```
    typedef double XFLOAT
    typedef double OTA_FLOAT

    typedef double OTA_FLOAT
    typedef MAT_DCplx OTA_CPLX


{

typedef struct
{
    float FrameWeightWeight
    bool  UseRelDistance
    float ViterbiDistanceWeightFactor
} VITERBI_PARA

typedef struct
{
    long Samplerate
    int mSRDetectFineAlignCorrlen
    int mDelayFineAlignCorrlen
    int WindowSize[8]
    int CoarseAlignCorrlen[8]
    float pViterbiDistanceWeightFactor[8]
} SPEECH_WINDOW_PARA

typedef struct
{
    SPEECH_WINDOW_PARA Win[3]
    float LowEnergyThresholdFactor
    float LowCorrelThreshold

    float FineAlignLowEnergyThresh
    float FineAlignLowEnergyCorrel
    float FineAlignShortDropOfCorrelR
    float FineAlignShortDropOfCorrelRLastBest
    float ViterbiDistanceWeightFactorDist
    float ViterbiDistanceWeightFactor
} SPEECH_TA_PARA

typedef struct
{
    SPEECH_WINDOW_PARA Win[3]
    float LowEnergyThresholdFactor
    float LowCorrelThreshold

    float FineAlignLowEnergyThresh
    float FineAlignLowEnergyCorrel
    float FineAlignShortDropOfCorrelR
    float FineAlignShortDropOfCorrelRLastBest
    float ViterbiDistanceWeightFactorDist
    float ViterbiDistanceWeightFactor
} AUDIO_TA_PARA

typedef struct
{
    float mCorrForSkippingInitialDelaySearch
    int CoarseAlignSegmentLengthInMs
} GENERAL_TA_PARA

typedef struct
{
    void Init(long Samplerate)
    {
        if (Samplerate==16000)      MaxWin=4
        else if (Samplerate==8000)  MaxWin=4
```

```
        else                        MaxWin=4

        LowPeakEliminationThreshold= 0.2000000029802322

        if (Samplerate==16000)      PercentageRequired = 0.05F
        else if (Samplerate==8000)  PercentageRequired = 0.1F
        else                        PercentageRequired = 0.02F

        MaxDistance = 14

        MinReliability = 7

        PercentageRequired = 0.7
        OTA_FLOAT MaxGradient = 1.1
        OTA_FLOAT MaxTimescaling = 0.1

        if (Samplerate==48000)      MaxStepPerFrame = MaxGradient * 1024.0
        else if (Samplerate==8000)  MaxStepPerFrame = MaxGradient * 128.0
        MaxBins = ((int)(MaxStepPerFrame*2.0*0.9))
        MaxStepPerFrame *= 4

    }

    float LowEnergyThresholdFactor
    float LowCorrelThreshold

    int     MaxStepPerFrame
    int     MaxBins
    int     MaxWin
    int     MinHistogramData

    float   MinReliability

    double  LowPeakEliminationThreshold
    float   MinFrequencyOfOccurrence
    float   LargeStepLimit

    float   MaxDistanceToLast
    float   MaxDistance
    float   MaxLargeStep

    float   ReliabilityThreshold
    float   PercentageRequired

    float   AllowedDistancePara2
    float   AllowedDistancePara3
} SR_ESTIMATION_PARA

class CParameters
{
    public:
        CParameters()
        {
            int i
            mTAPara.mCorrForSkippingInitialDelaySearch = 0.6F
            mTAPara.CoarseAlignSegmentLengthInMs = 600

            SPEECH_WINDOW_PARA      SpeechWinPara[] =
            {
                    {8000,   32, 32,
                        {128,  256, 128,  64,  32,  0, 0},
                        {-1,    -1,  -1,  86,  34,  0,  0},
                        {-1,    -1,  -1,  15,  12,  0,  0}},
                    {16000,  64, 64,
                        {256,  512, 256, 128,  64,  0},
                        {-1,    -1,  -1,  63,  33,  0},
                        {-1,    -1,  -1,  13,  10,  0}},
                    {48000, 256, 256,
```

```
                        {512, 1024, 512, 512, 128,  0},
                        {-1,    -1,  -1, 115,  61,  0},
                        {-1,    -1,  -1,  17,  16,  0}}
            }

            for (i=0  i<3  i++)
            {
                mSpeechTAPara.Win[i].Samplerate = SpeechWinPara[i].Samplerate
                mSpeechTAPara.Win[i].mDelayFineAlignCorrlen =
SpeechWinPara[i].mDelayFineAlignCorrlen
                mSpeechTAPara.Win[i].mSRDetectFineAlignCorrlen =
SpeechWinPara[i].mSRDetectFineAlignCorrlen
                for (int k=0  k<8  k++)
                {
                    mSpeechTAPara.Win[i].CoarseAlignCorrlen[k] =
SpeechWinPara[i].CoarseAlignCorrlen[k]
                    mSpeechTAPara.Win[i].WindowSize[k]         = SpeechWinPara[i].WindowSize[k]

                    mSpeechTAPara.Win[i].pViterbiDistanceWeightFactor[k] =
SpeechWinPara[i].pViterbiDistanceWeightFactor[k]
                }
            }
            mSpeechTAPara.LowEnergyThresholdFactor = 15.0F
            mSpeechTAPara.LowCorrelThreshold = 0.4F
            mSpeechTAPara.FineAlignLowEnergyThresh = 2.0
            mSpeechTAPara.FineAlignLowEnergyCorrel = 0.6F
            mSpeechTAPara.FineAlignShortDropOfCorrelR = -1
            mSpeechTAPara.FineAlignShortDropOfCorrelRLastBest = 0.65F

            mSpeechTAPara.ViterbiDistanceWeightFactorDist = 5

            SPEECH_WINDOW_PARA      AudioWinPara[] =
            {
                    {8000,   32, 32,
                        {64,   128,  64,  64,  16,  0, 0},
                        {-1,    -1,  -1, 128,  32,  0, 0},
                        {-1,    -1,  -1,   6,   6,  0, 0}},
                    {16000,  64, 64,
                        {128,  256, 128, 128,  32,  0},
                        {-1,    -1,  -1,  64,  32,  0},
                        {-1,    -1,  -1,  12,  12,  0}},
                    {48000, 256, 2048,
                        {512, 1024, 512, 512, 256,  128,   0},
                        {-1,    -1,  -1, 512, 1024, 2048,  0},
                        {-1,    -1,  -1,  16,  16,  32,    0}}
            }

            for (i=0  i<3  i++)
            {
                mAudioTAPara.Win[i].Samplerate = AudioWinPara[i].Samplerate
                mAudioTAPara.Win[i].mDelayFineAlignCorrlen    =
AudioWinPara[i].mDelayFineAlignCorrlen
                mAudioTAPara.Win[i].mSRDetectFineAlignCorrlen =
AudioWinPara[i].mSRDetectFineAlignCorrlen
                for (int k=0  k<8  k++)
                {
                    mAudioTAPara.Win[i].CoarseAlignCorrlen[k] =
AudioWinPara[i].CoarseAlignCorrlen[k]
                    mAudioTAPara.Win[i].WindowSize[k]         = AudioWinPara[i].WindowSize[k]
                    mAudioTAPara.Win[i].pViterbiDistanceWeightFactor[k] =
AudioWinPara[i].pViterbiDistanceWeightFactor[k]
                }
            }
            mAudioTAPara.LowEnergyThresholdFactor = 1
            mAudioTAPara.LowCorrelThreshold = 0.85F
            mAudioTAPara.FineAlignLowEnergyThresh = 32.0
            mAudioTAPara.FineAlignLowEnergyCorrel = 0.8F
            mAudioTAPara.FineAlignShortDropOfCorrelR = -1
```

```
            mAudioTAPara.FineAlignShortDropOfCorrelRLastBest = 0.8F
            mAudioTAPara.ViterbiDistanceWeightFactorDist = 6

            mSREPara.LowEnergyThresholdFactor = 15.0F
