso.RealtimeThread
implements Schedulable
```

CONSTRUCTORS

- *NoHeapRealtimeThread*

```
public NoHeapRealtimeThread( espresso.PriorityParameters pp )
```

 - **Usage**
 - * Create a `NoHeapRealtimeThread` instance with the specified priority.
 - **Parameters**
 - * `pp` - the priority of this thread.

- *NoHeapRealtimeThread*

```
public NoHeapRealtimeThread( espresso.PriorityParameters pp,
    java.lang Runnable logic )
```

 - **Usage**
 - * Create a NoHeapRealtimeThread instance with the specified priority and logic.
 - **Parameters**
 - * `pp` - the priority of this thread.
 - * `logic` - a Runnable whose run() method will be executed for this.

METHODS INHERITED FROM CLASS `expresso.RealtimeThread`

(in 1.2.25, page 46)

- *currentRealtimeThread*

```
public static native RealtimeThread currentRealtimeThread( )
```
- *getCurrentMemoryArea*

```
public static MemoryArea getCurrentMemoryArea( )
```

 - **Usage**
 - * Return the instance of MemoryArea which is the current memory area for this.
- *run*

```
public void run( )
```
- *sleep*

```
public static void sleep( espresso.AbsoluteTime time )
```
- *start*

```
public void start( )
```

METHODS INHERITED FROM CLASS `java.lang.Thread`

- *run*

```
public abstract void run( )
```

 - **Usage**
 - * The method executed by this thread. Subclasses of Thread should override this method.
- *start*

```
public abstract void start( )
```

 - **Usage**
 - * Activate this thread. All the activated thread will start at the beginning of the mission phase, after the end of the main method of the application.
 - **Exceptions**
 - * `java.lang.IllegalThreadStateException` - if the thread was already started.
 - **See Also**
 - * `java.lang.Thread.run()`

1.2.19 CLASS **PeriodicParameters**

Expresso schedulable objects are threads. An instance of `PeriodicParameters` class is bound to a given periodic thread. An instance of `PeriodicParameters` class defines release characteristics of the thread to which it is bound.

Ravenscar-Java (YCS 342) non-compliance:

- NC009: Ravenscar-Java modifier for methods `AbsoluteTime()` and `RelativeTime()` is protected

Expresso Software Requirements Traceability:

- Thread04

DECLARATION

```
public class PeriodicParameters
extends expresso.ReleaseParameters
```

CONSTRUCTORS

- *PeriodicParameters*
`public PeriodicParameters(expresso.AbsoluteTime startTime,
expresso.RelativeTime period)`

METHODS

- *getPeriod*
`public RelativeTime getPeriod()`
- *getStartTime*
`public AbsoluteTime getStartTime()`

METHODS INHERITED FROM CLASS `expresso.ReleaseParameters`

(in 1.2.27, page 53)

1.2.20 CLASS *PeriodicThread*

Espresso periodic threads are instances of *PeriodicThread*. Periodic threads invoke the `waitForNextPeriod` method at the end of their main loop to delay until their next period.

Ravenscar-Java (YCS 342) non-compliance:

- NC012: Ravenscar-Java *PeriodicThread* class is defined as follows:


```
public class PeriodicThread extends NoHeapRealtimeThread
{
    public PeriodicThread(PriorityParameters pp, PeriodicParameters p,
                          PeriodicParameters p, java.lang.Runnable logic);
    void run();
    void start();
}
```

Espresso Software Requirements Traceability:

- Thread04

DECLARATION

<pre>public class PeriodicThread extends espresso.NoHeapRealtimeThread</pre>

CONSTRUCTORS

- *PeriodicThread*

```
public PeriodicThread( espresso.PriorityParameters pp,
                       espresso.PeriodicParameters p, espresso.LTMemory ma )
```
- *PeriodicThread*

```
public PeriodicThread( espresso.PriorityParameters pp,
                       espresso.PeriodicParameters p, espresso.LTMemory ma, java.lang.Runnable
logic )
```

METHODS

- *handlePeriod*

