```
1
2
3       typedef double XFLOAT
4       typedef double OTA_FLOAT
5
6
7   {
8
9    SQFUNCS_POLQA_INTERNAL
10  {
11  using  std
12
13  static int const FFTSIZE = 1024
14
15  double matXMean(const double* in, int len)
16  {
17      double mean= 0.0
18      for (int i=0  i < len  i++)
19          mean += in[i]
20
21      mean /= len
22      return mean
23  }
24
25  SQSignal::SQSignal(XFLOAT const *fInputSignal,
26                     long          lInputLen,
27                     int           iInputRate,
28                     int           iBitResolution,
29                     MAT_HANDLE    inMatHandle,
30                     FILE*         pLogFile)
31
32  : mBitResolution        (iBitResolution),
33    mFrameSize            (-1),
34    mOverlapFac           (-1.0f),
35    mNrOfFrames           (-1),
36    mSamplingFreq         (iInputRate),
37    mOrigSampFreq         (iInputRate),
38    mNrOfSamples          (-1),
39    mStart                (-1),
40    mEnd                  (-1),
41    mNumPause             (-1),
42    mPauseCenter          (NULL),
43    mDCOffset             (0.0f),
44    mCurrentASL           (0.0f),
45    mUnalignedASL         (0.0f),
46    mASLBeforeFilt        (0.0f),
47    mCurNoiseLevel        (0.0f),
48    mUnalignedNoiseLev    (0.0f),
49    mNoiseLevBeforeFilt   (0.0f),
50    mSpeechThreshold      (0.0f),
51    mSpeechActivity       (-1.0f),
52    mData                 (NULL),
53    mEnv                  (NULL),
54    mVADprofile           (NULL),
55    mAvgSpeechSpec        (NULL),
56    mAvgNoiseSpec         (NULL),
57    mUsedFiltFreqResp     (NULL),
58    mIsInvalidSignal      (false),
59    mPreprocessed         (false),
60    mFiltered             (false),
61    mLevAligned           (false),
62    matHandle             (inMatHandle),
63    mpLogFile             (pLogFile)
64  {
65      OPTTRY
66      {
67          if (FFTSIZE != (int)(FRAME_LEN * STD_SAMPLING_RATE))
68              OPTTHROW ((string("ATTENTION: Value for FFTSIZE needs to changed to match current
```

```
    configuration!")))
69
70          if (fInputSignal   == NULL              ||
71              lInputLen      <= 0                 ||
72              iInputRate     <  MIN_SAMPLING_RATE ||
73              iBitResolution <= 2)
74              OPTTHROW(( string("Invalid input parameters.")))
75
76          mNrOfSamples      = lInputLen
77          mData             = (XFLOAT*)matMalloc(mNrOfSamples * sizeof(XFLOAT))
78          mUsedFiltFreqResp = (XFLOAT*)matMalloc(FILTERS_FREQRESP_LEN * sizeof(XFLOAT))
79          mMaxAmplitude     = (((MAX_AMP_32BIT) < (pow(2.0f, mBitResolution-1))) ? (MAX_AMP_32BIT) :
    (pow(2.0f, mBitResolution-1)))
80
81          vmov(fInputSignal, mData, mNrOfSamples)
82
83          matbSet(1.0f, mUsedFiltFreqResp, FILTERS_FREQRESP_LEN)
84
85      }
86      OPTCATCH((string errorMsg))
87      {
88          matFree(mData)
89          matFree(mUsedFiltFreqResp)
90          OPTTHROW ((string("ERROR in SQSignal::SQSignal(float*, long, int, int): " + errorMsg +
    "\n")))
91      }
92  }
93
94  SQSignal::SQSignal(SQSignal const   &Signal,
95                     int               iInputRate,
96                     int               iBitResolution)
97
98  : mBitResolution        (iBitResolution),
99    mFrameSize            (-1),
100   mOverlapFac           (-1.0f),
101   mNrOfFrames           (-1),
102   mSamplingFreq         (iInputRate),
103   mOrigSampFreq         (iInputRate),
104   mNrOfSamples          (-1),
105   mStart                (-1),
106   mEnd                  (-1),
107   mNumPause             (-1),
108   mPauseCenter          (NULL),
109   mDCOffset             (0.0f),
110   mCurrentASL           (0.0f),
111   mUnalignedASL         (0.0f),
112   mASLBeforeFilt        (0.0f),
113   mCurNoiseLevel        (0.0f),
114   mUnalignedNoiseLev    (0.0f),
115   mNoiseLevBeforeFilt   (0.0f),
116   mSpeechThreshold      (0.0f),
117   mSpeechActivity       (-1.0f),
118   mData                 (NULL),
119   mEnv                  (NULL),
120   mVADprofile           (NULL),
121   mAvgSpeechSpec        (NULL),
122   mAvgNoiseSpec         (NULL),
123   mUsedFiltFreqResp     (NULL),
124   mMaxAmplitude         (0.0f),
125   mIsInvalidSignal      (false),
126   mPreprocessed         (false),
127   mFiltered             (false),
128   mLevAligned           (false)
129 {
130     OPTTRY
131     {
132         if (FFTSIZE != (int)(FRAME_LEN * STD_SAMPLING_RATE))
133             OPTTHROW(( string("ATTENTION: Value for FFTSIZE needs to changed to match current
```

```
    configuration!")))
134
135         if (Signal.Data()       == NULL                 ||
136             Signal.NrOfSamples()<= 0                    ||
137             iInputRate          <  MIN_SAMPLING_RATE    ||
138             iBitResolution      <= 2)