            mSREPara.LowCorrelThreshold = 0.4F

            mSREPara.MaxStepPerFrame = 160
            mSREPara.MaxBins = ((int)(mSREPara.MaxStepPerFrame*2.0*0.9))

            mSREPara.MaxWin=4
            mSREPara.LowPeakEliminationThreshold=0.2000000029802322F
            mSREPara.PercentageRequired = 0.04F

            mSREPara.LargeStepLimit = 0.08F
            mSREPara.MaxDistanceToLast = 7
            mSREPara.MaxLargeStep = 5
            mSREPara.MaxDistance = 14

            mSREPara.MinReliability = 7
            mSREPara.MinFrequencyOfOccurrence = 3

            mSREPara.AllowedDistancePara2 = 0.85F
            mSREPara.AllowedDistancePara3 = 1.5F

            mSREPara.ReliabilityThreshold = 0.3F
            mSREPara.MinHistogramData = 8

            mViterbi.UseRelDistance = false
            mViterbi.FrameWeightWeight = 1.0F
        }

        void Init(long Samplerate)
        {
            mSREPara.Init(Samplerate)
        }

        VITERBI_PARA        mViterbi
        GENERAL_TA_PARA     mTAPara
        SPEECH_TA_PARA      mSpeechTAPara
        AUDIO_TA_PARA       mAudioTAPara
        SR_ESTIMATION_PARA  mSREPara
}
}


{

class CProcessData
{
    public:
        CProcessData()
        {
            int i

            mCurrentIteration = -1
            mStartPlotIteration=10
            mLastPlotIteration =10
            mEnablePlotting=false
            mpLogFile = 0

            mWindowSize = 2048
            mSRDetectFineAlignCorrlen = 1024
            mDelayFineAlignCorrlen = 1024
            mOverlap    = 1024
            mSamplerate = 48000
            mNumSignals = 0
            mpMathlibHandle = 0
            mMinLowVarDelay = -99999999
```

```
            mMaxHighVarDelay = 99999999

            mMinStaticDelayInMs = -2500
            mMaxStaticDelayInMs = 2500

            mMaxToleratedRelativeSamplerateDifference = 1.0

            for (i=0  i<8  i++)
                mpViterbiDistanceWeightFactor[i] = 0.0001F
        }

        int mMinStaticDelayInMs
        int mMaxStaticDelayInMs

        int mMinLowVarDelayInSamples
        int mMaxHighVarDelayInSamples

        int mStartPlotIteration
        int mLastPlotIteration
        bool mEnablePlotting
        long mSamplerate

        FILE* mpLogFile

        int mCurrentIteration

        int mpWindowSize[8]

        int mpOverlap[8]

        int mpCoarseAlignCorrlen[8]

        float mpViterbiDistanceWeightFactor[8]

        int mDelayFineAlignCorrlen
        int mSRDetectFineAlignCorrlen
        float mMaxToleratedRelativeSamplerateDifference
        int mWindowSize

        int mOverlap

        int mCoarseAlignCorrlen

        int mNumSignals
        void* mpMathlibHandle

        int mMinLowVarDelay
        int mMaxHighVarDelay
        int mStepSize

        bool Init(int Iteration, float MoreDownsampling)
        {
            assert(MoreDownsampling)

            mCurrentIteration = Iteration
            mP.Init(mSamplerate)

            mWindowSize = (int)((float)mpWindowSize[Iteration]*MoreDownsampling)
            mOverlap = (int)((float)mpOverlap[Iteration]*MoreDownsampling)
            mCoarseAlignCorrlen = mpCoarseAlignCorrlen[Iteration]
            mStepSize = mWindowSize - mOverlap
            mMinLowVarDelay = mMinLowVarDelayInSamples / mStepSize
            mMaxHighVarDelay = mMaxHighVarDelayInSamples / mStepSize

            float D = mpViterbiDistanceWeightFactor[Iteration]
            D = D * mSamplerate / mStepSize / 1000
            float F = ((float)log(1+0.5)) / (D*D)
            mP.mViterbi.ViterbiDistanceWeightFactor = F
```

```
332
333             D = mP.mSpeechTAPara.ViterbiDistanceWeightFactorDist
334             D = D * mSamplerate / 1000
335             F =((float) log(1+0.5) / (D*D))
336             mP.mSpeechTAPara.ViterbiDistanceWeightFactor = F
337
338             return true
339         }
340
341         CParameters   mP
342 }
343
344 class SECTION
345 {
346     public:
347         int Start
348         int End
349         int Len() {return End-Start }
350         void CopyFrom(const SECTION &src)
351         {
352             this->Start = src.Start
353             this->End   = src.End
354         }
355 }
356
357 typedef struct OTA_RESULT
358 {
359     void CopyFrom(const OTA_RESULT* src)
360     {
361         mNumFrames           = src->mNumFrames
362         mStepsize            = src->mStepsize
363         mResolutionInSamples = src->mResolutionInSamples
364         if (src->mpDelay != NULL && mNumFrames > 0)
365         {
366             matFree(mpDelay)
367             mpDelay = (long*)matMalloc(mNumFrames * sizeof(long))
368             for (int i = 0  i < mNumFrames  i++)
369                 mpDelay[i] = src->mpDelay[i]
370         }
371         else
372         {
373             matFree(mpDelay)
374             mpDelay = NULL
375         }
376
377         if (src->mpReliability != NULL && mNumFrames > 0)
378         {
379             matFree(mpReliability)
380             mpReliability = (OTA_FLOAT*)matMalloc(mNumFrames * sizeof(OTA_FLOAT))
381             for (int i = 0  i < mNumFrames  i++)
382                 mpReliability[i] = src->mpReliability[i]
383         }
384         else
385         {
386             matFree(mpReliability)
387             mpReliability = NULL
388         }
389         mAvgReliability   = src->mAvgReliability
390         mRelSamplerateDev = src->mRelSamplerateDev
391
392         mNumUtterances = src->mNumUtterances
393         if (src->mpStartSampleUtterance != NULL && mNumUtterances > 0)
394         {
395             matFree(mpStartSampleUtterance)
396             mpStartSampleUtterance = (int*)matMalloc(mNumUtterances * sizeof(int))
397             for (int i = 0  i < mNumUtterances  i++)
398                 mpStartSampleUtterance[i] = src->mpStartSampleUtterance[i]
399         }
```

```
        else
        {
            matFree(mpStartSampleUtterance)
            mpStartSampleUtterance = NULL
        }
        if (src->mpStopSampleUtterance != NULL && mNumUtterances > 0)
        {
            matFree(mpStopSampleUtterance)
            mpStopSampleUtterance = (int*)matMalloc(mNumUtterances * sizeof(int))
            for (int i = 0  i < mNumUtterances  i++)
                mpStopSampleUtterance[i] = src->mpStopSampleUtterance[i]
        }
        else
        {
            matFree(mpStopSampleUtterance)
            mpStopSampleUtterance = NULL
        }
        if (src->mpDelayUtterance != NULL && mNumUtterances > 0)
        {
            matFree(mpDelayUtterance)
            mpDelayUtterance = (int*)matMalloc(mNumUtterances * sizeof(int))
            for (int i = 0  i < mNumUtterances  i++)
