4.9 The JMF time model

4.9.1 Time

The JMF keeps track of the time using the class "Time" [userguide]. A Time object represents a particular instant of time in the time axis. Basically the Time object uses a field of the data type long. This field holds the value of time that has passed from a time origin. The value of time is maintained to nanosecond precision. You can construct a Time object using one of its following two constructors. One constructor takes as parameter the value of the time to nanosecond precision and the other constructor takes to second precision. The constructors are

public Time(double seconds) , and

public Time(long nanoseconds).

The methods

long getNanoseconds( ) , and

double getSeconds( ) ,

can be used to get the value of the time from the Time object to either nanosecond precision or second precision respectively.

4.9.2 TimeBase

The interface TimeBase abstracts a constantly ticking source of time. TimeBase is analogous to an energized crystal oscillator in a digital clock. An energized crystal oscillator generates clock pulses at a certain frequency, that is dependent upon the physical characteristics of the crystal. An energized crystal oscillator can neither be paused and then restarted (while the oscillator remains energized), nor can we change the rate of the oscillator. Likewise the TimeBase does not allow the user to control the generation of the timing signals. In essence TimeBase is just an uncontrolled source of clock ticks.

JMF provides a default implementation of the TimeBase interface. It is called the SystemTimeBase, that presumably makes use of the timing information provided by the operating system. The method

public Time getTime( )

of TimeBase returns a Time object representing the time instant at which this method is invoked. The returned Time object can then be used to get the value of time to second or nanosecond precision. Alternatively the following convenient method

public long getNanoseconds( )

of TimeBase can be invoked to get the time directly in nanoseconds at any time instant.

4.9.3 Clock

We saw that the TimeBase is an uncontrolled source of time. On the other hand we would like to have a controlled source of time, and that is provided by the interface Clock. The JMF Clock is analogous to a digital clock in the sense that it uses the timing signals of a TimeBase (crystal oscillator) to derive its clock ticks. However the Clock allows the user to control the generation of the clock ticks. The Clock can be started, stopped, restarted and its rate can also

be adjusted.

The time that is kept track by the Clock is called the media-time. The time maintained by the TimeBase is called time-base time. Methods

public Time getMediaTime( )

and

public long getMediaNanoseconds( )

can be used on a clock to get its media time.

JMF interfaces like Players, Processors have to keep track of time and therefore they extend the Clock interface. Most of the JMF objects implementing the Clock interface use the default SystemTimeBase. However some Clock interfaces implemented by hardware encoders, decoders and renderers may use TimeBase represented by hardware clock. Methods

public void setTimeBase(TimeBase master)
throws IncompatibleTimeBaseException

and

public TimeBase getTimeBase( )

can be used to set and get the TimeBase of a Clock. In a multimedia presentation you may want the Clocks of two media players to be synchronized. For example you may want to synchronize the audio and video playback. Synchronization can be done by getting the TimeBase from one player and setting it as the TimeBase for another.

The rate of a Clock represents how fast the Clock is running with respect to its TimeBase. A rate of 1.0 represents normal playback rate while values more than 1.0 represent fast forwarding. A rate inbetween 0.0 and 1.0 denotes slow motion, and a negative rate denotes rewinding of the media. The rate of the Clock can be set using the method

public float setRate(float factor).

The returned value of the above method is the actual rate that is set. The rate of a Clock can be obtained through the method

public float getRate( ).

When a media presentation starts the media time of the Clock that controls the media presentation is initialized to zero. The maximum value of the media time of the Clock is the maximum time duration for which the media is supposed to be presented at the normal playback rate. You start the Clock by invoking its method

public void syncStart(Time at)

wherein you pass the time-base time at which the Clock should start. Suppose you want to start a media presentation from its begining at a given time-base time which we denote as tbst(time-base start time). You then invoke syncStart(tbst) on the Clock. When time-base time(tbt) reaches tbst, the clock starts and the media time (mt) of the clock is set to the media start time (mst). In this case mst is zero as it is the beginning of the presentation. Afterwards both the tbt and mt advances and they are related by the equation