```
public void handlePeriod( )
```

 - Usage

* Invoke the method `run()` of this thread

- *run*

`public final void run()`

- **Usage**

* The method `run()` of periodic threads. Periodic threads first wait for the initial trigger (start time), then in an infinite loop they perform their specific job and wait for the next period.

- *waitForNextPeriod*

`public boolean waitForNextPeriod()`

- **Usage**

* Used by threads that have a reference to a `SchedulingParameters` type of `PeriodicParameters` to block until the start of each period. This method will block until the start of the next period unless the thread is in either an overrun or deadline miss condition.

- **Returns** - a boolean value which is true except in case of overrun or deadline miss.

METHODS INHERITED FROM CLASS `expresso.NoHeapRealtimeThread`

(in 1.2.18, page 37)

METHODS INHERITED FROM CLASS `expresso.RealtimeThread`

(in 1.2.25, page 46)

- *currentRealtimeThread*

`public static native RealtimeThread currentRealtimeThread()`

- *getCurrentMemoryArea*

`public static MemoryArea getCurrentMemoryArea()`

- **Usage**

* Return the instance of `MemoryArea` which is the current memory area for this.

- *run*

`public void run()`

- *sleep*

`public static void sleep(expresso.AbsoluteTime time)`

- *start*

`public void start()`

METHODS INHERITED FROM CLASS `java.lang.Thread`

- *run*

`public abstract void run()`

- **Usage**

* The method executed by this thread. Subclasses of `Thread` should override this method.

- *start*

```
public abstract void start( )
```

 - **Usage**
 - * Activate this thread. All the activated thread will start at the beginning of the mission phase, after the end of the main method of the application.
 - **Exceptions**
 - * `java.lang.IllegalThreadStateException` - if the thread was already started.
 - **See Also**
 - * `java.lang.Thread.run()`

1.2.21 CLASS PriorityCeilingEmulation

Monitor control class specifying use of the priority ceiling emulation protocol for monitor objects. Objects under the influence of this protocol have the effect that a thread entering the monitor has its effective priority raised to the ceiling on entry, and is restored to its previous effective priority when it exits the monitor.

Ravenscar-Java (YCS 342) non-compliance:

- None.

Expresso Software Requirements Traceability:

- Thread07

DECLARATION

```
public class PriorityCeilingEmulation
extends expresso.MonitorControl
```

CONSTRUCTORS

- *PriorityCeilingEmulation*

```
public PriorityCeilingEmulation( int ceiling )
```

METHODS

- *getDefaultCeiling*

```
public int getDefaultCeiling( )
```

METHODS INHERITED FROM CLASS `expresso.MonitorControl`

(in 1.2.17, page 36)

- *setMonitorControl*
`public static void setMonitorControl(java.lang.Object monitor,
expresso.PriorityCeilingEmulation monitorControl)`

1.2.22 CLASS `PriorityParameters`

Instances of this class should be assigned to threads that are managed by the fixed-priority preemptive scheduler which uses a single integer to determine execution order.

Ravenscar-Java (YCS 342) non-compliance:

- NC007: Ravenscar-Java supports the method `toString()`

Espresso Software Requirements Traceability:

- Thread03

DECLARATION

```
public class PriorityParameters
extends espresso.SchedulingParameters
```

CONSTRUCTORS

- *PriorityParameters*
`public PriorityParameters(int priority)`
 - **Usage**
 - * Create an instance of `SchedulingParameters` with the given priority.

METHODS

- *getPriority*
`public int getPriority()`
 - **Usage**
 - * Get the priority

METHODS INHERITED FROM CLASS `expresso.SchedulingParameters`

(in 1.2.29, page 55)

1.2.23 CLASS `PriorityScheduler`

Fixed priority preemptive scheduling policy is assumed.

Ravenscar-Java (YCS 342) non-compliance:

- NC006: Ravenscar-Java priorities are defined as follows:
`public static final int MAX_PRIORITY;`
`public static final int MIN_PRIORITY;`

Expresso Software Requirements Traceability:

- Thread03

DECLARATION