                mpDelayUtterance[i] = src->mpDelayUtterance[i]
        }
        else
        {
            matFree(mpDelayUtterance)
            mpDelayUtterance = NULL
        }

        mNumSections = src->mNumSections
        if (src->mpRefSections != NULL && mNumSections > 0)
        {
            delete[] mpRefSections
            mpRefSections = new SECTION[mNumSections]
            for (int i = 0  i < mNumSections  i++)
                mpRefSections[i].CopyFrom(src->mpRefSections[i])
        }
        else
        {
            delete[] mpRefSections
            mpRefSections = NULL
        }
        if (src->mpDegSections != NULL && mNumSections > 0)
        {
            delete[] mpDegSections
            mpDegSections = new SECTION[mNumSections]
            for (int i = 0  i < mNumSections  i++)
                mpDegSections[i].CopyFrom(src->mpDegSections[i])
        }
        else
        {
            delete[] mpDegSections
            mpDegSections = NULL
        }

        mSNRRefdB = src->mSNRRefdB
        mSNRDegdB = src->mSNRDegdB
        mNoiseLevelRef = src->mNoiseLevelRef
        mNoiseLevelDeg = src->mNoiseLevelDeg
        mSignalLevelRef = src->mSignalLevelRef
        mSignalLevelDeg = src->mSignalLevelDeg
        mNoiseThresholdRef = src->mNoiseThresholdRef
        mNoiseThresholdDeg = src->mNoiseThresholdDeg

        if (src->mpActiveFrameFlags != NULL && mNumFrames > 0)
        {
            matFree(mpActiveFrameFlags)
```

```
            mpActiveFrameFlags = (int*)matMalloc(mNumFrames * sizeof(int))
            for (int i = 0  i < mNumFrames  i++)
                mpActiveFrameFlags[i] = src->mpActiveFrameFlags[i]
        }
        else
        {
            matFree(mpActiveFrameFlags)
            mpActiveFrameFlags = NULL
        }

        if (src->mpIgnoreFlags != NULL && mNumFrames > 0)
        {

            matFree(mpIgnoreFlags)
            mpIgnoreFlags = (int*)matMalloc(mNumFrames * sizeof(int))
            mNumIngoreFlags = src->mNumIngoreFlags
            for (int i = 0  i < mNumFrames  i++)
                mpIgnoreFlags[i] = src->mpIgnoreFlags[i]
        }
        else
        {
            matFree(mpIgnoreFlags)
            mpIgnoreFlags = NULL
        }

        for (int i = 0  i < 5  i++)
            mTimeDiffs[i] = src->mTimeDiffs[i]

        mAslFrames = src->mAslFrames
        mAslFramelength = src->mAslFramelength
        if (src->mpAslActiveFrameFlags != NULL && mAslFrames > 0)
        {
            matFree(mpAslActiveFrameFlags)
            mpAslActiveFrameFlags = (int*)matMalloc(mAslFrames * sizeof(int))
            for (int i = 0  i < mAslFrames  i++)
                mpAslActiveFrameFlags[i] = src->mpAslActiveFrameFlags[i]
        }
        else
        {
            matFree(mpAslActiveFrameFlags)
            mpAslActiveFrameFlags = NULL
        }

        mAslFramesDeg = src->mAslFramesDeg
        if (src->mpAslActiveFrameFlagsDeg != NULL && mAslFramesDeg > 0)
        {
            matFree(mpAslActiveFrameFlagsDeg)
            mpAslActiveFrameFlagsDeg = (int*)matMalloc(mAslFramesDeg * sizeof(int))
            for (int i = 0  i < mAslFramesDeg  i++)
                mpAslActiveFrameFlagsDeg[i] = src->mpAslActiveFrameFlagsDeg[i]
        }
        else
        {
            matFree(mpAslActiveFrameFlagsDeg)
            mpAslActiveFrameFlagsDeg = NULL
        }

        FirstRefSample = src->FirstRefSample
        FirstDegSample = src->FirstDegSample
    }

    OTA_RESULT()
    {
        mNumFrames = 0
        mpDelay = NULL

        mpReliability = NULL

```

```
        mNumUtterances = 0
        mpStartSampleUtterance = NULL
        mpStopSampleUtterance  = NULL
        mpDelayUtterance       = NULL

        mNumSections = 0
        mpRefSections = NULL
        mpDegSections = NULL

        mpActiveFrameFlags = NULL
        mpIgnoreFlags = NULL
        mNumIngoreFlags = 0

        mAslFramelength = 0
        mAslFrames = 0
        mpAslActiveFrameFlags = NULL
        mAslFramesDeg = 0
        mpAslActiveFrameFlagsDeg = NULL

        FirstRefSample = FirstDegSample = 0
    }

    ~OTA_RESULT()
    {
        matFree(mpDelay)
        mpDelay = NULL

        matFree(mpReliability)
        mpReliability = NULL

        matFree(mpStartSampleUtterance)
        mpStartSampleUtterance = NULL

        matFree(mpStopSampleUtterance)
        mpStopSampleUtterance  = NULL

        matFree(mpDelayUtterance)
        mpDelayUtterance       = NULL

        delete[] mpRefSections
        mpRefSections = NULL
        delete[] mpDegSections
        mpDegSections = NULL

        matFree(mpActiveFrameFlags)
        mpActiveFrameFlags = NULL

        matFree(mpIgnoreFlags)
        mpIgnoreFlags = NULL

        matFree(mpAslActiveFrameFlags)
        mpAslActiveFrameFlags = NULL
        matFree(mpAslActiveFrameFlagsDeg)
        mpAslActiveFrameFlagsDeg = NULL
    }

    long mNumFrames
    int mStepsize
    int mResolutionInSamples
    int mPitchFrameSize
    long *mpDelay
    OTA_FLOAT *mpReliability
    OTA_FLOAT mAvgReliability
    OTA_FLOAT mRelSamplerateDev

    int  mNumUtterances
    int* mpStartSampleUtterance
    int* mpStopSampleUtterance
```

```
    int* mpDelayUtterance
    int FirstRefSample
    int FirstDegSample

    int         mNumSections
    SECTION     *mpRefSections
    SECTION     *mpDegSections

    double mSNRRefdB, mSNRDegdB
    double mNoiseLevelRef, mNoiseLevelDeg
    double mSignalLevelRef, mSignalLevelDeg
    double mNoiseThresholdRef, mNoiseThresholdDeg

    int *mpActiveFrameFlags

    int *mpIgnoreFlags
    int mNumIngoreFlags
    int mAslFrames
    int mAslFramelength
    int *mpAslActiveFrameFlags
    int mAslFramesDeg
    int *mpAslActiveFrameFlagsDeg

    double mTimeDiffs[5]

}OTA_RESULT

struct FilteringParameters
{
    int pListeningCondition
    double cutOffFrequencyLow
    double cutOffFrequencyHigh
    double disturbedEnergyQuotient
}

class ITempAlignment
{
    public:

        virtual bool Init(CProcessData* pProcessData)=0
        virtual void Free()=0
        virtual void Destroy()=0

        virtual bool SetSignal(int Index, unsigned long SampleRate, unsigned long NumSamples, int
NumChannels, OTA_FLOAT** pSignal)=0

        virtual void GetFilterCharacteristics(FilteringParameters *FilterParams)=0

        virtual bool FilterSignal(int Index, FilteringParameters *FilterParams)=0

        virtual bool Run(unsigned long Control, OTA_RESULT* pResult, int TArunIndex)=0

        virtual void GetNoiseSwitching(OTA_FLOAT* pBGNSwitchingLevel, OTA_FLOAT*
pNoiseLevelSpeechDeg, OTA_FLOAT* pNoiseLevelSilenceDeg)=0

        virtual OTA_FLOAT GetPitchFreq(int Signal, int Channel)=0