mt = mst + (tbt – tbst) * rate . ---------- Eq. 1.1

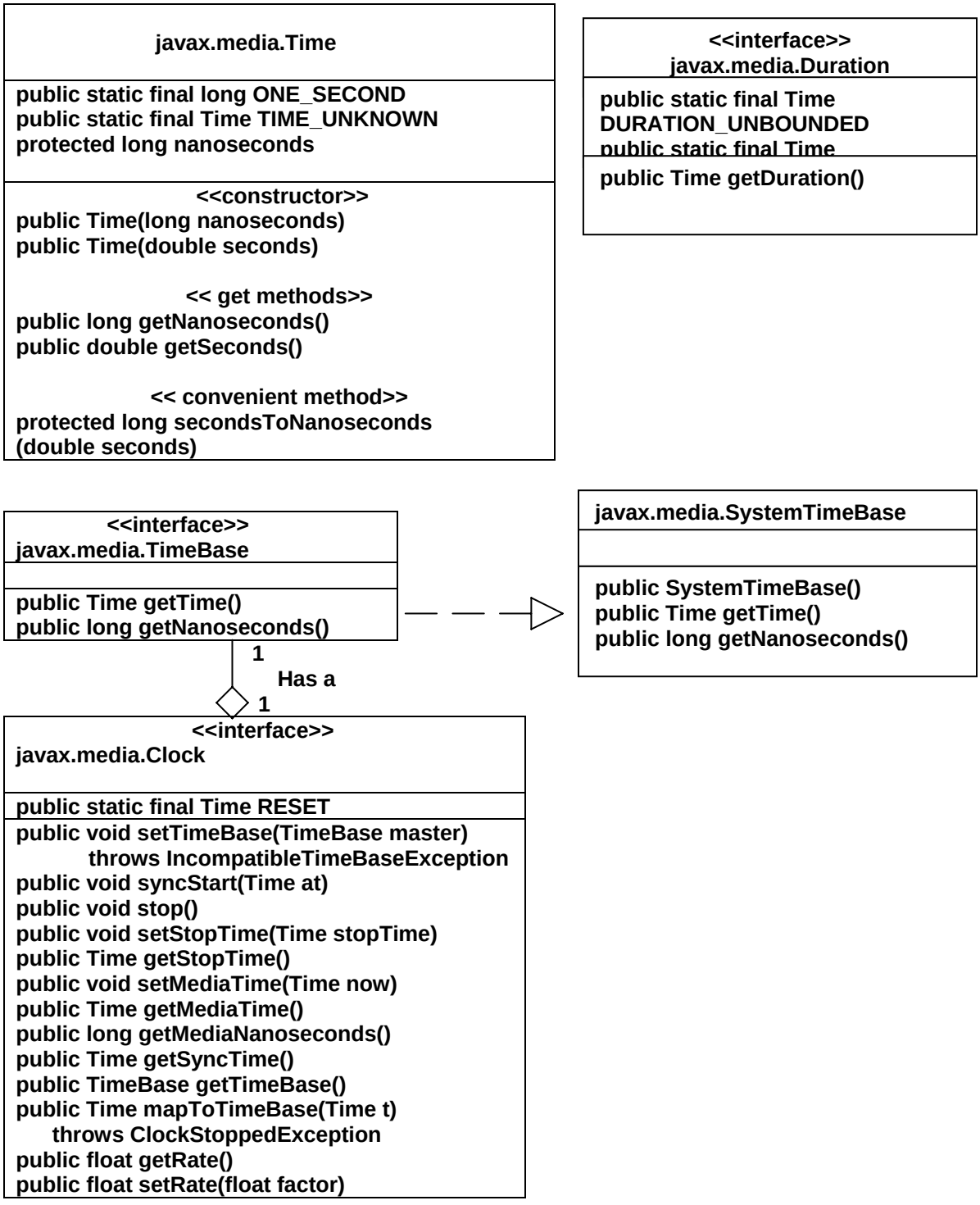

Figure 4.4 : The JMF Time model

Fig 4.3 shows this relation.

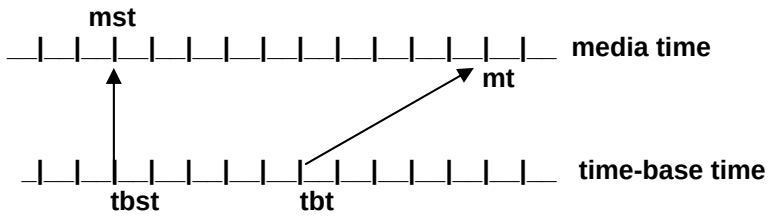

Figure 4.3 Relationship between time-base time and media time for rate = 2.0

Suppose you want to pause the media presentation then you can call the method

public void stop( )

which stops the clock immediately.

To pause the media presentation you can also call the method

public void setStopTime(Time stopTime).

By using the setStopTime method you set a time-base time at which you want to stop the clock and the presentation. Once the clock is stopped the media time is frozen, it does not advance. However the time-base time keeps on running at its own pace.

Now suppose you want to restart the presentation then you should invoke the method syncStart(Time newTBST) by supplying a new TBST. When the TBT reaches newTBST, the frozen media time becomes the media start time for the restarting procedure. The clock starts running from the newTBST, and the media time starts advancing and then onwards mt is again governed by the equation Eq. 4.1.

4.9.4 Duration

The Duration interface is implemented by those classes that can compute and advertise the duration of the media they are handling. For example Players and Processors extend Duration interface. Interface Duration has only one method

public Time getDuration( ),

which returns a Time object that indicates the duration of the media handled by that class. The returned duration is computed at the default playback rate. The method returns the Time object DURATION_UNKNOWN if the duration of the media cannot be computed