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

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 1.16 2001/11/10 00:53:11 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 _SLIDING_AVG {
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
```

```
85  struct {
86      const char *          Name;
87      SOS_UINT32            Default;
88  } g_ParameterList[] = {
89      {"AudioSync/EarlyReject",       5000},
90      {"AudioSync/LateReject",      500000},
91      {"AudioSync/EarlyCopy",           50},
92      {"AudioSync/LateDrop",            50},
93      {"AudioSync/EarlyResample",        4},
94      {"AudioSync/LateResample",         4},
95      {"AudioSync/MinResamplePercent",  99},
96      {"AudioSync/MaxResamplePercent", 101},
97      {"AudioSync/Smoothing",            8},
98      {"AudioSync/Damping",              8},
99      {NULL,0}
100 };
101
102 typedef enum {
103     PARAM_EARLY_REJECT         = 0,
104     PARAM_LATE_REJECT          = 1,
105     PARAM_EARLY_COPY           = 2,
106     PARAM_LATE_DROP            = 3,
107     PARAM_EARLY_RESAMPLE       = 4,
108     PARAM_LATE_RESAMPLE        = 5,
109     PARAM_MIN_RESAMPLE_PERCENT = 6,
110     PARAM_MAX_RESAMPLE_PERCENT = 7,
111     PARAM_SMOOTHING            = 8,
112     PARAM_DAMPING              = 9,
113     PARAM_COUNT                = 10
114 } PARAM_INDEX;
115
116 typedef struct _AUDIOSYNC_CONTEXT {
117     SOS_BOOLEAN           ContextReady;
118     SOS_ISAMPLECLOCK *    MasterClock;
119     SOS_ISAMPLECLOCK *    RenderClock;
120     SOS_IAUDIOCONTEXT *   AudioContext;
121     SLIDING_AVG           AvgError;
122     SOS_BOOLEAN           First;
123     SOS_INT32             Param[PARAM_COUNT];
124 } AUDIOSYNC_CONTEXT;
125
126 /*++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
…++++++
127 Utility Functions
```

```
128… +++++*/
129
130  static
131  SOS_UINT32
132  Scale(
133      SOS_UINT32       Value,
134      SOS_UINT32       Numerator,
135      SOS_UINT32       Denominator
136  )
137  {
138      SOS_UINT32 whole  = Value / Denominator;
139      SOS_UINT32 remain = Value % Denominator;
140      SOS_UINT32 result = whole * Numerator + remain * Numerator /
…  Denominator;
141      return result;
142  }
143
144  /*++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
…  ++++++
145  Params
146  ++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
…  +++++*/
147  static
148  SOS_UINT32
149  IntParamGet(
150      SOS_PATH *            Path,
151      const char *          Name,
152      SOS_UINT32            Default
153  )
154  {
155      SOS_UINT32 value;
156      SOS_REGOBJECT *object = NULL;
157      if (SOS_FAILED(
158              SOS_Path_AttributeGet(
159                  Path,
160                  Name,
161                  &object)) ||
162          SOS_FAILED(
163              SOS_RegObject_Int32ValueGet(
164                  object,
165                  &value
166              ))) {
167          value = Default;
168      }
```

```
170     return value;
171 }
172 
173 static
174 void
175 ParamsGet(
176     SOS_PATH *          Path,
177     AUDIOSYNC_CONTEXT * Context
178 )
179 {
180     size_t i;
181     for (i=0;g_ParameterList[i].Name;i++) {
182         Context->Param[i] = IntParamGet(
183             Path,
184             g_ParameterList[i].Name,
185             g_ParameterList[i].Default
186         );
187     }
188 }
189 
190 /*++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
… ++++++
191 Sliding Average Routines
192 ----------------------------------------------------------------------
… ----*/
193 static
194 SOS_STATUS
195 SlidingAvg_Init(
196     SLIDING_AVG *         Avg,
197     SOS_UINT32            Size
198 )
199 {
200 //    SOS_Debug_StringPrint("SlidingAvg_Init\n");
201     Avg->Value = SOS_Mem_Alloc(sizeof(SOS_UINT32)*Size);
202     SOS_memset(Avg->Value, 0, sizeof(SLIDING_AVG));
203     Avg->Size = Size;
204     return Avg->Value ? SOS_Success : SOS_ErrorResourceAllocation;
205 }
206 
207 static
208 void
209 SlidingAvg_Uninit(
210     SLIDING_AVG *         Avg
211 )
```

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.