139             OPTTHROW(( string("Invalid input parameters.")))
140
141         mNrOfSamples      = Signal.NrOfSamples()
142         mData             = (XFLOAT*)matMalloc(mNrOfSamples * sizeof(XFLOAT))
143         mUsedFiltFreqResp = (XFLOAT*)matMalloc(FILTERS_FREQRESP_LEN * sizeof(XFLOAT))
144         mMaxAmplitude     = (((MAX_AMP_32BIT) < (pow(2.0f, mBitResolution-1))) ? (MAX_AMP_32BIT) :
    (pow(2.0f, mBitResolution-1)))
145         mIsInvalidSignal  = Signal.IsInvalidSignal()
146         matHandle         = Signal.matHandle
147
148         vmov(Signal.Data(), mData, mNrOfSamples)
149         matbSet(1.0f, mUsedFiltFreqResp, FILTERS_FREQRESP_LEN)
150     }
151     OPTCATCH ((string errorMsg))
152     {
153         matFree(mData)
154         matFree(mUsedFiltFreqResp)
155         OPTTHROW ((string("ERROR in SQSignal::SQSignal(SQSignal, int, int): " + errorMsg + "\n")))
156     }
157 }
158 SQSignal::SQSignal(SQSignal const &Signal)
159
160 : mBitResolution        (-1),
161   mFrameSize            (-1),
162   mOverlapFac           (-1.0f),
163   mNrOfFrames           (-1),
164   mSamplingFreq         (-1),
165   mOrigSampFreq         (-1),
166   mNrOfSamples          (-1),
167   mStart                (-1),
168   mEnd                  (-1),
169   mNumPause             (-1),
170   mPauseCenter          (NULL),
171   mDCOffset             (0.0),
172   mCurrentASL           (0.0f),
173   mUnalignedASL         (0.0f),
174   mASLBeforeFilt        (0.0f),
175   mCurNoiseLevel        (0.0f),
176   mUnalignedNoiseLev    (0.0f),
177   mNoiseLevBeforeFilt   (0.0f),
178   mSpeechThreshold      (0.0f),
179   mSpeechActivity       (-1.0f),
180   mData                 (NULL),
181   mEnv                  (NULL),
182   mVADprofile           (NULL),
183   mAvgSpeechSpec        (NULL),
184   mAvgNoiseSpec         (NULL),
185   mUsedFiltFreqResp     (NULL),
186   mMaxAmplitude         (0.0),
187   mIsInvalidSignal      (false),
188   mPreprocessed         (false),
189   mFiltered             (false),
190   mLevAligned           (false)
191 {
192
193     OPTTRY
194     {
195         if (FFTSIZE != (int)(FRAME_LEN * STD_SAMPLING_RATE))
196             OPTTHROW(( string("ATTENTION: Value for FFTSIZE needs to changed to match current
    configuration!")))
197
198         if (Signal.Data()       == NULL ||
```

```
            Signal.NrOfSamples()<= 0    )
            OPTTHROW ((string("Invalid input parameters.")))

        mNrOfSamples        = Signal.NrOfSamples()
        mBitResolution      = Signal.BitResolution()
        mMaxAmplitude       = (((MAX_AMP_32BIT) < (pow(2.0f, mBitResolution-1))) ? (MAX_AMP_32BIT) :
(pow(2.0f, mBitResolution-1)))
        mFrameSize          = Signal.FrameSize()
        mOverlapFac         = Signal.OverlapFac()
        mNrOfFrames         = Signal.NrOfFrames()
        mSamplingFreq       = Signal.SamplingFreq()
        mOrigSampFreq       = Signal.OrigSampFreq()
        mStart              = Signal.Start()
        mEnd                = Signal.End()
        mNumPause           = Signal.NumPause()
        mDCOffset           = Signal.DCOffset()
        mCurrentASL         = Signal.CurrentASL()
        mUnalignedASL       = Signal.UnalignedASL()
        mASLBeforeFilt      = Signal.ASLBeforeFilt()
        mCurNoiseLevel      = Signal.CurrentNoiseLevel()
        mUnalignedNoiseLev  = Signal.UnalignedNoiseLevel()
        mNoiseLevBeforeFilt = Signal.NoiseLevelBeforeFilt()
        mSpeechThreshold    = Signal.SpeechThreshold()
        mSpeechActivity     = Signal.SpeechActivity()
        mIsInvalidSignal    = Signal.IsInvalidSignal()
        mPreprocessed       = Signal.Preprocessed()
        mFiltered           = Signal.Filtered()
        mLevAligned         = Signal.LevelAligned()
        matHandle           = Signal.matHandle
        mpLogFile           = Signal.mpLogFile

        mData = (XFLOAT*)matMalloc(mNrOfSamples * sizeof(XFLOAT))
        vmov(Signal.Data(), mData, mNrOfSamples)

        int const FFTSIZE = (int)(FRAME_LEN * mSamplingFreq)

        if (Signal.UsedFiltFreqResp() != NULL)
        {
            mUsedFiltFreqResp = (XFLOAT*)matMalloc(FILTERS_FREQRESP_LEN * sizeof(XFLOAT))
            vmov(Signal.UsedFiltFreqResp(), mUsedFiltFreqResp, FILTERS_FREQRESP_LEN)
        }
        if (Signal.Env() != NULL)
        {
            mEnv = (XFLOAT*)matMalloc(mNrOfFrames * sizeof(XFLOAT))
            vmov(Signal.Env(), mEnv, mNrOfFrames)
        }
        if (Signal.VADprofile() != NULL)
        {
            mVADprofile = (short*)matMalloc(mNrOfFrames * sizeof(short))
            sivmov(Signal.VADprofile(), mVADprofile, mNrOfFrames)
        }
        if (Signal.AvgSpeechSpec() != NULL)
        {
            mAvgSpeechSpec = (XFLOAT*)matMalloc(FFTSIZE/2 * sizeof(XFLOAT))
            vmov (Signal.AvgSpeechSpec(), mAvgSpeechSpec, FFTSIZE/2)
        }
        if (Signal.AvgNoiseSpec() != NULL)
        {
            mAvgNoiseSpec = (XFLOAT*)matMalloc(FFTSIZE/2 * sizeof(XFLOAT))
            vmov (Signal.AvgNoiseSpec(), mAvgNoiseSpec, FFTSIZE/2)
        }
        if (Signal.PauseCenter() != NULL)
        {
            mPauseCenter = (long*)matMalloc(mNumPause * sizeof(long))