```
public class PriorityScheduler
extends expresso.Scheduler
```

FIELDS

- `public static final int MAX_PRIORITY`
—
- `public static final int MIN_PRIORITY`
—

CONSTRUCTORS

- *PriorityScheduler*
`public PriorityScheduler()`

METHODS INHERITED FROM CLASS `expresso.Scheduler`

(in 1.2.28, page 54)

1.2.24 CLASS `RealtimeClock`

The clock that represents the realtime clock.

Ravenscar-Java (YCS 342) non-compliance:

- NC023: The Ravenscar-Java profile does not require the class `RealtimeClock`.

Espresso Software Requirements Traceability:

- Thread05

DECLARATION

```
public class RealtimeClock
extends espresso.Clock
```

METHODS

- *getResolution*

```
public RelativeTime getResolution( )
```

 - **Usage**
 - * Return the resolution of the clock (the interval between ticks).
 - **Returns** - : A newly allocated `RelativeTime` object representing the resolution of this

- *getTime*

```
public void getTime( espresso.AbsoluteTime time )
```

 - **Usage**
 - * Return the current time in an existing object. The time represented by the given `AbsoluteTime` is changed some time between the invocation of the method and the return of the method.
 - **Parameters**
 - * **time** - The `AbsoluteTime` object which will have its time changed (if null then nothing happens).

METHODS INHERITED FROM CLASS `expresso.Clock`

(in 1.2.5, page 15)

- *getRealtimeClock*

```
public static Clock getRealtimeClock( )
```

– Usage

- * There is always one clock object available: a realtime clock that advances in sync with the external world. This is the default `Clock` and the only one supported by this profile.

- *getResolution*

```
public abstract RelativeTime getResolution( )
```

– Usage

- * Return the resolution of the clock (the interval between ticks).

– Returns - : A newly allocated `RelativeTime` object representing the resolution of this

- *getTime*

```
public AbsoluteTime getTime( )
```

– Usage

- * Return the current time in a freshly allocated object.

– Returns - An `AbsoluteTime` object representing the current time.

- *getTime*

```
public abstract void getTime( espresso.AbsoluteTime time )
```

– Usage

- * Return the current time in an existing object. The time represented by the given `AbsoluteTime` is changed some time between the invocation of the method and the return of the method.

– Parameters

- * `time` - The `AbsoluteTime` object which will have its time changed (if null then nothing happens).

1.2.25 CLASS `RealtimeThread`

`RealtimeThread` extends `java.lang.Thread` and includes classes and methods to get and set parameter objects, manage the execution of those threads with scheduling parameters. As consequence of the Espresso memory model, instances of `RealtimeThread` are not allowed. See also `expresso.NoHeapRealtimeThread`

Ravenscar-Java (YCS 342) non-compliance:

- NC010: Ravenscar-Java `RealTimeThread` class is defined as follows:

```
public class RealTimeThread extends java.lang.Thread
    implements Schedulable
{
    RealTimeThread( PriorityParameters pp,
        PeriodicParameters p);
    RealTimeThread( PriorityParameters pp,
        PeriodicParameters p, MemoryArea ma);
    public static RealtimeThread currentRealtimeThread();
    public MemoryArea getCurrentMemoryArea();
}
```

```

    void start();
    static boolean waitForNextPeriod();
}

```

Espresso Software Requirements Traceability:

- Thread01
- Thread02

DECLARATION

```

public abstract class RealtimeThread
extends java.lang.Thread
implements Schedulable

```

METHODS

- *currentRealtimeThread*
public static native RealtimeThread currentRealtimeThread()
- *getCurrentMemoryArea*
public static MemoryArea getCurrentMemoryArea()
 - **Usage**
 - * Return the instance of MemoryArea which is the current memory area for this.
- *run*
public void run()
- *sleep*
public static void sleep(espresso.AbsoluteTime time)
- *start*
public void start()

METHODS INHERITED FROM CLASS java.lang.Thread

- *run*
public abstract void run()
 - **Usage**
 - * The method executed by this thread. Subclasses of Thread should override this method.

- *start*