        virtual OTA_FLOAT GetPitchVector(int Signal, int Channel, OTA_FLOAT* pVector, int NumFrames,
int SamplesPerFrame)=0
        virtual int GetPitchFrameSize()=0
}

enum AlignmentType
{
    TA_FOR_SPEECH=0,

}
```

```
669 ITempAlignment* CreateAlignment(AlignmentType Type)
670
671 }
672
673
674 {
675
676 CSpeechActiveFrameDetection::CSpeechActiveFrameDetection()
677 {
678     for (int s=0  s<2  s++)
679     {
680         for (int c=0  c<2  c++)
681         {
682             mppActiveFrameFlags[s][c]=0
683             mDataValidFlags[s][c]=0
684             mSignalLevels[s][c]=0
685             mNoiseLevels[s][c]=0
686             mNoiseThresholds[s][c]=0
687         }
688     }
689 }
690
691 CSpeechActiveFrameDetection::~CSpeechActiveFrameDetection()
692 {
693     Free()
694 }
695
696 void CSpeechActiveFrameDetection::Free()
697 {
698     for (int s=0  s<2  s++)
699     {
700         for (int c=0  c<2  c++)
701         {
702             if(mppActiveFrameFlags[s][c]) delete[] mppActiveFrameFlags[s][c]
703             mppActiveFrameFlags[s][c]=0
704             mDataValidFlags[s][c]=0
705             mSignalLevels[s][c]=0
706             mNoiseLevels[s][c]=0
707             mNoiseThresholds[s][c]=0
708         }
709     }
710 }
711
712 bool CSpeechActiveFrameDetection::Init(CProcessData* pProcessData)
713 {
714     bool rc=true
715
716     mProcessData = *pProcessData
717     return rc
718 }
719
720 bool CSpeechActiveFrameDetection::Start(CTASignal** pSignals)
721 {
722     bool rc=true
723
724     mProcessData.Init(0, 1)
725
726     rc = mFeatureList.Create(pSignals, &mProcessData, OTA_FLTYPE_VAD)
727
728     for (int i=0  rc && i<2  i++)
729         mNumFeatureFrames[i] = mFeatureList.GetFVector(0, i, 0)->mSize
730
731     return rc
732 }
733
734 void CSpeechActiveFrameDetection::GetNoiseThreshold(OTA_FLOAT* Vec, int VecLen, OTA_FLOAT* pNoise,
    OTA_FLOAT* pSignal, OTA_FLOAT* pNoiseThreshold)
735 {
```

```
    OTA_FLOAT NoiseThreshold
    OTA_FLOAT NoiseLevel
    OTA_FLOAT StdDevOfNoisyPart
    OTA_FLOAT SignalLevel
    OTA_FLOAT MinLevel

    NoiseThreshold = matMean(Vec, VecLen)
    MinLevel = matMax(Vec, VecLen)
    MinLevel = MinLevel >0 ? MinLevel * 1.0e-5 : 0.5
    matbThresh1(Vec, VecLen, MinLevel, MAT_GT)

    for( int i = 0  i < 12  i++ )
    {
        //ToDo: Compute mean and StdDev of the noise power (StdDevOfNoisyPart)*/

        //... Code missing in public C code

        NoiseThreshold = NoiseLevel + 2.005 * StdDevOfNoisyPart
        NoiseThreshold *= 1.001
    }

    //* Compute the signal and noise levels */
    //* ToDo:
    //- Set NoiseLevel  to the mean of all samples <= NoiseThreshold
    //- Set SignalLevel to the mean of all samples > NoiseThreshold
    //- Limit the thresholds to 1e-7
    //

}

OTA_FLOAT CSpeechActiveFrameDetection::ModifyThreshold(OTA_FLOAT NoiseLevel, OTA_FLOAT SignalLevel,
OTA_FLOAT NoiseThreshold, bool IsRefSignal, int* MinSpeechLength)
{
    const OTA_FLOAT LowSNRdB=3

    OTA_FLOAT SNRdB = 10*log10(SignalLevel/NoiseLevel)
    OTA_FLOAT NoisedB = 10*log10(NoiseLevel)

    if (SNRdB<12.0)
    {
        if (SNRdB<LowSNRdB)
        {
            *MinSpeechLength =4
            NoiseThreshold = NoiseLevel + 0.03*(SignalLevel-NoiseLevel)
        }
    }

    else if (SNRdB>16.0 && SNRdB<35)
    {
        NoiseThreshold *= 5
    }
    else if (SNRdB>=35)
    {
        NoiseThreshold *= 1.5
    }

    return NoiseThreshold
}

void CSpeechActiveFrameDetection::IdentifyActivity(OTA_FLOAT* Vec, OTA_FLOAT* pPitchVec, int VecLen,
OTA_FLOAT NoiseLevel, OTA_FLOAT SignalLevel, OTA_FLOAT NoiseThreshold, bool IsRefSignal)
{
    const OTA_FLOAT LowSNRdB=3
    OTA_FLOAT LevelMin=1
    OTA_FLOAT g
    int  count
    int  iteration
    int  start
```

```
    int  finish

    OTA_FLOAT SNRdB = 10*log10(SignalLevel/NoiseLevel)
    OTA_FLOAT NoisedB = 10*log10(NoiseLevel)

    int MaxDropoutLength = MSecondsToFrames(10)
    int MinSpeechLength = MSecondsToFrames(50)
    if (IsRefSignal) MinSpeechLength = MSecondsToFrames(35)
    int MinPauseLength  = MSecondsToFrames(200)
    int MinPauseLength2 = MSecondsToFrames(300)
    int MinPauseLength3 = MSecondsToFrames(500)
    int IsIsolatedDistance = MSecondsToFrames(20)

    NoiseThreshold = ModifyThreshold(NoiseLevel, SignalLevel, NoiseThreshold, IsRefSignal,
&MinSpeechLength)

    for( count = 0L  count < VecLen  count++ )
        if( Vec[count] <= NoiseThreshold )
            Vec[count] = -Vec[count]

    Vec[0] = -LevelMin
    Vec[VecLen-1] = -LevelMin

    int InactivityStart = 0
    int InactivityFinish = 0
    int ActivityStart = 0
    int NextActivityStart = 0
    int ActivityFinish = 0
    for( count = 1  count < VecLen  count++ )
    {
        if( (Vec[count-1] > 0.0f) && (Vec[count] <= 0.0f) )
            ActivityFinish = InactivityStart = count
        if( (Vec[count-1] <= 0.0f) && (Vec[count] > 0.0f) )
        {
            ActivityStart = NextActivityStart
            NextActivityStart = InactivityFinish = count
            if( (InactivityFinish - InactivityStart) <= MaxDropoutLength )
            {
                bool DoIt=false
                if (ActivityFinish-ActivityStart<MinSpeechLength)
                {
                    for (iteration=InactivityFinish  iteration<VecLen &&
iteration<InactivityFinish+MinSpeechLength  iteration++)
                        if (!Vec[iteration])
                            break
                    if (iteration>=InactivityFinish+3*MinSpeechLength)
                        DoIt=true
                }
                else DoIt=true

                if (DoIt)
                    for( iteration = InactivityStart  iteration < InactivityFinish  iteration++ )
                        Vec[iteration] = LevelMin
            }
        }
    }

    if (SNRdB<LowSNRdB)
        //Code missing: extend all active segmants by one frame at either end

    //Missing: Eliminate pauses shorter than MinPauseLength

    if( SignalLevel >= (NoiseLevel * 1000.0f) )
    {
        for( count = 1  count < VecLen  count++ )
        {
            if( (Vec[count] > 0.0f) && (Vec[count-1] <= 0.0f) )
                start = count
```

```
            if( (Vec[count] <= 0.0f) && (Vec[count-1] > 0.0f) )
            {
                finish = count
                g = 0.0f
                for( iteration = start  iteration < finish  iteration++ )
                    g += Vec[iteration]
                if( g < 3.0f * NoiseThreshold * (finish - start) )
                    for( iteration = start  iteration < finish  iteration++ )
                        Vec[iteration] = -Vec[iteration]
            }
        }
    }

    //not available: - Join sections of speech that are separated by less than MinPauseLength */