            ivmov (Signal.PauseCenter(), mPauseCenter, mNumPause)
        }
    }
    OPTCATCH ((string errorMsg))
```

```
    {
        if(mData)
            matFree(mData)
        if(mUsedFiltFreqResp)
            matFree(mUsedFiltFreqResp)
        if(mEnv)
            matFree(mEnv)
        if(mVADprofile)
            matFree(mVADprofile)
        if(mAvgSpeechSpec)
            matFree(mAvgSpeechSpec)
        if(mAvgNoiseSpec)
            matFree(mAvgNoiseSpec)
        if(mPauseCenter)
            matFree(mPauseCenter)
        OPTTHROW(( string("ERROR in SQSignal::SQSignal(SQSignal): " + errorMsg + "\n")))
    }
}

SQSignal::~SQSignal()
{
    if(mData)
        matFree(mData)
    if(mEnv)
        matFree(mEnv)
    if(mVADprofile)
        matFree(mVADprofile)
    if(mAvgSpeechSpec)
        matFree(mAvgSpeechSpec)
    if(mAvgNoiseSpec)
        matFree(mAvgNoiseSpec)
    if(mUsedFiltFreqResp)
        matFree(mUsedFiltFreqResp)
    if(mPauseCenter)
        matFree(mPauseCenter)

    mData = mEnv = mAvgSpeechSpec = mAvgNoiseSpec = mUsedFiltFreqResp = NULL
    mVADprofile = NULL
    mPauseCenter = NULL
    matHandle = 0
}

SQSignal SQSignal::operator+= (SQSignal const *other)
{

    XFLOAT thisRatio  = this->mNrOfSamples   / (XFLOAT)(this->mNrOfSamples + other->NrOfSamples()),
           otherRatio = other->NrOfSamples() / (XFLOAT)(this->mNrOfSamples + other->NrOfSamples())
    vsmul(this->mUsedFiltFreqResp, thisRatio / otherRatio, this->mUsedFiltFreqResp,
FILTERS_FREQRESP_LEN)
    vadd (this->mUsedFiltFreqResp, other->UsedFiltFreqResp(), this->mUsedFiltFreqResp,
FILTERS_FREQRESP_LEN)
    vsmul(this->mUsedFiltFreqResp, otherRatio, this->mUsedFiltFreqResp, FILTERS_FREQRESP_LEN)

    XFLOAT *fCompleteData = (XFLOAT*)matMalloc((this->mNrOfSamples + other->NrOfSamples()) *
sizeof(XFLOAT))
    vmov (this->mData, fCompleteData, this->mNrOfSamples)
    vmov (other->mData, fCompleteData + this->mNrOfSamples, other->NrOfSamples())

    delete this->mData
    this->mData = fCompleteData
    this->mNrOfSamples += other->NrOfSamples()

    this->mFrameSize            = -1
    this->mOverlapFac           = -1
    this->mNrOfFrames           = -1
    this->mStart                = -1
    this->mEnd                  = -1
    this->mNumPause             = -1
```

```
331     this->mDCOffset             = -1
332     this->mCurrentASL           =  0
333     this->mUnalignedASL         =  0
334     this->mASLBeforeFilt        =  0
335     this->mCurNoiseLevel        =  0
336     this->mUnalignedNoiseLev    =  0
337     this->mNoiseLevBeforeFilt   =  0
338     this->mSpeechThreshold      =  0
339     this->mSpeechActivity       = -1
340     this->mIsInvalidSignal      = this->mIsInvalidSignal && other->IsInvalidSignal()
341     this->mPreprocessed         = false
342     this->mFiltered             = false
343     this->mLevAligned           = false
344
345     if (mEnv           != NULL)
346     {
347         delete mEnv
348         mEnv = NULL
349     }
350     if (mVADprofile    != NULL)
351     {
352         delete mVADprofile
353         mVADprofile = NULL
354     }
355     if (mAvgSpeechSpec != NULL)
356     {
357         delete mAvgSpeechSpec
358         mAvgSpeechSpec = NULL
359     }
360     if (mAvgNoiseSpec  != NULL)
361     {
362         delete mAvgNoiseSpec
363         mAvgNoiseSpec = NULL
364     }
365
366     return *this
367 }
368
369 void SQSignal::assign (SQSignal const *other)
370 {
371     this->mNrOfSamples          = other->NrOfSamples()
372     this->mBitResolution        = other->BitResolution()
373     this->mMaxAmplitude         = other->MaxAmplitude()
374     this->mSamplingFreq         = other->SamplingFreq()
375     this->mOrigSampFreq         = other->OrigSampFreq()
376     this->mFrameSize            = other->FrameSize()
377     this->mOverlapFac           = other->OverlapFac()
378     this->mNrOfFrames           = other->NrOfFrames()
379     this->mStart                = other->Start()
380     this->mEnd                  = other->End()
381     this->mNumPause             = other->NumPause()
382     this->mDCOffset             = other->DCOffset()
383     this->mCurrentASL           = other->CurrentASL()
384     this->mUnalignedASL         = other->UnalignedASL()
385     this->mASLBeforeFilt        = other->ASLBeforeFilt()
386     this->mCurNoiseLevel        = other->CurrentNoiseLevel()
387     this->mUnalignedNoiseLev    = other->UnalignedNoiseLevel()
388     this->mNoiseLevBeforeFilt   = other->NoiseLevelBeforeFilt()
389     this->mSpeechThreshold      = other->SpeechThreshold()
390     this->mSpeechActivity       = other->SpeechActivity()
391     this->mIsInvalidSignal      = other->IsInvalidSignal()
392     this->mPreprocessed         = other->Preprocessed()
393     this->mFiltered             = other->Filtered()
394     this->mLevAligned           = other->LevelAligned()
395
396     if (this->mData             != NULL) matFree(mData)             mData = NULL
397     if (this->mEnv              != NULL) matFree(mEnv)              mEnv  = NULL
398     if (this->mVADprofile       != NULL) matFree(mVADprofile)       mVADprofile = NULL