```
public abstract void start( )
```

 - **Usage**
 - * Activate this thread. All the activated thread will start at the beginning of the mission phase, after the end of the main method of the application.
 - **Exceptions**
 - * `java.lang.IllegalThreadStateException` - if the thread was already started.
 - **See Also**
 - * `java.lang.Thread.run()`

1.2.26 CLASS **RelativeTime**

An object that represents a time interval `millis/1E3+nanos/1E9` seconds long. It generally is used to represent a time relative to now.

Ravenscar-Java (YCS 342) non-compliance:

- NC020: the following Ravenscar-Java methods of `HighResolutionTime` are not supported:

```
public void addInterarrivalTo(AbsoluteTime destination)
public RelativeTime getInterarrivalTime()
public RelativeTime getInterarrivalTime(RelativeTime destination)
```
- NC021: the subclass `RationalTime` is not supported.

Expresso Software Requirements Traceability:

- Thread05

DECLARATION

```
public class RelativeTime
extends expresso.HighResolutionTime
```

CONSTRUCTORS

- *RelativeTime*

```
public RelativeTime( )
```

 - **Usage**
 - * Equivalent to `new RelativeTime(0,0)`
-

- *RelativeTime*

```
public RelativeTime( long millis, int nanos )
```

- **Usage**

- * Construct a `RelativeTime` object which means a time `millis` milliseconds plus `nanos` nanoseconds past the `Clock` time.

- **Parameters**

- * `millis` - The milliseconds component of the time past the `Clock` time.
- * `nanos` - The nanoseconds component of the time past the `Clock`.

- *RelativeTime*

```
public RelativeTime( expresso.RelativeTime time )
```

- **Usage**

- * Make a new `RelativeTime` object from the given `RelativeTime` object

- **Parameters**

- * `time` - The `RelativeTime` object used as the source for the copy.

METHODS

- *absolute*

```
public AbsoluteTime absolute( expresso.Clock clock )
```

- **Usage**

- * Convert this time to an absolute time. For a `RelativeTime`, this involve adding the clocks notion of now to this interval and constructing a new `AbsoluteTime` based on the sum.

- **Parameters**

- * `clock` - if null, `Clock.getRealTimeClock()` is used.

- **Returns** - the `AbsoluteTime` representing the current time plus this.

- *absolute*

```
public AbsoluteTime absolute( expresso.Clock clock, expresso.AbsoluteTime destination )
```

- **Usage**

- * Convert this time to an absolute time. For a `RelativeTime`, this involve adding the clocks notion of now to this interval. The sum is stored in the `destination` parameter. A new object is allocated if `destination` is null.

- **Parameters**

- * `clock` - if null, `Clock.getRealTimeClock()` is used.
- * `destination` - to store the result of the sum.

- **Returns** - the `AbsoluteTime` representing the current time plus this.

- *add*

```
public RelativeTime add( long millis, int nanos )
```

- **Usage**

- * Add a specific number of milli and nano seconds to this. A new object is allocated if `destination` is null, otherwise store there.

- **Parameters**
 - * `millis` - milli seconds to add.
 - * `nanos` - nano seconds to add.
 - * `destination` - to store the result.
 - **Returns** - A new object containing the result.
-

- *add*

```
public RelativeTime add( long  millis, int  nanos, espresso.RelativeTime
destination )
```

- **Usage**
 - * Return this + time. Add a specific number of milli and nano seconds to this. A new object is allocated if destination is null, otherwise store there.
 - **Parameters**
 - * `millis` - milli seconds to add.
 - * `nanos` - nano seconds to add.
 - * `destination` - to place the result in
 - **Returns** - the result.
-

- *add*