    //- Join sections of speech that are separated by less than MinPauseLength2 and which exceed the threshold at least once */

    //Todo: Make sure that there is at least one active section

    for( count = 0L  count < VecLen  count++ )
        if( Vec[count] <= 0.0f ) Vec[count] = 0.0f
        else                     Vec[count] = LevelMin

    int PauseStart=Vec[0]>0 ? -1:0
    int SpeechStart= Vec[0]>0 ? 0:-1
    int PreviousPauseStart= -1
    int PreviousSpeechStart= -1
    for( count = 1L  count < VecLen  count++ )
    {
        if (Vec[count-1]>0 && Vec[count]<=0)
        {   PreviousPauseStart = PauseStart  PauseStart = count  }

        if (Vec[count-1]<=0 && Vec[count]>0 )
        {
            PreviousSpeechStart = SpeechStart
            SpeechStart = count

            if (   PauseStart-PreviousSpeechStart < MinSpeechLength*2
                && PreviousSpeechStart - PreviousPauseStart > MSecondsToFrames(400)
                && SpeechStart - PauseStart > MSecondsToFrames(400))
            {
                if (PreviousPauseStart>=0 && PreviousSpeechStart>=0 && PreviousPauseStart>=0 && PauseStart>=0)
                    for (int i=PreviousSpeechStart  i<PauseStart  i++)
                        Vec[i] = 0.0
            }
            else if (   PauseStart-PreviousSpeechStart < MinSpeechLength*2
                && PreviousPauseStart == 0
                && SpeechStart - PauseStart > MSecondsToFrames(400))
            {
                if (PreviousPauseStart>=0 && PreviousSpeechStart>=0 && PreviousPauseStart>=0 && PauseStart>=0)
                    for (int i=PreviousSpeechStart  i<PauseStart  i++)
                        Vec[i] = 0.0
            }

        }
    }

    start = 0L
    finish = 0L
    for( count = 1  count < VecLen  count++ )
    {
        if( (Vec[count] > 0.0f) && (Vec[count-1] <= 0.0f) )
        {
            start = count
            if( (finish > 0L) && ((start - finish) <= MinPauseLength3) )
                for( iteration = finish  iteration < start  iteration++ )
```

```
                    Vec[iteration] = LevelMin
        }
        if( (Vec[count] <= 0.0f) && (Vec[count-1] > 0.0f) )
            finish = count
    }

}

int CSpeechActiveFrameDetection::SkipConstLevel(OTA_FLOAT* Vec, int VecLen)
{
    int StartConstSection=0
    int EndConstSection
    {
        while(StartConstSection<VecLen && Vec[StartConstSection]<0) StartConstSection++

        int i
        OTA_FLOAT AvgE=0
        int NumFramesInWin = MSecondsToFrames(50)
        for (i=StartConstSection  i<VecLen && i<StartConstSection+NumFramesInWin  i++)
            AvgE += Vec[i]
        AvgE /= NumFramesInWin

        EndConstSection = StartConstSection+1
        while (EndConstSection<VecLen && fabs(Vec[EndConstSection])<20*AvgE &&
fabs(Vec[EndConstSection])>0.05*AvgE)
            EndConstSection++

    }

    if (EndConstSection>VecLen-3)
        EndConstSection = 0

    if (EndConstSection-StartConstSection>MSecondsToFrames(50))
        return EndConstSection
    else return 0
}

int CSpeechActiveFrameDetection::SearchStartFrame(int Signal, int Channel, int EarliestStartFrame)
{
    int i, j
    int* pVec = mppActiveFrameFlags[Signal][Channel]
    OTA_FLOAT* pEnergy = mFeatureList.GetFVector(0, Signal, Channel)->mpVector
    int VecLen = (int)mFeatureList.GetFVector(0, Signal, Channel)->mSize

    OTA_FLOAT SigLevel, NoiseLevel, Threshold
    i=EarliestStartFrame  while (i>0 && pVec[i--])  while (i>0 && !pVec[i--])
    j=EarliestStartFrame  while (j<VecLen && !pVec[j++])  while (j<VecLen && pVec[j++])
    GetNoiseThreshold(pEnergy+i, j-i, &NoiseLevel, &SigLevel, &Threshold)
    OTA_FLOAT LocalSNRdB = 10 * log10(SigLevel/NoiseLevel)

    if (0 && LocalSNRdB>7)
    {
        int FramesToSkip = SkipConstLevel(pEnergy+EarliestStartFrame, VecLen-EarliestStartFrame)
        EarliestStartFrame += FramesToSkip
    }

    int NumFramesRequired = (int)(0.050F * (float)mProcessData.mSamplerate /
(float)mProcessData.mStepSize)
    int PotentialStartFrame = EarliestStartFrame
    OTA_FLOAT AvgE = SigLevel

    const OTA_FLOAT MaxSNRdB = 25
    if (LocalSNRdB>MaxSNRdB)
    {
        AvgE = Threshold
    }
    else
    {
```

```
        LocalSNRdB = (((MaxSNRdB) < (LocalSNRdB)) ? (MaxSNRdB) : (LocalSNRdB))
        AvgE = SigLevel-LocalSNRdB*(SigLevel-Threshold)/MaxSNRdB
    }

    for (i=EarliestStartFrame  i<mNumFeatureFrames[Signal]-NumFramesRrequired  i++)
    {
        if (pVec[i])
        {
            bool AvgEExceeded=false
            for (j=1  j<NumFramesRequired  j++)
            {
                if (pEnergy[i+j]>AvgE)
                    AvgEExceeded = true
                if (!pVec[i+j])
                    break
            }
            if (j==NumFramesRequired && AvgEExceeded)
            {
                PotentialStartFrame=i
                break
            }
            else
            {
                i+=j
            }
        }
    }

    if (PotentialStartFrame>EarliestStartFrame+MSecondsToFrames(64))
        PotentialStartFrame-=MSecondsToFrames(64)

    if (PotentialStartFrame>0.98*mNumFeatureFrames[Signal])
        PotentialStartFrame = EarliestStartFrame

    return PotentialStartFrame
}

OTA_FLOAT CSpeechActiveFrameDetection::GetSectionSnrIndB(int Signal, int Channel, int
EarliestStartFrame, OTA_FLOAT* pNoiseLevel, OTA_FLOAT* pSignalLevel, OTA_FLOAT* pThreshold)
{
    int i, j
    int* pVec = mppActiveFrameFlags[Signal][Channel]
    OTA_FLOAT* pEnergy = mFeatureList.GetFVector(0, Signal, Channel)->mpVector
    int VecLen = (int)mFeatureList.GetFVector(0, Signal, Channel)->mSize

    OTA_FLOAT SigLevel, NoiseLevel, Threshold
    i=EarliestStartFrame  while (i>0 && pVec[i--])  while (i>0 && !pVec[i--])  i+=2
    j=EarliestStartFrame  while (j<VecLen && !pVec[j++])  while (j<VecLen && pVec[j++])  j-=2
    GetNoiseThreshold(pEnergy+i, j-i, &NoiseLevel, &SigLevel, &Threshold)
    *pNoiseLevel = 10 * log10(NoiseLevel)
    *pSignalLevel = 10 * log10(SigLevel)
    *pThreshold = 10 * log10(Threshold)
    OTA_FLOAT LocalSNRdB = 10 * log10(SigLevel/NoiseLevel)

    return LocalSNRdB
}

void CSpeechActiveFrameDetection::GetClassificationMeasure(OTA_FLOAT* Vec, int VecLen, int*
NumActiveSections, int* AvgActiveSectionLen, int* TotalActiveSectionLen, OTA_FLOAT*
ActiveInactiveRatio)
{
    int i
    *NumActiveSections=0
    *AvgActiveSectionLen=0
    *TotalActiveSectionLen=0
    int AStart=0
    int AEnd=0
    for( i = 1L  i < VecLen  i++ )
```

```
    {
        if (Vec[i]>0 && Vec[i-1]<=0)
            AStart = i

        if (Vec[i]<=0 && Vec[i-1]>0 || (i==VecLen-1 && Vec[i]>0))
        {
            AEnd = i
            *TotalActiveSectionLen += AEnd-AStart