```

```
399     if (this->mAvgSpeechSpec    != NULL) matFree(mAvgSpeechSpec)     mAvgSpeechSpec = NULL
400     if (this->mAvgNoiseSpec     != NULL) matFree(mAvgNoiseSpec)      mAvgNoiseSpec = NULL
401     if (this->mUsedFiltFreqResp != NULL) matFree(mUsedFiltFreqResp)  mUsedFiltFreqResp = NULL
402     if (this->mPauseCenter      != NULL) matFree(mPauseCenter)       mPauseCenter = NULL
403
404     int const FFTSIZE = (int)(FRAME_LEN * this->mSamplingFreq)
405
406     if (other->Data() != NULL)
407     {
408         mData = (XFLOAT*)matMalloc(mNrOfSamples * sizeof(XFLOAT))
409         vmov(other->Data(), mData, mNrOfSamples)
410     }
411     if (other->Env() != NULL)
412     {
413         mEnv = (XFLOAT*)matMalloc(mNrOfFrames * sizeof(XFLOAT))
414         vmov(other->Env(), mEnv, mNrOfFrames)
415     }
416     if (other->VADprofile() != NULL)
417     {
418         mVADprofile = (short*)matMalloc(mNrOfFrames * sizeof(short))
419         sivmov(other->VADprofile(), mVADprofile, mNrOfFrames)
420     }
421     if (other->AvgSpeechSpec() != NULL)
422     {
423         mAvgSpeechSpec = (XFLOAT*)matMalloc(FFTSIZE/2 * sizeof(XFLOAT))
424         vmov (other->AvgSpeechSpec(), mAvgSpeechSpec, FFTSIZE/2)
425     }
426     if (other->AvgNoiseSpec() != NULL)
427     {
428         mAvgNoiseSpec = (XFLOAT*)matMalloc(FFTSIZE/2 * sizeof(XFLOAT))
429         vmov (other->AvgNoiseSpec(), mAvgNoiseSpec, FFTSIZE/2)
430     }
431     if (other->UsedFiltFreqResp() != NULL)
432     {
433         mUsedFiltFreqResp = (XFLOAT*)matMalloc(FILTERS_FREQRESP_LEN * sizeof(XFLOAT))
434         vmov (other->UsedFiltFreqResp(), mUsedFiltFreqResp, FILTERS_FREQRESP_LEN)
435     }
436     if (other->PauseCenter() != NULL)
437     {
438         mPauseCenter = (long*)matMalloc(mNumPause * sizeof(long))
439         ivmov (other->PauseCenter(), mPauseCenter, mNumPause)
440     }
441 }
442
443 void SQSignal::PreprocessingProperties (XFLOAT fEnvFrameLen,
444                                         XFLOAT fEnvOverlapRatio,
445                                         XFLOAT fEnvLowerLimdB,
446                                         XFLOAT * const maxSigLenBuff)
447 {
448     CalcEnvelope(fEnvFrameLen, fEnvOverlapRatio, fEnvLowerLimdB, maxSigLenBuff)
449     CalcASLandNoiseLevel(FRAME_LEN, maxSigLenBuff)
450     FindActBoundaries((((mCurNoiseLevel+(XFLOAT)6.0) < ((XFLOAT)-50.0)) ?
    (mCurNoiseLevel+(XFLOAT)6.0) : ((XFLOAT)-50.0)), 4, (XFLOAT)0.008, 6, false)
451     FindPauseCenter ((mSpeechThreshold + mCurrentASL) / 2)
452
453     mPreprocessed = true
454     return
455 }
456
457 void SQSignal::Preprocess(int   iTargetFs,
458                           XFLOAT fTargetASLdBov,
459                           XFLOAT fEnvFrameLen,
460                           XFLOAT fEnvOverlapRatio,
461                           XFLOAT fEnvLowerLimdB,
462                           int   appType,
463                           XFLOAT * const maxSigLenBuff,
464                           int   IRSSendfilter,
465                           FILE* pLogFile)
```

```
{
    OPTTRY
    {
        Resample(iTargetFs)
        RecalcDCOffset()
        vsadd(mData, -(mDCOffset * mMaxAmplitude / (XFLOAT)100.0), mData, mNrOfSamples)

        CalcEnvelope(fEnvFrameLen, fEnvOverlapRatio, fEnvLowerLimdB, maxSigLenBuff)

        CalcASLandNoiseLevel(FRAME_LEN, maxSigLenBuff)

        mFiltered = true && !mIsInvalidSignal

        CalcEnvelope(fEnvFrameLen, fEnvOverlapRatio, fEnvLowerLimdB, maxSigLenBuff)
        CalcASLandNoiseLevel(FRAME_LEN, maxSigLenBuff)
        FindActBoundaries((((mCurNoiseLevel + (XFLOAT)6.0) < ((XFLOAT)-50.0)) ? (mCurNoiseLevel +
(XFLOAT)6.0) : ((XFLOAT)-50.0)), 4, (XFLOAT)0.008, 6, false)
        FindPauseCenter ((mSpeechThreshold + mCurrentASL) / 2)
        if (fabs(fTargetASLdBov - SQ_SIGNAL_NO_LEVEL_ALIGN) > (XFLOAT)0.001)
            LevelAlign(fTargetASLdBov)
        mPreprocessed = true && !mIsInvalidSignal
        return
    }
    OPTCATCH( (string errorMsg))
    {
        errorMsg.insert(0, string("ERROR in SQSignal::Preprocess(): "))
        OPTTHROW(( string(errorMsg)))
    }
}

XFLOAT SQSignal::RecalcDCOffset()
{
    OPTTRY
    {
        if (mData == NULL || mBitResolution < 2 || mNrOfSamples < 1)
            OPTTHROW(-1)

        XFLOAT fTemp = (XFLOAT)0.0
        fTemp = matMean(mData, mNrOfSamples)
        mDCOffset = (XFLOAT)100.0 * fTemp / mMaxAmplitude

        long Offset = (long)(mDCOffset * (XFLOAT)1.0e9)
        mDCOffset = (XFLOAT)Offset / (XFLOAT)1.0e9
        return mDCOffset
    }
    OPTCATCH( (...))
    {
        OPTTHROW(( string("RecalcDCOffset failed.\n")))
    }
}

void SQSignal::CalcEnvelope(XFLOAT fFrameLen,
                            XFLOAT fOverlapRatio,
                            XFLOAT fLowerdBLim,
                            XFLOAT * const maxSigLenBuff)
{
    OPTTRY
    {
        if (mData == NULL)
            OPTTHROW(( string("No data in signal.")))
        if (mSamplingFreq < 1 || mNrOfSamples < 1 ||
            fOverlapRatio > 1.0f || fOverlapRatio <= 0.0f ||
            mBitResolution < 2)
            OPTTHROW(( string("Invalid signal parameters.")))

