```
/
*+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
++++++++++++

Copyright (c) 2001 BeComm Corporation

Filename:

    sosisampleclock.h

Group Name:

    Sample Clock Interface

Group Overview:

    A "Sample Clock" provides a mechanism for synchronizing
    two streams of multimedia.

    Each stream is characterized
    by a frequency and divisor to get the
    result in samples per millisecond.
    For instance a 44100Hz
    audio stream has a frequency of 44100Hz and a divisor
    of 1000 (44100 / 1000 = 4.41 samples per millisecond).
    A 25 frame-per-second video stream has a frequency of
    25/1000 frames per millisecond.

    A sample clock contains the frequency and divisor for
    the stream, plus an instantaneous position mark,
    consisting of a wall-clock time (in milliseconds) and
    a sample position (in samples).

    Given this set of values we can project back to get the time
at
    which sample 0 would have been played - the "epoch" for this
stream.
    By comparing two epochs, we can determine the time shift
    required to bring them into synchronization.

    The epoch for a given stream may drift with time for several
    reasons; audio hardware playout rates do not always exactly
    match the system clock; in some cases the drift varies
depending on
    the level of interrupt activity on the system.  If a stream
is
    paused due to system contention or user intervention the
epoch
    will change.

    Typically one clock is identified as the master clock, and
    all other clocks attempt to match the master.  For instance
```

```
    when playing audio and video, the audio clock is typically
the
    master clock (because audio playout rate is usually regulated
    by an external DSP clock, and users are more perceptive to
changes
    in audio delivery rate).

    Periodically the playout position for the audio stream
    is copied into the master sample clock.  The video stream
then
    uses the difference between it's own sample clock and the
audio-derived
    master clock to determine the frame delivery time for each
frame.

    Overflow errors must be considered when dealing with samples.
    A 32-bit millisecond counter will wrap after 2^32
milliseconds
    which is approximately 50 days.  A 32-bit sample counter at
44100Hz
    will wrap around in 2^32/44100 seconds which is approximately
    27 hours.  If any addition or multiplication is performed on
sample
    values, overflow errors may occur much earlier.

Owner:

    Guy Carpenter (guyc) 21-Aug-2001

------------------------------------------------------------------
------------*/

/* cvsid: $Id: sosisampleclock.h,v 1.5 2001/09/13 20:50:24 guyc
Exp $ */

#ifndef _SOSISAMPLECLOCK_H_
#define _SOSISAMPLECLOCK_H_

#include <sosstrings.h>

/*++
Macro Description:
   Defines the global identifier for the SampleClock interface.
--*/

#define SOS_ISAMPLECLOCK_ID "sampleclock"

typedef struct _SOS_ISAMPLECLOCK SOS_ISAMPLECLOCK;

/*++
Prototype Name:
```

```
    SOS_ISAMPLECLOCK_FREQUENCYSET

Prototype Description:

    Set the sample frequency of the sample clock.  The
    frequency is expressed as a ratio of two 32-bit unsigned
    integers.

Parameters:

    SOS_ISAMPLECLOCK *          Interface - [in]
        Interface to object.

    SOS_UINT32                  Frequency - [in]
        Frequency numerator.

    SOS_UINT32                  Divisor   - [in]
        Frequency divisor.  May NOT be zero.

Return Value:

    SOS_STATUS -
        SOS_Success for successful completion.

        SOS_ErrorParameter if interface is invalid or the divisor
is zero.

--*/
typedef
SOS_STATUS
(*SOS_ISAMPLECLOCK_FREQUENCYSET)(
    SOS_ISAMPLECLOCK *          Interface,
    SOS_UINT32                  Frequency,
    SOS_UINT32                  Divisor
);

/*++
Prototype Name:

    SOS_ISAMPLECLOCK_FREQUENCYGET

Prototype Description:

    Returns the current frequency ratio.  If no call
    has been made to FrequencySet for this object, the
    values will be zero.

    Note that the sampleclock implementation may return
    different values for Frequency and Divisor than those
    originally set - the ratio should remain the same.

Parameters:
```

```
    SOS_ISAMPLECLOCK *          Interface - [in]
        Interface to object to be queried.

    SOS_UINT32 *                Frequency - [out]
        If not null, current frequency numerator will be
        stored at Frequency.

    SOS_UINT32 *                Divisor   - [out]
        If not null, current frequency divisor will be
        stored at Divisor.

Return Value:

    SOS_STATUS -
        SOS_Success for successful completion.

        SOS_ErrorParameter if the interface is invalid.

--*/
typedef
SOS_STATUS
(*SOS_ISAMPLECLOCK_FREQUENCYGET)(
    SOS_ISAMPLECLOCK *          Interface,
    SOS_UINT32 *                Frequency,
    SOS_UINT32 *                Divisor
);

/*++
Prototype Name:

    SOS_ISAMPLECLOCK_UPDATE

Prototype Description:

    Updates the sample position for this clock.  The stream
    position is expressed in the units defined by the call
    to FrequencySet - typically samples for audio, or frames
    for video.

Parameters:

    SOS_ISAMPLECLOCK *          Interface - [in]
        Interface to object being updated.

    SOS_CLOCK_TICK              Time      - [in]
        Clock time at which this sample was/will be delivered.

    SOS_UINT32                  Sample    - [in]
        Stream position.

Return Value:
```

```
    SOS_STATUS -
        SOS_Success for successful completion.

        SOS_ErrorParameter if interfaces is not valid.

--*/
typedef
SOS_STATUS
(*SOS_ISAMPLECLOCK_UPDATE)(
    SOS_ISAMPLECLOCK *          Interface,
    SOS_CLOCK_TICK              Time,
    SOS_UINT32                  Sample
);

/*++
Prototype Name:

    SOS_ISAMPLECLOCK_EPOCHGET

Prototype Description:

    Gets the nominal epoch (or stream-start time) for this
    clock.  The epoch is the time at which frame zero would
    have been delivered, computed from the last update
    time and frequency settings.  This value does not reflect
    the time frame/sample 0 was actually delivered, but rather
    the projected time based on the current frame position.

Parameters:

    SOS_ISAMPLECLOCK *          Interface - [in]
        Interface to clock object.

    SOS_CLOCK_TICK *            Epoch     - [out]
        Location to store the computed epoch.

Return Value:

    SOS_STATUS -
        SOS_Success for successful completion.

        SOS_ErrorParameter if the interface is invalid,
        or if Epoch is NULL, or if the frequency has not
        been set.

--*/
typedef
SOS_STATUS
(*SOS_ISAMPLECLOCK_EPOCHGET)(
    SOS_ISAMPLECLOCK *          Interface,
    SOS_CLOCK_TICK *            Epoch
```

