

Synchronization in NMM

Motama GmbH, Saarbruecken, Germany
(<http://www.motama.com>)

April 2010

Copyright (C) 2005-2010
Motama GmbH, Saarbruecken, Germany
<http://www.motama.com>

Permission is granted to copy, distribute and/or modify this document under the terms of the GNU Free Documentation License, Version 1.2 or any later version published by the Free Software Foundation; with the Invariant Sections being all sections, no Front-Cover Texts, and no Back-Cover Texts. A copy of the license can be found in the file COPYING.FDL.

THE DOCUMENT IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, TITLE AND NON-INFRINGEMENT. IN NO EVENT SHALL THE COPYRIGHT HOLDERS OR ANYONE DISTRIBUTING THE DOCUMENT BE LIABLE FOR ANY DAMAGES OR OTHER LIABILITY, WHETHER IN CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE DOCUMENT OR THE USE OR OTHER DEALINGS IN THE DOCUMENT.

1. Time Representation - Time and Interval

There are two types for representation of time in NMM. **Time** represents a point of time, **Interval** stands for a duration (which can be considered as the difference between two points of time.) Both Time and Interval have a precision of one nanosecond. They are internally stored as

```
struct Time {
    long int sec;
    long int nsec;
};

struct Interval {
    long int sec;
    long int nsec;
};
```

Since a set of standard operators comes along with the two types, and so it should not be necessary to manipulate them at that low level. There is also a type for a human readable time representation, `UserTime`.

```
struct UserTime {
    int hour;
    int min;
    int sec;
    int msec;
};
```

2. How to get the Time - Clock and TimedElement

Objects that should have access to a time source are inherited from **TimedElement**. All `TimedElement`s of one application share one clock and therefore have the same time. They can get the time with the method

```
Time getTime();
```

They don't have to care about the creation or destruction of the clock.

3. Time Information in the Data Stream - Timestamp

In lots of applications it is necessary to send some time information with the data stream. For this reason the class `Message` has a timestamp. A timestamp is build up like this:

```
struct Timestamp {
    Time sync_time;
    long int stream_counter;
    bool is_valid;
};
```

The `sync_time` carries the time information. It should be usually mark the beginning time of the media data (for example for an audio buffer of 0.02 seconds duration). The `stream_counter` simply counts the data buffer in the stream. The flag `is_valid` indicates if the time value has been set correctly. This is necessary because you do not always have enough information to timestamp each outgoing buffer correctly. With this flag you can indicate that a buffer's timestamp contains no useful time information. You can get and set a message's timestamp with the two methods

```
void setTimestamp(const Timestamp timestamp);
Timestamp getTimestamp();
```

from the class `Message`.

4. How to create Timestamps - StreamTimer

Nodes that want to create timestamps for a data stream can use a stream-timer. A stream-timer has two different modes, `REAL_TIME` and `CONST_RATE`. You can choose the mode with the method

```
Result setMode(const Mode mode);
```

In the `REAL_TIME` - Mode the stream-timer looks at the common clock to create the timestamps, in the `CONST_RATE` - Mode it computes the timestamps regarding to the framerate (set by the

```
Result setRate(const float rate);  
Result setInterval(const Interval interval);
```

methods). The timestamps are set into the messages with

```
Result setTimestamp(Message* message);
```

5. Sink Nodes - GenericSyncSinkNode

All sink nodes that should be synchronized are inherited from `GenericSyncSinkNode`. Instead of the `process` - or `procuce` - methods from the other Generics, these nodes have the methods

```
prepareBuffer();  
presentBuffer();
```

In the `prepare`-method, all time-wasting preparations for the presentation of the buffer should happen. In lots of cases, this method is not used and then simply should not be overwritten. In the `present`-method the presentation should be as soon as possible.

6. Events related to Synchronization

At the moment there are two instream events that refer to synchronization. They are called

```
setSynchronized, and  
syncReset.
```

The event `setSynchronized` has only an effect on synchronized sink nodes (i.e. all subclasses of `GenericSyncSinkNode`). The `instream` event has the same functionality as `setSynchronized(true)` and `setSynchronized(false)`. This way it is possible to disable or enable the synchronization after or before the presentation of a certain buffer. The second event, `syncReset`, is also handled by many other nodes (e.g. `MPEGVideoDecodeNode`, `MPEGAudioDecodeNode`, `AC3DecodeNode` etc.). It is used to indicate that the parameters of synchronization should be reset. For example, this is necessary if you switch to another channel on TV or if you choose a new chapter in the DVD application.

7. Synchronization in the Application Code

A synchronizer like the **MultiAudioVideoSynchronizer** has to be created and connected to the audio and video sinks. This is done by the following lines of code:

```
MultiAudioVideoSynchronizer av_sync;
IMultiAudioVideoSynchronizer_var sync
    (av_sync.getCheckedInterface< IMultiAudioVideoSynchronizer >());

ISynchronizedSink_var audio_sync
    (audio_play->getParentObject()->getCheckedInterface< ISynchronizedSink >());
ISynchronizedSink_var video_sync
    (display->getParentObject()->getCheckedInterface< ISynchronizedSink >());

ISinkController_var audio_controller(audio_sync->getController());
ISinkController_var video_controller(video_sync->getController());

sync->addAudioSinkController(audio_controller.get());
sync->addVideoSinkController(video_controller.get());
```

Or, if you are using `GraphDescriptions` to set up a flow graph, following lines are required to enable the synchronization.

```
ClientRegistry& registry = app-> getRegistry();
registry.requestGraph(graph);

graph.connectSinkSynchronizer();
```

Note: Most of the sink nodes have an input queue. In general, synchronization only works if this queue's mode is set to `MODE_SUSPEND`.