        mOverlapFac     = fOverlapRatio
        mFrameSize      = (int)(mSamplingFreq * fFrameLen)
        mNrOfFrames     = (int)( (XFLOAT)mNrOfSamples / (mFrameSize*mOverlapFac) ) - 1
```

```
533         if (mNrOfFrames < 1)
534         {
535             mIsInvalidSignal = true
536 
537             matFree(mEnv)
538             mEnv = NULL
539             return
540         }
541         if (mEnv == NULL)
542             mEnv = (XFLOAT*)matMalloc(mNrOfFrames * sizeof(XFLOAT))
543         else
544         {
545             matFree(mEnv)
546             mEnv = (XFLOAT*)matMalloc(mNrOfFrames * sizeof(XFLOAT))
547         }
548 
549         for (int i = 0  i < mNrOfFrames  i++)
550             rmvesq(&mData[(int)(i*mOverlapFac*mFrameSize)], &mEnv[i], mFrameSize)
551 
552         XFLOAT rmsFloor = (((dBtoRMS(fLowerdBLim)) > ((XFLOAT)1e-16f)) ? (dBtoRMS(fLowerdBLim)) :
    ((XFLOAT)1e-16f))
553         if (maxSigLenBuff == NULL)
554         {
555             vsdiv(mEnv, mMaxAmplitude, mEnv, mNrOfFrames)
556             matbThresh1(mEnv, mNrOfFrames, rmsFloor, MAT_LT)
557             vlog10(mEnv, mEnv, mNrOfFrames)
558             vsmul(mEnv, (XFLOAT)20.0, mEnv, mNrOfFrames)
559             vclip(mEnv, -fLowerdBLim, mEnv, mNrOfFrames)
560         }
561         else
562         {
563             vsdiv (mEnv,            mMaxAmplitude, mEnv,          mNrOfFrames)
564             matbThresh1(mEnv, mNrOfFrames, rmsFloor, MAT_LT)
565             vlog10(mEnv,            maxSigLenBuff, mNrOfFrames)
566             vsmul (maxSigLenBuff,   (XFLOAT)20.0,         mEnv,          mNrOfFrames)
567             vclip (mEnv,            -fLowerdBLim,  mEnv,          mNrOfFrames)
568         }
569 
570         return
571     }
572     OPTCATCH( (string errorMsg))
573     {
574         OPTTHROW(( string("ERROR in SQSignal::CalcEnvelope(): " + errorMsg + "\n")))
575     }
576 }
577 
578 void SQSignal::FindActBoundaries(XFLOAT fMinActdB,
579                                  int   iMinContActLen,
580                                  XFLOAT fFrameLenInSec,
581                                  int   iSafetyOffset,
582                                  bool  doUseVADprofile)
583 {
584     OPTTRY
585     {
586         if (mData == NULL)
587             OPTTHROW(( string("No data in signal.")))
588 
589         if (mIsInvalidSignal)
590         {
591 
592             return
593         }
594         if (mBitResolution < 2 || mSamplingFreq < 1 || mNrOfSamples < 1)
595             OPTTHROW(( string("Invalid signal parameters.")))
596 
597         if (doUseVADprofile && (mVADprofile == NULL || mNrOfFrames < 10))
598             OPTTHROW(( string("No or invalid VAD profile. Call CalcASLandNoiseLevel() first.")))
599 
```

```
600         if (!doUseVADprofile &&
601             (fMinActdB >= 0.0f || iMinContActLen < 0 ||
602             fFrameLenInSec < 0.0f || iSafetyOffset < 0))
603             OPTTHROW(( string("Invalid input arguments.")))
604 
605         if (doUseVADprofile)
606         {
607             int i, start, end
608 
609             for (i = 0  i < mNrOfFrames && mVADprofile[i] == SQ_VAD_NO_SPEECH  i++)
610             if (i < mNrOfFrames) start = i
611             else                 start = -1
612 
613             for (i = mNrOfFrames-1  i > start && mVADprofile[i] == SQ_VAD_NO_SPEECH  i--)
614             if (i > start)       end = i
615             else                 end = -1
616 
617             if (start > 0 && end > start && end < mNrOfFrames)
618             {
619                 mStart = ((((int) (start * mFrameSize * mOverlapFac)) > (0)) ? ((int) (start *
    mFrameSize * mOverlapFac)) : (0))
620                 mEnd   = ((((int)ceil( end * mFrameSize * mOverlapFac)) < (mNrOfSamples-1)) ?
    ((int)ceil( end * mFrameSize * mOverlapFac)) : (mNrOfSamples-1))
621                 return
622             }
623             else
624             {
625 
626             }
627         }
628 
629         XFLOAT const MIN_ACT_RMS     = pow((XFLOAT)10.0, fMinActdB / (XFLOAT)20.0) * mMaxAmplitude
630         int const   FINE_FRAME_SIZE = (int)(fFrameLenInSec * mSamplingFreq)
631 
632         int         iContActCounter = 0
633         long        i
634         XFLOAT      fTemp
635 
636         for (i = 0  i < mNrOfSamples - FINE_FRAME_SIZE  i += FINE_FRAME_SIZE)
637         {
638             rmvesq(&mData[i], &fTemp, FINE_FRAME_SIZE)
639             if (fTemp > MIN_ACT_RMS)    iContActCounter++
640             else                        iContActCounter = 0
641 
642             if (iContActCounter == iMinContActLen)
643                 break
644         }
645 
646         if (iContActCounter != iMinContActLen)
647         {
648             mStart = mEnd = -1
649             return
650         }
651 
652         mStart = i - (iContActCounter-1)*FINE_FRAME_SIZE
653         mStart = (((0) > (mStart - iSafetyOffset*FINE_FRAME_SIZE)) ? (0) : (mStart -
    iSafetyOffset*FINE_FRAME_SIZE))
654 
655         iContActCounter = 0
656         for (i = mNrOfSamples-FINE_FRAME_SIZE  i > mStart+FINE_FRAME_SIZE  i -= FINE_FRAME_SIZE)
657         {
658             rmvesq(&mData[i], &fTemp, FINE_FRAME_SIZE)
659             if (fTemp > MIN_ACT_RMS)    iContActCounter++
660             else                        iContActCounter = 0
661 
662             if (iContActCounter == iMinContActLen)
663                 break
664         }
```

```
665
666         if (iContActCounter != iMinContActLen)
667         {
668             mStart = mEnd = -1
669             return
670         }
671
672         mEnd = i + iContActCounter*FINE_FRAME_SIZE - 1
673         mEnd = (((mNrOfSamples-1) < (mEnd + iSafetyOffset*FINE_FRAME_SIZE)) ? (mNrOfSamples-1) :
    (mEnd + iSafetyOffset*FINE_FRAME_SIZE))
674
675         return
676     }
677     OPTCATCH( (string errorMsg))
678     {
679         OPTTHROW(( string("ERROR in SQSignal::FindActBoundaries(): " + errorMsg + "\n")))
680     }
681 }
682
683 void SQSignal::FindPauseCenter (XFLOAT fMinActivitydB)
684 {
685     OPTTRY
686     {
687         if (mIsInvalidSignal)
688         {
689
690             return
691         }
692         if (mData == NULL || mEnv == NULL)