```
public RelativeTime add( espresso.RelativeTime  time )
```

- **Usage**
 - * Return this + time. A new object is allocated for the result.
 - **Parameters**
 - * `time` - the time to add to this.
 - **Returns** - the result.
-

- *add*

```
public RelativeTime add( espresso.RelativeTime  time, espresso.RelativeTime
destination )
```

- **Usage**
 - * Return this + time. If destination is non-null, the result is placed there and dest is returned. Otherwise a new object is allocated for the result.
 - **Parameters**
 - * `time` - the time to add to this.
 - * `destination` - to place the result in.
 - **Returns** - the result.
-

- *relative*

```
public RelativeTime relative( espresso.Clock  clock )
```

- **Usage**
 - * Return this.
 - **Returns** - this.
-

- *relative*

```
public RelativeTime relative( espresso.Clock  clock, espresso.RelativeTime
destination )
```

- **Usage**
 - * If destination is not null, set it to the same value as this and return it, if destination is null, return this.
 - **Returns** - destination modified to the same value as this, or this if destination is null.
-

- *subtract*

```
public final RelativeTime subtract( espresso.RelativeTime time )
```

- **Usage**
 - * Return this - time.
 - **Parameters**
 - * **time** - relative time to subtract from this Returns: this-time. A new object is allocated for the result.
-

- *subtract*

```
public final RelativeTime subtract( espresso.RelativeTime time,
espresso.RelativeTime destination )
```

- **Usage**
 - * Subtract time to this. If the destination parameter is not null, set it to the result, else a new object is allocated for the result. This is not modified.
 - **Parameters**
 - * **time** - relative time to subtract from this.
 - * **destination** - the RelativeTime that will be set to the result.
 - **Returns** - destination if not null, else the newly allocated object with the result of subtracting time to this.
-

- *toString*

```
public String toString( )
```

- **Usage**
 - * Return a new string representing a printable version of this time.
- **Returns** - a newly allocated String object representing this time.

METHODS INHERITED FROM CLASS `espresso.HighResolutionTime`

(in 1.2.11, page 26)

- *absolute*

```
public abstract AbsoluteTime absolute( espresso.Clock clock )
```

- **Usage**
 - * Convert this time to an absolute time, relative to some clock. Convenient for situations where you really need an absolute time. Allocates a destination object if necessary. See the derived class comments for more specific information.
 - **Parameters**
 - * **clock** - This clock is used to convert this time into absolute time.
-

- *absolute*

```
public abstract AbsoluteTime absolute( espresso.Clock clock, espresso.AbsoluteTime dest )
```

- **Usage**

- * Convert this time to an absolute time, relative to some clock. Convenient for situations where you really need an absolute time. Allocates a destination object if necessary. See the derived class comments for more specific information.

– **Parameters**

- * `clock` - This clock is used to convert this time into absolute time.
- * `dest` - if null, a new object is created and returned as result, else `dest` is returned.

- *compareTo*

```
public int compareTo( espresso.HighResolutionTime time )
```

– **Usage**

- * Compares this `HighResolutionTime` with the specified `HighResolutionTime`.

– **Parameters**

- * `time` - compares with this time.

- *compareTo*

```
public int compareTo( java.lang.Object time )
```

– **Usage**

- * For the `Comparable` interface. Specified By:
`java.lang.Comparable.compareTo(java.lang.Object)` in interface `java.lang.Comparable`

- *equals*

```
public boolean equals( espresso.HighResolutionTime time )
```

– **Usage**

- * Returns true if the argument object has the same values as this.

– **Parameters**

- * `time` - Values are compared to this.

- *equals*

```
public boolean equals( java.lang.Object object )
```

– **Usage**

- * Returns true if the argument is a `HighResolutionTime` reference and has the same values as this.

– **Parameters**

- * `object` - Values are compared to this.

- *getMilliseconds*

```
public final long getMilliseconds( )
```

– **Usage**

- * Returns the milliseconds component of this.

– **Parameters**

- * `The` - milliseconds component of the time represented by this.

- *getNanoseconds*