            (*NumActiveSections)++
        }
    }
    if (*NumActiveSections)
    {
        *AvgActiveSectionLen = *TotalActiveSectionLen / *NumActiveSections
        *ActiveInactiveRatio = (OTA_FLOAT)(*TotalActiveSectionLen) /
(OTA_FLOAT)(VecLen-*TotalActiveSectionLen)
    }
    else
    {
        *AvgActiveSectionLen = 0
        *NumActiveSections = 0,
        *ActiveInactiveRatio = 0.0
        *TotalActiveSectionLen = 1.0E-15
    }
}

void CSpeechActiveFrameDetection::ClassifyDistortion(int Channel, OTA_FLOAT* ClickLevelOfDeg)
{
    int i, VecLen
    int Dummy
    OTA_FLOAT pNoiseLevel[2]
    OTA_FLOAT pSignalLevel[2]
    OTA_FLOAT pNoiseThreshold[2]
    int NumActiveSections[2]
    int AvgActiveSectionLen[2]
    int TotalActiveSectionLen[2]
    OTA_FLOAT ActiveInactiveRatio[2]

    VecLen = ((((int)mFeatureList.GetFVector(0, 0, Channel)->mSize) <
((int)mFeatureList.GetFVector(0, 1, Channel)->mSize)) ? ((int)mFeatureList.GetFVector(0, 0,
Channel)->mSize) : ((int)mFeatureList.GetFVector(0, 1, Channel)->mSize))

    if (1 || VecLen)
    {
        OTA_FLOAT* Vec = (OTA_FLOAT*)matMalloc(VecLen * sizeof(OTA_FLOAT))
        matbCopy(mFeatureList.GetFVector(0, 0, Channel)->mpVector, Vec, VecLen)
        GetNoiseThreshold(Vec, VecLen,  pNoiseLevel+0,  pSignalLevel+0,  pNoiseThreshold+0)
        pNoiseThreshold[0] = ModifyThreshold(pNoiseLevel[0], pSignalLevel[0], pNoiseThreshold[0],
true, &Dummy)

        for( i = 0L  i < VecLen  i++ )
            if( Vec[i] <= pNoiseThreshold[0] )
                Vec[i] = -Vec[i]
        Vec[0] = Vec[VecLen-1] = -1.0
        GetClassificationMeasure(Vec, VecLen, NumActiveSections+0, AvgActiveSectionLen+0,
TotalActiveSectionLen+0, ActiveInactiveRatio+0)

        matbCopy(mFeatureList.GetFVector(0, 1, Channel)->mpVector, Vec, VecLen)
        GetNoiseThreshold(Vec, VecLen,  pNoiseLevel+1,  pSignalLevel+1,  pNoiseThreshold+1)
        pNoiseThreshold[1] = ModifyThreshold(pNoiseLevel[1], pSignalLevel[1], pNoiseThreshold[1],
false, &Dummy)

        for( i = 0L  i < VecLen  i++ )
            if( Vec[i] <= pNoiseThreshold[1] )
                Vec[i] = -Vec[i]
        Vec[0] = Vec[VecLen-1] = -1.0
        GetClassificationMeasure(Vec, VecLen, NumActiveSections+1, AvgActiveSectionLen+1,
TotalActiveSectionLen+1, ActiveInactiveRatio+1)
```

```
1125
1126        if (mProcessData.mpLogFile)
1127        {
1128            fprintf(mProcessData.mpLogFile, "\tClassification:\n")
1129            fprintf(mProcessData.mpLogFile,
    "\tSignal\tNumActiveSections\tAvgActiveSectionLen\tTotalActiveSectionLen\tActiveInactive
    Ratio\n")
1130            for (i=0  i<2  i++)
1131                fprintf(mProcessData.mpLogFile, "\t%d\t%d\t%d\t%d\t%.2f\n", i, NumActiveSections[i],
    AvgActiveSectionLen[i], TotalActiveSectionLen[i], (float)ActiveInactiveRatio[i])
1132            fprintf(mProcessData.mpLogFile, "\tRatio Deg / Ratio Ref:\t%.3f\n",
    (float)(ActiveInactiveRatio[1] / ActiveInactiveRatio[0]))
1133        }
1134
1135        if (pNoiseLevel[1]<1000.0 && ActiveInactiveRatio[1] / ActiveInactiveRatio[0] > 1.7)
1136            *ClickLevelOfDeg = 1.0
1137        else
1138            *ClickLevelOfDeg = 0.0
1139
1140        matFree(Vec)
1141    }
1142 }
1143
1144 int CSpeechActiveFrameDetection::CalculateActivityFlags(int Signal, int Channel, int* pFlagBuffer,
    int VecLen)
1145 {
1146     int i=0
1147     int PotentialStartFrame=0
1148
1149     OTA_FLOAT* pNoiseLevel     = &mNoiseLevels[Signal][Channel]
1150     OTA_FLOAT* pSignalLevel    = &mSignalLevels[Signal][Channel]
1151     OTA_FLOAT* pNoiseThreshold = &mNoiseThresholds[Signal][Channel]
1152     OTA_FLOAT ClicklevelOfDeg=0
1153
1154     if (Signal==1)
1155     {
1156         ClassifyDistortion(Channel, &ClicklevelOfDeg)
1157         if (mProcessData.mpLogFile)
1158             fprintf(mProcessData.mpLogFile, "\tDetected click level: %0.2f\n",
    (float)ClicklevelOfDeg)
1159     }
1160
1161     VecLen = (((VecLen) < ((int)mFeatureList.GetFVector(0, Signal, Channel)->mSize)) ? (VecLen) :
    ((int)mFeatureList.GetFVector(0, Signal, Channel)->mSize))
1162
1163     OTA_FLOAT* Vec = (OTA_FLOAT*)matMalloc(VecLen * sizeof(OTA_FLOAT))
1164     matbCopy(mFeatureList.GetFVector(0, Signal, Channel)->mpVector, Vec, VecLen)
1165
1166     OTA_FLOAT* PitchVec=0
1167
1168     GetNoiseThreshold(Vec+PotentialStartFrame, VecLen-PotentialStartFrame,  pNoiseLevel,
    pSignalLevel,  pNoiseThreshold)
1169     if (ClicklevelOfDeg) *pNoiseThreshold += (*pSignalLevel-*pNoiseThreshold) / 40.0
1170     IdentifyActivity (Vec, PitchVec, VecLen, *pNoiseLevel, *pSignalLevel, *pNoiseThreshold,
    Signal==0)
1171
1172     for (i=0  i<VecLen  i++)
1173     {
1174         if (Vec[i]>0)
1175             pFlagBuffer[i] = 1
1176         else
1177             pFlagBuffer[i] = 0
1178     }
1179
1180     if (mProcessData.mpLogFile)
1181         fprintf(mProcessData.mpLogFile, "\tInitialy measured
    levels:\tSignal=%.1fdB,\tNoise=%.1fdB,\tThreshold=%.1fdB\n",
    (float)(10.0*log10(*pSignalLevel)), (float)(10.0*log10(*pNoiseLevel)),
```

```
    (float)(10.0*log10(*pNoiseThreshold)))
1182
1183        i=VecLen-1
1184        while ( i>0 && !pFlagBuffer[i]) i--
1185        if (VecLen / (VecLen-i) <= 3)
1186        {
1187            matbCopy(mFeatureList.GetFVector(0, Signal, Channel)->mpVector, Vec,
    VecLen-PotentialStartFrame)
1188            GetNoiseThreshold(Vec+PotentialStartFrame, i-PotentialStartFrame,  pNoiseLevel,
    pSignalLevel,  pNoiseThreshold)
1189            if (ClicklevelOfDeg) *pNoiseThreshold += (*pSignalLevel-*pNoiseThreshold) / 40.0
1190            IdentifyActivity (Vec, PitchVec, VecLen, *pNoiseLevel, *pSignalLevel, *pNoiseThreshold,
    Signal==0)
1191
1192            for (i=0  i<VecLen  i++)
1193            {
1194                if (Vec[i]>0)
1195                    pFlagBuffer[i] = 1
1196                else
1197                    pFlagBuffer[i] = 0
1198            }
1199            if (mProcessData.mpLogFile)
1200                fprintf(mProcessData.mpLogFile, "\tAfter requalification:
    \tSignal=%.1fdB,\tNoise=%.1fdB,\tThreshold=%.1fdB\n",
    (float)(10.0*log10(*pSignalLevel)), (float)(10.0*log10(*pNoiseLevel)),
    (float)(10.0*log10(*pNoiseThreshold)))
1201        }
1202
1203        OTA_FLOAT SnrPerSection[100]
1204        OTA_FLOAT Noiselevel[100]
1205        OTA_FLOAT SignalLevel[100]
1206        OTA_FLOAT Threshold[100]
1207
1208        for(i=0 i<100 i++)Threshold[i]=0
1209
1210        int NumSections=0
1211        for (i=1  i<VecLen  i++)
1212        {
1213            if ( pFlagBuffer[i] && !pFlagBuffer[i-1] && NumSections<100)
1214            {