693             OPTTHROW(( string("Invalid input data.")))
694
695         if(mPauseCenter)
696             matFree(mPauseCenter)
697         mPauseCenter = NULL
698
699         XFLOAT const MIN_PAUSE_LENGTH_S = 1.0
700         int   const MAX_NUM_PAUSE = MAX_NUM_SENTENCES - 1
701
702         bool inPause = false
703         int  pauseStart = 0
704         int  pauseLength = 0
705         int  numPause = 0
706         int  centerFrame[MAX_NUM_PAUSE] = {0}
707         int  firstFrame, lastFrame
708
709         mStart > 0 ? firstFrame = (int)((XFLOAT)mStart / (mFrameSize * mOverlapFac)) : firstFrame =
    0
710         mEnd   > 0 ? lastFrame  = (int)((XFLOAT)mEnd   / (mFrameSize * mOverlapFac)) : lastFrame  =
    mNrOfFrames
711         firstFrame = limit(firstFrame, 0, (int)mNrOfFrames-1)
712         lastFrame  = limit(lastFrame,  0, (int)mNrOfFrames)
713
714         for (int f = firstFrame  f < lastFrame  f++)
715         {
716             if (inPause == true)
717             {
718                 if (mEnv[f] > fMinActivitydB)
719                 {
720                     inPause = false
721                     pauseLength = f - pauseStart
722                     if (pauseLength > MIN_PAUSE_LENGTH_S * mSamplingFreq / (mFrameSize *
    mOverlapFac))
723                     {
724                         centerFrame[numPause] = pauseStart + (int)(0.5f*pauseLength)
725                         numPause ++
726                         if (numPause >= MAX_NUM_PAUSE)
727                             break
728                     }
```

```
729                 }
730             }
731             else
732             {
733                 if (mEnv[f] <= fMinActivitydB)
734                 {
735                     inPause = true
736                     pauseStart = f
737                 }
738             }
739         }
740 
741         mNumPause = numPause
742         if (mNumPause)
743         {
744             mPauseCenter = (long*)matMalloc(mNumPause * sizeof(long))
745             for (int p = 0  p < mNumPause  p++)
746             {
747                 if (centerFrame[p] > 0)
748                 {
749                     mPauseCenter[p] = round (centerFrame[p] * mFrameSize * mOverlapFac)
750                 }
751                 else
752                     OPTTHROW(( string ("Pause at position 0")))
753             }
754         }
755     }
756     OPTCATCH( (string errorMsg))
757     {
758         OPTTHROW(( string("ERROR in SQSignal::FindPauseCenter(): " + errorMsg + "\n")))
759     }
760     OPTCATCH( (...))
761     {
762         OPTTHROW(( string("ERROR in SQSignal::FindPauseCenter(): Unknown error.\n")))
763     }
764 }
765 
766 void SQSignal::CurrentSentence (int iSample, long *lStart, long *lEnd, short *sIndex)
767 {
768     *sIndex = -1
769     for (short s = 0  s < mNumPause  s++)
770     {
771         if (iSample < mPauseCenter[s])
772         {
773             *sIndex = s
774             s == 0 ? *lStart = 0 : *lStart = mPauseCenter[s-1]
775             *lEnd = mPauseCenter[s]
776             break
777         }
778     }
779     if (*sIndex == -1)
780     {
781         *sIndex = mNumPause
782         *lStart = mPauseCenter[mNumPause]
783         *lEnd   = mNrOfSamples
784     }
785 }
786 
787 void SQSignal::CurrentSentence (int iIdx, long *lStart, long *lEnd) const
788 {
789     if (mNumPause < 0)
790         OPTTHROW(( string ("Pauses not yet calculated.")))
791 
792     if (iIdx == 0)
793         *lStart = 0
794     else
795         *lStart = mPauseCenter[iIdx-1]
796 
```

```
797     if (iIdx == mNumPause)
798         *lEnd = mNrOfSamples
799     else
800         *lEnd = mPauseCenter[iIdx]
801
802     if (*lStart < 0 || *lEnd < 0 || *lStart > mNrOfSamples || *lEnd > mNrOfSamples)
803     {
804         OPTTHROW(( string ("Pauses invalid.")))
805     }
806 }
807
808 void SQSignal::CalcASLandNoiseLevel(XFLOAT fFrameLen, XFLOAT * const maxSigLenBuff)
809 {
810 OPTTRY
811     {
812         if (mIsInvalidSignal)
813         {
814
815             return
816         }
817         if (mBitResolution < 2 || mSamplingFreq < 1 || mData == NULL)
818             OPTTHROW(( string("Invalid input parameters.\n")))
819         if (mNrOfSamples < 10*mSamplingFreq*fFrameLen*FRAME_OVERLAP_RATIO)
820         {
821             mIsInvalidSignal = true
822             return
823         }
824
825         int newFrameSize = (int)(fFrameLen * mSamplingFreq)
826         if (mEnv == NULL || mFrameSize != newFrameSize || mOverlapFac != FRAME_OVERLAP_RATIO)
827             CalcEnvelope(fFrameLen, FRAME_OVERLAP_RATIO, MIN_LEVEL_DB, maxSigLenBuff)
828
829         if (mVADprofile == NULL)
830             mVADprofile = (short*)matMalloc(mNrOfFrames * sizeof(short))
831         else
832         {
833             matFree(mVADprofile)
834             mVADprofile = (short*)matMalloc(mNrOfFrames * sizeof(short))
835         }
836
837         if (SQcalcASLandNoise(mEnv,
838                               mVADprofile,
839                               mNrOfFrames,
840                               (int)(mFrameSize*mOverlapFac),
841                               MIN_LEVEL_DB,
842                               mSamplingFreq,
843                               &mSpeechActivity,
844                               &mCurrentASL,
845                               &mCurNoiseLevel,
846                               &mSpeechThreshold)
847             != SQ_NO_ERRORS)
848         {
849
850             rmvesq(mData, &mCurrentASL, mNrOfSamples)
851             mCurrentASL = 20.0f * log10((mCurrentASL+1e-16f) / (mMaxAmplitude+1e-16f))
852             mCurrentASL = mCurNoiseLevel = (((mCurrentASL) > (MIN_LEVEL_DB)) ? (mCurrentASL) :
    (MIN_LEVEL_DB))
853             mIsInvalidSignal = true
854         }
855
856         long x = (long)((XFLOAT)1.0e7*((((-200)) > (mCurrentASL)) ? ((-200)) : (mCurrentASL)))
857         mCurrentASL = (XFLOAT)x / (XFLOAT)1.0e7
858
859         x = (long)((XFLOAT)1.0e7*((((-200)) > (mCurNoiseLevel)) ? ((-200)) : (mCurNoiseLevel)))
860         mCurNoiseLevel = (XFLOAT)x / (XFLOAT)1.0e7
861
862         x = (long)((XFLOAT)1.0e7*((((-200)) > (mSpeechThreshold)) ? ((-200)) : (mSpeechThreshold)))
863         mSpeechThreshold = (XFLOAT)x / (XFLOAT)1.0e7
```

```
864
865         if (!mFiltered)
866         {
867             mASLBeforeFilt      = mCurrentASL
868             mNoiseLevBeforeFilt = mCurNoiseLevel
869         }
870         if (!mLevAligned)