```
public final int getNanoseconds( )
```

– **Usage**

- * Returns the nanoseconds component of this.

– **Parameters**

- * `The` - nanoseconds component of the time represented by this.

- *hashCode*

```
public int hashCode( )
```

- *relative*

```
public abstract RelativeTime relative( espresso.Clock clock )
```

- **Usage**

- * Convert this time to a relative time, relative to the specified clock. Convenient for situations where you really need an relative time. Allocates a destination object if necessary. See the derived class comments for more specific information.

- **Parameters**

- * `clock` - This clock is used to convert this time into relative time.
-

- *relative*

```
public abstract RelativeTime relative( espresso.Clock clock, espresso.RelativeTime dest )
```

- **Usage**

- * Convert this time to a relative time, relative to the specified clock. Convenient for situations where you really need an relative time. Allocates a destination object if necessary. See the derived class comments for more specific information.

- **Parameters**

- * `clock` - This clock is used to convert this time into relative time.
 - * `dest` - if null, a new object is created and returned as result, else dest is returned.
-

- *set*

```
public void set( espresso.HighResolutionTime time )
```

- **Usage**

- * Changes the time represented by the argument to the same value as the one given in parameter.

- **Parameters**

- * `time` - The HighResolutionTime that will be used to set the time of this object.
-

- *set*

```
public void set( long millis )
```

- **Usage**

- * Sets the millisecond component of this to the given argument.
-

- *set*

```
public void set( long millis, int nanos )
```

- **Usage**

- * Sets the millisecond and nanosecond components of this.

- **Parameters**

- * `millis` - Value to set millisecond part of this. If millis is negative the millisecond value of this is set to negative value. Although logically this may represent time before the epoch, invalid results may occur if a HighResolutionTime representing time before the epoch is given as a parameter to the methods.
- * `nanos` - Value to set nanosecond part of this. If nanos is negative the millisecond value of this is set to negative value. Although logically this may represent time before the epoch, invalid results may occur if a HighResolutionTime representing time before the epoch is given as a parameter to the methods.

1.2.27 CLASS ReleaseParameters

The abstract top-level class for release characteristics of threads.

Ravenscar-Java (YCS 342) non-compliance:

- NC008: Ravenscar-Java ReleaseParameters class is defined as follows:

```
public class ReleaseParameters
{
    Protected ReleaseParameters();
}
```

Espresso Software Requirements Traceability:

- Thread04
- Thread06
- Thread08

DECLARATION

```
public abstract class ReleaseParameters
extends java.lang.Object
```

1.2.28 CLASS Scheduler

An instance of Scheduler manages the execution of schedulable objects. Fixed priority preemptive scheduling policy is assumed.

Ravenscar-Java (YCS 342) non-compliance:

- None

Espresso Software Requirements Traceability:

- Thread03

DECLARATION

```
public abstract class Scheduler
extends java.lang.Object
```

CONSTRUCTORS

- *Scheduler*
`public Scheduler()`

1.2.29 CLASS *SchedulingParameters*

The subclass *PriorityParameters* provide the parameters to be used by the scheduler. Changes to the values in a parameters object affects the scheduling behavior of all the schedulable to which it is bound.

Ravenscar-Java (YCS 342) non-compliance:

- None

Expresso Software Requirements Traceability:

- Thread03

DECLARATION

```
public abstract class SchedulingParameters
extends java.lang.Object
```

1.2.30 CLASS *ScopedMemory*

ScopedMemory is the abstract base class of all classes dealing with representations of memory spaces with a limited lifetime.

Ravenscar-Java (YCS 342) non-compliance:

- NC005: Ravenscar-Java *ScopedMemory* class is defined as follows:

```
public abstract class ScopedMemory extends MemoryArea
{
    public ScopedMemory (long size);
    public ScopedMemory (SizeEstimator size);
    public void enter();
    public int getReferenceCount();
}
```

Expresso Software Requirements Traceability:

- Mem01
- Mem03
- Mem04
- Mem05

DECLARATION