DOCKET ALARM

Explore Litigation Insights

Docket Alarm provides insights to develop a more informed litigation strategy and the peace of mind of knowing you're on top of things.

Real-Time Litigation Alerts

Keep your litigation team up-to-date with **real-time alerts** and advanced team management tools built for the enterprise, all while greatly reducing PACER spend.

Our comprehensive service means we can handle Federal, State, and Administrative courts across the country.

Advanced Docket Research

With over 230 million records, Docket Alarm's cloud-native docket research platform finds what other services can't. Coverage includes Federal, State, plus PTAB, TTAB, ITC and NLRB decisions, all in one place.

Identify arguments that have been successful in the past with full text, pinpoint searching. Link to case law cited within any court document via Fastcase.

Analytics At Your Fingertips

Learn what happened the last time a particular judge, opposing counsel or company faced cases similar to yours.

Advanced out-of-the-box PTAB and TTAB analytics are always at your fingertips.

API

Docket Alarm offers a powerful API (application programming interface) to developers that want to integrate case filings into their apps.

LAW FIRMS
Build custom dashboards for your attorneys and clients with live data direct from the court.

Automate many repetitive legal tasks like conflict checks, document management, and marketing.

FINANCIAL INSTITUTIONS
Litigation and bankruptcy checks for companies and debtors.

E-DISCOVERY AND LEGAL VENDORS
Sync your system to PACER to automate legal marketing.