871         {
872             mUnalignedASL      = mCurrentASL
873             mUnalignedNoiseLev = mCurNoiseLevel
874         }
875     }
876     OPTCATCH( (string errorMsg))
877     {
878         OPTTHROW(( string("ERROR in SQSignal::CalcASLandNoiseLevel(): " + errorMsg + "\n")))
879     }
880     OPTCATCH( (...))
881     {
882         OPTTHROW(( string("ERROR in SQSignal::CalcASLandNoiseLevel(): Unknown error.\n")))
883     }
884 }
885
886 void SQSignal::LevelAlign(XFLOAT fTargetASLdBov)
887 {
888     OPTTRY
889     {
890         if (mIsInvalidSignal)
891         {
892
893             return
894         }
895         if (mCurrentASL > 0.0f || fTargetASLdBov > 0.0f || mData == NULL || mNrOfSamples <= 0)
896         OPTTHROW(( string("Invalid signal data or speech level. Did you forget to call
    CalcASLandNoiseLevel() before?")))
897
898         XFLOAT fLevCorrFac = dBtoRMS(fTargetASLdBov - mCurrentASL)
899         vsmul(mData, fLevCorrFac, mData, mNrOfSamples)
900
901         if (mEnv           != NULL)
902             vsadd(mEnv,           fTargetASLdBov - mCurrentASL,        mEnv,           mNrOfFrames)
903         if (mAvgSpeechSpec != NULL)
904             vsmul(mAvgSpeechSpec, dBtoPow(fTargetASLdBov-mCurrentASL), mAvgSpeechSpec, FFTSIZE/2)
905         if (mAvgNoiseSpec  != NULL)
906             vsmul(mAvgNoiseSpec,  dBtoPow(fTargetASLdBov-mCurrentASL), mAvgNoiseSpec,  FFTSIZE/2)
907
908         mSpeechThreshold += fTargetASLdBov - mCurrentASL
909         mCurNoiseLevel   += fTargetASLdBov - mCurrentASL
910         mCurrentASL       = fTargetASLdBov
911         mLevAligned       = true && !mIsInvalidSignal
912     }
913     OPTCATCH( (string errorMsg))
914     {
915         OPTTHROW(( string("ERROR in SQSignal::LevelAlign(): " + errorMsg + "\n")))
916     }
917 }
918
919 void SQSignal::SetSamplingFrequency(int iFrequency)
920 {
921     mSamplingFreq = iFrequency
922 }
923
924 int SQSignal::Resample(int iTargetFs)
925 {
926     if (mData == NULL || mNrOfSamples <= 1 || mSamplingFreq < 1 || iTargetFs < 1)
927     {
928
929         return SQERR_RESAMPLING
930     }
```

```
931
932     int iRetVal = SQ_NO_ERRORS
933
934     if (iTargetFs == mSamplingFreq)
935         return iRetVal
936
937     OPTTRY
938     {
939         unsigned long newNrOfSamples
940
941         XFLOAT ResamplingFactor = (XFLOAT)iTargetFs/(XFLOAT)mSamplingFreq
942         int newBufferLength = (int)(ceil(ResamplingFactor*1000)*mNrOfSamples/1000)
943         XFLOAT *resampledSignal = (XFLOAT*)matMalloc(newBufferLength * sizeof(XFLOAT))
944
945         iRetVal = matConvertSamplerate(mData, mNrOfSamples, resampledSignal, newBufferLength,
    ResamplingFactor, &newNrOfSamples)
946
947         mNrOfSamples = newNrOfSamples
948         matFree(mData)
949         mData = (XFLOAT*)matMalloc(mNrOfSamples * sizeof(XFLOAT))
950         matbCopy(resampledSignal, mData, newNrOfSamples)
951         matFree(resampledSignal)
952         resampledSignal = NULL
953         mSamplingFreq = iTargetFs
954     }
955     OPTCATCH((string errorMsg))
956     {
957         errorMsg += "ERROR in SQSignal::Resample():\n"
958
959         iRetVal = SQERR_RESAMPLING
960     }
961
962     return iRetVal
963 }
964
965 void SQSignal::SlidingWinMeanRemoval(int WINLEN)
966 {
967     if (mData == NULL || mNrOfSamples < 1 || WINLEN < 3 || mNrOfSamples < WINLEN+3)
968         OPTTHROW(( string("ERROR in SlidingWinMeanRemoval: Invalid input arguments.\n")))
969     if (mIsInvalidSignal)
970     {
971
972         return
973     }
974
975     if ((WINLEN | 0x1) != WINLEN)
976         WINLEN--
977
978     XFLOAT *buff = NULL
979     OPTTRY
980     {
981
982         buff = (XFLOAT*)matMalloc(WINLEN/2 * sizeof(XFLOAT))
983         matbZero(buff, WINLEN/2)
984
985         XFLOAT windowSum = (XFLOAT)0.0
986         windowSum = matSum(mData, WINLEN/2)
987
988         int   rightWinPos = WINLEN/2
989         int   bufferPos   = 0
990         int   curPos      = 0
991         XFLOAT newVal
992         for (  curPos < mNrOfSamples - WINLEN/2  curPos++, rightWinPos++)
993         {
994             newVal = mData[curPos]
995             mData[curPos] -= windowSum/WINLEN
996
997             windowSum += mData[rightWinPos] - buff[bufferPos]
```

```
998             buff[bufferPos] = newVal
999             bufferPos = (bufferPos+1) % (WINLEN/2)
1000        }
1001
1002        for (  curPos < mNrOfSamples  curPos++)
1003        {
1004            mData[curPos] -= windowSum/WINLEN
1005
1006            windowSum -= buff[bufferPos]
1007            bufferPos = (bufferPos+1) % (WINLEN/2)
1008        }
1009
1010        matFree(buff)