```
public abstract class ScopedMemory
extends expresso.MemoryArea
```

METHODS

- *enter*

```
public void enter( java.lang.Runnable logic )
```

- **Usage**

- * Associate this memory area to the current real-time thread for the duration of the execution of the run() method of the given java.lang.Runnable. During this bound period of execution, all objects are allocated from this ScopedMemory area until another one takes effect, or the enter() method is exited. ScopedMemory areas are not sharable between threads.

- **Parameters**

- * **logic** - a Runnable object.

- **Exceptions**

- * **expresso.ExpressoNestedScopeException** - if the current thread is already in a ScopedMemory area.
- * **expresso.ExpressoWrongThreadException** - if this ScopedMemory object is not associated to the current thread (see **setAssociatedThread(RealtimeThread)**).

-
- *memoryConsumed*

```
public final long memoryConsumed( )
```

- **Returns** - the number of bytes in use of this ScopedMemory area.

-
- *memoryRemaining*

```
public final long memoryRemaining( )
```

- **Returns** - the number of not used bytes of this ScopedMemory area.

- *setAssociatedThread*

```
public void setAssociatedThread( espresso.RealtimeThread thread )
```

– **Usage**

- * Set the associated thread of this ScopedMemory. This is the only one thread that will be allowed to enter this ScopedMemory.

FIXME: this method is not defined in the RTSJ !

– **Exceptions**

- * `java.lang.IllegalArgumentException` - if this ScopedMemory object is already associated to a thread.
- * `expresso.ExpressoIllegalPhaseException` - if this is not the initialization phase of the application.

-
- *size*

```
public final long size( )
```

- **Returns** - the size of this ScopedMemory area in bytes.

METHODS INHERITED FROM CLASS `expresso.MemoryArea`

(in 1.2.16, page 35)

- *memoryConsumed*

```
public abstract long memoryConsumed( )
```

- *memoryRemaining*

```
public abstract long memoryRemaining( )
```

- *size*

```
public abstract long size( )
```

- **Returns** - The size of the memory area in bytes.

1.2.31 CLASS `SizeEstimator`

This is a convenient class to help people figure out how much memory they need.

Ravenscar-Java (YCS 342) non-compliance:

- None

Expresso Software Requirements Traceability:

- Mem03

DECLARATION

```
public final class SizeEstimator
extends java.lang.Object
```

CONSTRUCTORS

- *SizeEstimator*
public SizeEstimator()
 - **Usage**
 - * Construct an empty size estimator.

METHODS

- *getEstimate*
public int getEstimate()
 - **Usage**
 - * Return the size estimate of all the elements that have been reserved in this SizeEstimator.
 - **Returns** - the size estimation in bytes units.

- *reserve*
public void reserve(java.lang.Class c, int n)
 - **Usage**
 - * Account this estimator for the size of n element of the class c.
 - **Parameters**
 - * **c** - the class descriptor for the element to reserve size for.
 - * **n** - the number of instance of class c element to reserve size for.

- *reserve*
public void reserve(espresso.SizeEstimator s)
 - **Usage**
 - * Account this estimator for the size estimated currently by the specified estimator.
 - **Parameters**
 - * **s** - a SizeEstimator

- *reserve*
public void reserve(espresso.SizeEstimator s, int n)
 - **Usage**
 - * Account this estimator for n times the size estimated currently by the specified estimator.
 - **Parameters**
 - * **s** - a SizeEstimator
 - * **n** - an integer.

1.2.32 CLASS **SporadicEvent**

Sporadic events are minimal inter-arrival time-bounded asynchronous events.

Ravenscar-Java (YCS 342) non-compliance:

- None

Espresso Software Requirements Traceability:

- Thread06
- Thread08

DECLARATION