1215                SnrPerSection[NumSections] = GetSectionSnrIndB(Signal, Channel, i,
    Noiselevel+NumSections, SignalLevel+NumSections, Threshold+NumSections)
1216                NumSections++
1217            }
1218        }
1219
1220        bool SNRVariationDetected = false
1221        OTA_FLOAT AvgSnr=0
1222        for (i=0  i<NumSections  i++)
1223            AvgSnr += SnrPerSection[i]
1224        AvgSnr = NumSections > 0 ? AvgSnr/NumSections : (OTA_FLOAT)0.0
1225        for (i=1  i<NumSections  i++)
1226            if (fabs(SnrPerSection[i]-AvgSnr) > 7.0)
1227                SNRVariationDetected = true
1228
1229        OTA_FLOAT MaxThreshold = Threshold[0]
1230        OTA_FLOAT MinThreshold = Threshold[0]
1231        for (i=1  i<NumSections  i++)
1232        {
1233            MaxThreshold = (((MaxThreshold) > (Threshold[i])) ? (MaxThreshold) : (Threshold[i]))
1234            MinThreshold = (((MinThreshold) < (Threshold[i])) ? (MinThreshold) : (Threshold[i]))
1235        }
1236        if ((Signal!=0 || AvgSnr<25) && (MaxThreshold-MinThreshold>3 || SNRVariationDetected))
1237        {
1238            if (mProcessData.mpLogFile)
1239                fprintf(mProcessData.mpLogFile, "\tRequalifiyng thresholds and levels per section\n")
1240
1241            for (i=1  i<VecLen  i++)
```

```
        {
            if (pFlagBuffer[i] && !pFlagBuffer[i-1])
            {
                PotentialStartFrame = SearchStartFrame(Signal, Channel, i)
                if (PotentialStartFrame>i)
                {
                    for (i--  i<PotentialStartFrame  i++)
                        pFlagBuffer[i] = false
                }
                else
                {
                    for (int j=PotentialStartFrame  j<i  j++)
                        pFlagBuffer[j] = true
                }
            }
        }
    }

    PotentialStartFrame = SearchStartFrame(Signal, Channel, 0)

    matFree(Vec)

    return FramesToSamples(PotentialStartFrame)
}

int CSpeechActiveFrameDetection::GetMaxFrames(int Signal, int Channel)
{
    return mNumFeatureFrames[Signal]
}

void CSpeechActiveFrameDetection::GetLevels(int Signal, int Channel, int Downsampling, OTA_FLOAT*
pNoiseLevel, OTA_FLOAT* pSignalLevel, OTA_FLOAT* pNoiseThreshold, SEGMENT* pSegment)
{

    mppActiveFrameFlags[Signal][Channel] = DEBUG_NEW int[mNumFeatureFrames[Signal]]
    mStartSamples[Signal][Channel] = CalculateActivityFlags(Signal, Channel,
mppActiveFrameFlags[Signal][Channel], mNumFeatureFrames[Signal])
    mDataValidFlags[Signal][Channel] = true

    if (!pSegment)
    {
        *pNoiseLevel     =
mNoiseLevels[Signal][Channel]*((OTA_FLOAT)Downsampling/(OTA_FLOAT)mProcessData.mStepSize)
        *pSignalLevel    =
mSignalLevels[Signal][Channel]*((OTA_FLOAT)Downsampling/(OTA_FLOAT)mProcessData.mStepSize)
        *pNoiseThreshold =
mNoiseThresholds[Signal][Channel]*((OTA_FLOAT)Downsampling/(OTA_FLOAT)mProcessData.mStepSize
)
    }
    else
    {
        int VecStart = pSegment->Start / mProcessData.mStepSize
        int VecEnd = pSegment->End / mProcessData.mStepSize
        int VecLen = mFeatureList.GetFVector(0, Signal, Channel)->mSize
        if (VecEnd<=VecLen)
        {
            VecLen = VecEnd-VecStart
            OTA_FLOAT* Vec = matxMalloc(VecLen)
            matbCopy(mFeatureList.GetFVector(0, Signal, Channel)->mpVector+VecStart, Vec, VecLen)
            GetNoiseThreshold(Vec, VecLen, pNoiseLevel, pSignalLevel, pNoiseThreshold)
            matFree(Vec)
        }

    }
}

OTA_FLOAT CSpeechActiveFrameDetection::GetLevelBelowThreshold(SEGMENT* pSegment, int Signal, int
Channel)
```

```
1303 {
1304
1305     mppActiveFrameFlags[Signal][Channel] = DEBUG_NEW int[mNumFeatureFrames[Signal]]
1306     mStartSamples[Signal][Channel] = CalculateActivityFlags(Signal, Channel,
     mppActiveFrameFlags[Signal][Channel], mNumFeatureFrames[Signal])
1307     mDataValidFlags[Signal][Channel] = true
1308
1309     OTA_FLOAT NoiseThreshold = mNoiseThresholds[Signal][Channel]
1310
1311     OTA_FLOAT  AverageEnergy=0
1312     int        AverageCount=0
1313     int VecStart = pSegment->Start / mProcessData.mStepSize
1314     int VecEnd = pSegment->End / mProcessData.mStepSize
1315     int VecLen = mFeatureList.GetFVector(0, Signal, Channel)->mSize
1316     if (VecEnd<=VecLen)
1317     {
1318         VecLen = VecEnd-VecStart
1319         OTA_FLOAT* Vec = mFeatureList.GetFVector(0, Signal, Channel)->mpVector
1320
1321         for (int i=VecStart  i<VecEnd  i++)
1322             if (Vec[i]<NoiseThreshold)
1323                 {AverageEnergy+=Vec[i]  AverageCount++ }
1324         if (AverageCount)
1325             AverageEnergy /= AverageCount
1326         else
1327             AverageEnergy = NoiseThreshold
1328     }
1329     return AverageEnergy
1330 }
1331
1332 int CSpeechActiveFrameDetection::GetStartSample(int Signal, int Channel, int EarliestSamples)
1333 {
1334
1335     mppActiveFrameFlags[Signal][Channel] = DEBUG_NEW int[mNumFeatureFrames[Signal]]
1336     mStartSamples[Signal][Channel] = CalculateActivityFlags(Signal, Channel,
     mppActiveFrameFlags[Signal][Channel], mNumFeatureFrames[Signal])
1337     mDataValidFlags[Signal][Channel] = true
1338
1339     return SearchStartFrame(Signal, Channel, EarliestSamples/mProcessData.mStepSize) *
     mProcessData.mStepSize
1340 }
1341
1342 int CSpeechActiveFrameDetection::GetStartFrame(int Signal, int Channel, int Downsampling, int
     EarliestSamples)
1343 {
1344
1345     mppActiveFrameFlags[Signal][Channel] = DEBUG_NEW int[mNumFeatureFrames[Signal]]
1346     mStartSamples[Signal][Channel] = CalculateActivityFlags(Signal, Channel,
     mppActiveFrameFlags[Signal][Channel], mNumFeatureFrames[Signal])
1347     mDataValidFlags[Signal][Channel] = true
1348
1349     return (int)((OTA_FLOAT)SearchStartFrame(Signal, Channel,
     EarliestSamples/mProcessData.mStepSize) /
     ((OTA_FLOAT)Downsampling/(OTA_FLOAT)mProcessData.mStepSize))
1350 }
1351
1352 int CSpeechActiveFrameDetection::GetLastActiveFrame(int Signal, int Channel, int Downsampling, int
     EarliestSamples)
1353 {
1354     if (!mDataValidFlags[Signal][Channel])
1355     {
1356         mppActiveFrameFlags[Signal][Channel] = DEBUG_NEW int[mNumFeatureFrames[Signal]]
1357         mStartSamples[Signal][Channel] = CalculateActivityFlags(Signal, Channel,
     mppActiveFrameFlags[Signal][Channel], mNumFeatureFrames[Signal])
1358         mDataValidFlags[Signal][Channel] = true
1359     }
1360
1361     int LastActiveFrame = mNumFeatureFrames[Signal]-1
```

```
1362     while(!mppActiveFrameFlags[Signal][Channel][LastActiveFrame--])
1363     LastActiveFrame++
1364
1365     return (int)(
(OTA_FLOAT)LastActiveFrame*(OTA_FLOAT)mProcessData.mStepSize/(OTA_FLOAT)Downsampling )