1011        buff = NULL
1012    }
1013    OPTCATCH( (...))
1014    {
1015        matFree(buff)
1016        buff = NULL
1017        OPTTHROW(( string("ERROR in SlidingWinMeanRemoval: Unknown error.\n")))
1018    }
1019 }
1020
1021 XFLOAT const* SQSignal::Data() const
1022 {
1023     return mData
1024 }
1025
1026 XFLOAT* SQSignal::Env() const
1027 {
1028     return mEnv
1029 }
1030
1031 XFLOAT SQSignal::EnvLevel(int samplePos) const
1032 {
1033     if (mEnv == NULL || mNrOfFrames <= 0 || samplePos < 0)
1034         OPTTHROW(-1)
1035
1036     int safeFramePos =
1037         round(( samplePos - mFrameSize*(1-mOverlapFac) ) / ( mFrameSize*mOverlapFac ))
1038     safeFramePos = limit(safeFramePos, 0, (int)mNrOfFrames-1)
1039
1040     return mEnv[safeFramePos]
1041 }
1042
1043 int SQSignal::BitResolution() const
1044 {
1045     return mBitResolution
1046 }
1047
1048 short const* SQSignal::VADprofile() const
1049 {
1050     return mVADprofile
1051 }
1052
1053 long SQSignal::SamplePosToFrameNum(int samplePos) const
1054 {
1055     if (mNrOfFrames <= 0 || samplePos < 0)
1056         OPTTHROW(-1)
1057
1058     int safeFramePos =
1059         round(( samplePos - mFrameSize*(1-mOverlapFac) ) / ( mFrameSize*mOverlapFac ))
1060     safeFramePos = limit(safeFramePos, 0, (int)mNrOfFrames-1)
1061
1062     return safeFramePos
1063 }
1064
1065 XFLOAT const * SQSignal::AvgSpeechSpec() const
```

```
{
    return mAvgSpeechSpec
}

XFLOAT const * SQSignal::AvgNoiseSpec() const
{
    return mAvgNoiseSpec
}

XFLOAT const * SQSignal::UsedFiltFreqResp() const
{
    return mUsedFiltFreqResp
}

int SQSignal::SamplingFreq() const
{
    return mSamplingFreq
}

int SQSignal::OrigSampFreq() const
{
    return mOrigSampFreq
}

long SQSignal::NrOfSamples() const
{
    return mNrOfSamples
}

int SQSignal::FrameSize() const
{
    return mFrameSize
}

XFLOAT SQSignal::OverlapFac() const
{
    return mOverlapFac
}

long SQSignal::NrOfFrames() const
{
    return mNrOfFrames
}

long SQSignal::Start() const
{
    return mStart
}

long SQSignal::End() const
{
    return mEnd
}

short SQSignal::NumPause() const
{
    return mNumPause
}

long* SQSignal::PauseCenter() const
{
    return mPauseCenter
}

long SQSignal::PauseCenter(int idx) const
{
    if (idx >= mNumPause)
        OPTTHROW(( string ("Invalid index for pause center.")))
```

```
1134
1135        return mPauseCenter[idx]
1136    }
1137
1138    XFLOAT SQSignal::DCOffset() const
1139    {
1140        return mDCOffset
1141    }
1142
1143    XFLOAT SQSignal::CurrentASL() const
1144    {
1145        return mCurrentASL
1146    }
1147
1148    XFLOAT SQSignal::UnalignedASL() const
1149    {
1150        return mUnalignedASL
1151    }
1152
1153    XFLOAT SQSignal::ASLBeforeFilt() const
1154    {
1155        return mASLBeforeFilt
1156    }
1157
1158    XFLOAT SQSignal::CurrentNoiseLevel() const
1159    {
1160        return mCurNoiseLevel
1161    }
1162
1163    XFLOAT SQSignal::UnalignedNoiseLevel() const
1164    {
1165        return mUnalignedNoiseLev
1166    }
1167
1168    XFLOAT SQSignal::NoiseLevelBeforeFilt() const
1169    {
1170        return mNoiseLevBeforeFilt
1171    }
1172
1173    XFLOAT SQSignal::SpeechActivity () const
1174    {
1175        return mSpeechActivity
1176    }
1177
1178    XFLOAT SQSignal::SpeechThreshold () const
1179    {
1180        return mSpeechThreshold
1181    }
1182
1183    XFLOAT SQSignal::MaxAmplitude () const
1184    {
1185        return mMaxAmplitude
1186    }
1187
1188    bool SQSignal::IsInvalidSignal() const
1189    {
1190        return mIsInvalidSignal
1191    }
1192
1193    bool SQSignal::Preprocessed() const
1194    {
1195        return mPreprocessed
1196    }
1197
1198    bool SQSignal::Filtered() const
1199    {
1200        return mFiltered
1201    }
```

```
1202
1203 bool SQSignal::LevelAligned() const
1204 {
1205     return mLevAligned
1206 }
1207
1208 void SQSignal::SetNumPause(int num)
1209 {
1210     mNumPause = num
1211 }
1212
1213 void SQSignal::SetPauseCenter(long *origPauseCenter, int originalFreq)
1214 {
1215     if(mPauseCenter)
1216         matFree(mPauseCenter)
1217     mPauseCenter = NULL
1218
1219     if (mNumPause > 0)
1220     {
1221         mPauseCenter = (long*)matMalloc(mNumPause * sizeof(long))
1222         lCopyVector (origPauseCenter, (long)mNumPause, mPauseCenter)
1223         lScaleVector (mPauseCenter, (long)mNumPause, (XFLOAT)mSamplingFreq / originalFreq,
    mPauseCenter)
1224     }
1225 }
1226
1227 }
1228
1229 }
1230
1231
```