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

Copyright (c) 2001 BeComm Corporation

Filename:

    audiosync.c

Group Name:



Overview:

    This bead adjusts the audio stream by either
    dropping data, padding data or resampling data
    in an effort to make the path render clock match
    the path sample clock.

    This is done by computing the error in ms,
    smoothing the error over successive calls to
    the handler to reduce the noise in the signal.
    A damping factor is applied to correction to
    reduce the likelihood of over correction.  Note there
    is a significant amount of buffering between this
    bead and playout which adds latency to the feedback.
    Without damping it would be very possible to over
    correct and end up cycling.

    When a correction value has been found the
    stream is modified if necessary to
    bring the error back into tolerance.

    If the audio is very early, the packet is duplicated
    as necessary to delay it.

    If the audio is very late, part or all of the packet
    is discarded.

    If the audio is a little early or late, the packet
    is resampled to stretch or shrink it.

Notes:
```

```
45      Currently only supports 16-bit audio streams.
46      Supporting 8-bits would be relatively trivial.
47
48  Owner:
49
50      Guy Carpenter (guyc) 28-Sep-2001
51
52  ------------------------------------------------------------------------
…   ---*/
53
54  #define SOS_DEBUG_ZONE "/beads/audiosync"
55
56  #include <sosstrings.h>
57  #include <sosmultimedia.h>
58
59  SOS_SOURCE_VERSION (
60      "$Id: audiosync.c,v 1.12 2001/10/23 16:53:51 guyc Exp $"
61  );
62
63  /*++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
…   +++++
64  Named Constants
65  ------------------------------------------------------------------------
…   ---*/
66
67  /*
68   * Name of bead
69   */
70  static const char BEAD_NAME[] = "audiosync";
71
72  /*++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
…   +++++
73  Structs
74  ------------------------------------------------------------------------
…   ---*/
75
76  typedef struct {
77      SOS_UINT32              Size;   /* total size we can contain */
78      SOS_UINT32              Count;  /* current number of entries */
79      SOS_UINT32              Index;  /* index of next new value   */
80      SOS_INT32               Sum;    /* total of count values     */
81      SOS_INT32 *             Value;  /* count values              */
82  } SLIDING_AVG;
83
84  typedef struct {
```

```
85      SOS_BOOLEAN             ContextReady;
86      SOS_ISAMPLECLOCK *      MasterClock;
87      SOS_ISAMPLECLOCK *      RenderClock;
88      SOS_IAUDIOCONTEXT *     AudioContext;
89      SLIDING_AVG             AvgError;
90      SOS_BOOLEAN             First;
91  } AUDIOSYNC_CONTEXT;
92
93  #define EARLY_BAIL_THRESHOLD     5000 /* fail if > N ms early      */
94  #define  LATE_BAIL_THRESHOLD     500000 /* fail is < N ms late       */
95  #define EARLY_COPY_THRESHOLD     200 /* duplicate if > N ms early */
96  #define  LATE_DROP_THRESHOLD     200 /* drop if > N ms late       */
97  #define EARLY_RESAMPLE_THRESHOLD  4   /* resample if > N ms early  */
98  #define  LATE_RESAMPLE_THRESHOLD  4   /* resample if > N ms late   */
99
100 #define MAX_RESAMPLE_PERCENT 101      /* super-sample up to 1% */
101 #define MIN_RESAMPLE_PERCENT  99      /* sub-sample up to 1%   */
102
103 #define EARLY_THRESHOLD EARLY_RESAMPLE_THRESHOLD
104 #define LATE_THRESHOLD  LATE_RESAMPLE_THRESHOLD
105
106 #define SMOOTH_FACTOR  8  /* use average of last N errors */
107
108 /*
109  * REVISIT - can this value be determined empirically?
110  * Could be quite complicated to converge on the smallest
111  * value that doesn't over-compensate.
112  */
113 #define DAMPING_FACTOR 8  /* correction=N/DAMPING_FACTOR  */
114
115 /*++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
…+++++
116 Utility Functions
117 ++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
…+++*/
118
119 SOS_UINT32
120 Scale(
121     SOS_UINT32        Value,
122     SOS_UINT32        Numerator,
123     SOS_UINT32        Denominator
124 )
125 {
126     SOS_UINT32 whole  = Value / Denominator;
```

```
128     SOS_UINT32 result = whole * Numerator + remain * Numerator /
…   Denominator;
129     return result;
130 }
131 
132 
133 /*++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
…   +++++
134 Sliding Average Routines
135 ------------------------------------------------------------------------
…   ---*/
136 static
137 SOS_STATUS
138 SlidingAvg_Init(
139     SLIDING_AVG *         Avg,
140     SOS_UINT32            Size
141 )
142 {
143 //    SOS_Debug_StringPrint("SlidingAvg_Init\n");
144     Avg->Value = SOS_Mem_Alloc(sizeof(SOS_UINT32)*Size);
145     SOS_memset(Avg->Value, 0, sizeof(SLIDING_AVG));
146     Avg->Size = Size;
147     return Avg->Value ? SOS_Success : SOS_ErrorResourceAllocation;
148 }
149 
150 static
151 void
152 SlidingAvg_Uninit(
153     SLIDING_AVG *         Avg
154 )
155 {
156     SOS_Mem_Free(Avg->Value);
157 }
158 
159 
160 static
161 SOS_INT32
162 SlidingAvg_Add(
163     SLIDING_AVG *         Avg,
164     SOS_INT32             Value
165 )
166 {
167     SOS_DEBUGOUT_DETAIL(
168         "SlidingAvg_Add gets %ld\n",
```

```
170     );
171
172     if (Avg->Count==Avg->Size) {
173         Avg->Sum-=Avg->Value[Avg->Index];
174     } else {
175         Avg->Count++;
176     }
177     Avg->Value[Avg->Index]=Value;
178     Avg->Sum+=Value;
179
180     Avg->Index++;
181     if (Avg->Index==Avg->Size) {
182         Avg->Index=0;
183     }
184
185     SOS_DEBUGOUT_DETAIL(
186         "SlidingAvg_Add returns %ld/%lu=%ld\n",
187         Avg->Sum,
188         Avg->Count,
189         Avg->Sum/(SOS_INT32)Avg->Count
190     );
191     return Avg->Sum/(SOS_INT32)Avg->Count;
192 }
193
194
195 /*++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
…++++++
196 Context Stuff
197 ----------------------------------------------------------------------
…---*/
198
199 static
200 void
201 AudioSync_ContextDestroy(
202     AUDIOSYNC_CONTEXT *         Context
203 )
204 {
205     SOS_DEBUGOUT_FUNC_TRACE("AudioSync_ContextDestroy\n");
206
207     if (Context) {
208         SlidingAvg_Uninit(&(Context->AvgError));
209         SOS_Interface_Release(Context->MasterClock);
210         SOS_Interface_Release(Context->RenderClock);
211         SOS_Interface_Release(Context->AudioContext);
```

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.