1366 }
1367
1368 int CSpeechActiveFrameDetection::GetActiveFrameFlags(int Signal, int Channel, int Downsampling, int*
pFlags, int MaxNumFlags, int EarliestSample)
1369 {
1370     if (!mDataValidFlags[Signal][Channel])
1371     {
1372         mppActiveFrameFlags[Signal][Channel] = DEBUG_NEW int[mNumFeatureFrames[Signal]]
1373         mStartSamples[Signal][Channel] = CalculateActivityFlags(Signal, Channel,
mppActiveFrameFlags[Signal][Channel], mNumFeatureFrames[Signal])
1374         mDataValidFlags[Signal][Channel] = true
1375     }
1376
1377     int* pActiveFrameFlags=0
1378     pActiveFrameFlags = mppActiveFrameFlags[Signal][Channel]
1379
1380     if (pActiveFrameFlags)
1381     {
1382         int i
1383         int EarliestFrame = EarliestSample / mProcessData.mStepSize
1384
1385         if (Downsampling<mProcessData.mStepSize)
1386         {
1387             OTA_FLOAT fDownsampling = (OTA_FLOAT)Downsampling / (OTA_FLOAT)mProcessData.mStepSize
1388             int Hangover = mProcessData.mStepSize / Downsampling /2
1389             bool LastFrameWasActive=false
1390             for (i=EarliestFrame  i<MaxNumFlags  i++)
1391             {
1392                 int ff = (int)((OTA_FLOAT)i * fDownsampling + (OTA_FLOAT)0.5)
1393                 if (ff<mNumFeatureFrames[Signal])
1394                 {
1395                     if (LastFrameWasActive&&!pActiveFrameFlags[ff])
1396                     {
1397                         int Start = (((EarliestFrame) > (i-Hangover)) ? (EarliestFrame) :
(i-Hangover))
1398                         for (int j=Start  j<=i  j++)
1399                             pFlags[j] = 0
1400                         LastFrameWasActive = false
1401                     }
1402                     else if (!LastFrameWasActive&&pActiveFrameFlags[ff])
1403                     {
1404                         int End = i+Hangover
1405                         for ( i<MaxNumFlags && i<End  i++)
1406                             pFlags[i] = 0
1407                         i++
1408                         if (i<MaxNumFlags) pFlags[i] = 1
1409                         LastFrameWasActive = true
1410                     }
1411                     else LastFrameWasActive = pFlags[i] = pActiveFrameFlags[ff]
1412                 }
1413                 else pFlags[i] = 0
1414             }
1415
1416             return (((MaxNumFlags) < ((int)(mNumFeatureFrames[Signal]* fDownsampling))) ?
(MaxNumFlags) : ((int)(mNumFeatureFrames[Signal]* fDownsampling)))
1417         }
1418         else
1419         {
1420             Downsampling /= mProcessData.mStepSize
1421
1422             int CenterOffset = Downsampling / 2 - 1
1423             if (Downsampling==1) CenterOffset = 0
1424
```

```
1425            pFlags[0] = 0
1426
1427            int LastFrame = (((MaxNumFlags) < ((mNumFeatureFrames[Signal]-EarliestFrame-CenterOffset-1)/Downsampling)) ? (MaxNumFlags) : ((mNumFeatureFrames[Signal]-EarliestFrame-CenterOffset-1)/Downsampling))
1428            pFlags[0] = 0
1429            for (int mf=0  mf<LastFrame  mf++)
1430            {
1431                int Sum = 0
1432                int StartFeatureFrame = (((0) > (mf * Downsampling - CenterOffset)) ? (0) : (mf * Downsampling - CenterOffset))
1433                for (int s=0  s<Downsampling  s++)
1434                    Sum += pActiveFrameFlags[StartFeatureFrame+s+EarliestFrame]
1435                pFlags[mf] = Sum>CenterOffset ? 1 : 0
1436            }
1437
1438            for (i=LastFrame  i<MaxNumFlags  i++)
1439                pFlags[i] = 0
1440
1441            return LastFrame-1
1442        }
1443    }
1444    else
1445    {
1446        for (int i=0  i<MaxNumFlags  i++)
1447            pFlags[i] = 1
1448
1449        return MaxNumFlags
1450    }
1451 }
1452
1453 void CSpeechActiveFrameDetection::ImproveSegments(SEGMENT* pSegments)
1454 {
1455    for (int Signal=0  Signal<2  Signal++)
1456    {
1457        int Channel = 0
1458        if (!mDataValidFlags[Signal][Channel])
1459        {
1460            mppActiveFrameFlags[Signal][Channel] = DEBUG_NEW int[mNumFeatureFrames[Signal]]
1461            mStartSamples[Signal][Channel] = CalculateActivityFlags(Signal, Channel, mppActiveFrameFlags[Signal][Channel], mNumFeatureFrames[Signal])
1462            mDataValidFlags[Signal][Channel] = true
1463        }
1464    }
1465
1466    const OTA_FLOAT LowSNRdB=3
1467    OTA_FLOAT SNRdB = 10*log10(mSignalLevels[1][0]/mNoiseLevels[1][0])
1468
1469    if (SNRdB>8 && SNRdB<20)
1470    {
1471        OTA_FLOAT* pVecRef = mFeatureList.GetFVector(0, 0, 0)->mpVector
1472        OTA_FLOAT* pVecDeg = mFeatureList.GetFVector(0, 1, 0)->mpVector
1473
1474        OTA_FLOAT TriggerLevelRef = mSignalLevels[0][0]*2.0
1475        OTA_FLOAT TriggerLevelDeg = mSignalLevels[1][0]*2.0
1476
1477        int TriggerPointRef
1478        int TriggerEndRef   = (((SamplesToFrames(pSegments[0].End)) < (mFeatureList.GetFVector(0, 0, 0)->mSize)) ? (SamplesToFrames(pSegments[0].End)) : (mFeatureList.GetFVector(0, 0, 0)->mSize))
1479        do
1480        {
1481            TriggerPointRef = (((SamplesToFrames(pSegments[0].Start)) < (mFeatureList.GetFVector(0, 0, 0)->mSize)) ? (SamplesToFrames(pSegments[0].Start)) : (mFeatureList.GetFVector(0, 0, 0)->mSize))
1482            while(TriggerPointRef<TriggerEndRef && pVecRef[TriggerPointRef]<TriggerLevelRef) TriggerPointRef++
1483            TriggerLevelRef *= 0.8
```

```
1484        } while (TriggerPointRef>=TriggerEndRef)
1485
1486        int TriggerPointDeg
1487        int TriggerEndDeg   = (((SamplesToFrames(pSegments[0].End)) < (mFeatureList.GetFVector(0, 0,
    0)->mSize)) ? (SamplesToFrames(pSegments[0].End)) : (mFeatureList.GetFVector(0, 0,
    0)->mSize))
1488        do
1489        {
1490            TriggerPointDeg = (((SamplesToFrames(pSegments[0].Start)) < (mFeatureList.GetFVector(0,
    0, 0)->mSize)) ? (SamplesToFrames(pSegments[0].Start)) : (mFeatureList.GetFVector(0, 0,
    0)->mSize))
1491            while(TriggerPointDeg<TriggerEndDeg && pVecDeg[TriggerPointDeg]<TriggerLevelDeg)
    TriggerPointDeg++
1492            TriggerLevelDeg *= 0.8
1493        } while (TriggerPointDeg>=TriggerEndDeg)
1494
1495        int DelayBetweenSegments = FramesToSamples(TriggerPointRef-TriggerPointDeg)
1496        int Correction = -pSegments[1].Start + pSegments[0].Start - DelayBetweenSegments
1497
1498        if (mProcessData.mpLogFile)
1499            fprintf(mProcessData.mpLogFile, "---> Shifting deg segment by %d samples (%dms)\n",
    Correction, SamplesToMSeconds(Correction) )
1500
1501        pSegments[1].Start += Correction
1502        pSegments[1].End   += Correction
1503    }
1504 }
1505 }
1506
1507
```