```
public class SporadicEvent
extends espresso.AsyncEvent
```

CONSTRUCTORS

- *SporadicEvent*
public **SporadicEvent**(espresso.SporadicEventHandler handler)

METHODS

- *fire*
public void **fire**()

METHODS INHERITED FROM CLASS espresso.AsyncEvent

(in 1.2.2, page 11)

- *fire*
public void **fire**()

– Usage

* Fire (schedule the run() methods of) the handlers associated with this event.

1.2.33 CLASS *SporadicEventHandler*

SporadicEventHandler are *BoundAsyncEventHandler* with *ReleaseParameters* of type *SporadicParameters*.

Ravenscar-Java (YCS 342) non-compliance:

- NC017: Ravenscar-Java *BoundAsyncEventHandler* class is defined as follows:


```
public class SporadicEventHandler extends BoundAsyncEventHandler
{
    SporadicEventHandler(PriorityParameters pri,
                        SporadicParameters spor);
    SporadicEventHandler(PriorityParameters pri,
                        SporadicParameters spor,
                        java.lang.Runnable);
    public void handleAsyncEvent();
}
```

Expresso Software Requirements Traceability:

- Thread06
- Thread08

DECLARATION

<pre>public class SporadicEventHandler extends espresso.BoundAsyncEventHandler</pre>

CONSTRUCTORS

- *SporadicEventHandler*

```
public SporadicEventHandler( espresso.PriorityParameters pp,
    espresso.SporadicParameters sp, espresso.LTMemory ma )
```
- *SporadicEventHandler*

```
public SporadicEventHandler( espresso.PriorityParameters pp,
    espresso.SporadicParameters sp, espresso.LTMemory ma, java.lang.Runnable
    logic )
```

METHODS INHERITED FROM CLASS *espresso.BoundAsyncEventHandler*

(in 1.2.4, page 14)

METHODS INHERITED FROM CLASS `expresso.AsyncEventHandler`

(in 1.2.3, page 12)

- *handleAsyncEvent*

```
protected void handleAsyncEvent( )
```

– **Usage**

- * If this handler was constructed using a separate Runnable logic object, then that Runnable object's run method is called; This method will be invoked repeatedly while fireCount is greater than zero.

-
- *run*

```
public final void run( )
```

– **Usage**

- * This method invokes `handleAsyncEvent()` repeatedly while the fire count is greater than zero. Applications cannot override this method and should thus override `handleAsyncEvent()` in subclasses with the logic of the handler.

1.2.34 CLASS `SporadicInterrupt`

Sporadic interrupts like sporadic events which are minimal inter-arrival time-bounded asynchronous events.

Ravenscar-Java (YCS 342) non-compliance:

- NC014: Ravenscar-Java `SporadicInterrupt` class is defined as follows:

```
public class SporadicInterrupt extends AsyncEvent
{
    public SporadicInterrupt(SporadicEventHandler handler,
                             java.lang.String happening);
}
```

Expresso Software Requirements Traceability:

- Thread06
- Thread08

DECLARATION

<pre>public class <code>SporadicInterrupt</code> extends <code>expresso.AsyncEvent</code></pre>
--

METHODS INHERITED FROM CLASS `expresso.AsyncEvent`

(in 1.2.2, page 11)

- *fire*
`public void fire()`
 - **Usage**
 - * Fire (schedule the `run()` methods of) the handlers associated with this event.

1.2.35 CLASS *SporadicParameters*

Expresso schedulable objects are threads. An instance of *SporadicParameters* class is bound to a given sporadic thread. An instance of *SporadicParameters* class defines release characteristics of the thread to which it is bound.

Ravenscar-Java (YCS 342) non-compliance:

- NC010: The Ravenscar-Java modifier for method `RelativeTime()` is protected

Expresso Software Requirements Traceability:

- Thread06
- Thread08

DECLARATION

```
public class SporadicParameters
extends espresso.ReleaseParameters
```

CONSTRUCTORS

- *SporadicParameters*
`public SporadicParameters(espresso.RelativeTime minInterarrival)`

METHODS

- *getminInterarrival*
`public RelativeTime getminInterarrival()`

METHODS INHERITED FROM CLASS `expresso.ReleaseParameters`

(in 1.2.27, page 53)