```
    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         }
```

```
400         else
401         {
402             matFree(mpStartSampleUtterance)
403             mpStartSampleUtterance = NULL
404         }
405         if (src->mpStopSampleUtterance != NULL && mNumUtterances > 0)
406         {
407             matFree(mpStopSampleUtterance)
408             mpStopSampleUtterance = (int*)matMalloc(mNumUtterances * sizeof(int))
409             for (int i = 0  i < mNumUtterances  i++)
410                 mpStopSampleUtterance[i] = src->mpStopSampleUtterance[i]
411         }
412         else
413         {
414             matFree(mpStopSampleUtterance)
415             mpStopSampleUtterance = NULL
416         }
417         if (src->mpDelayUtterance != NULL && mNumUtterances > 0)
418         {
419             matFree(mpDelayUtterance)
420             mpDelayUtterance = (int*)matMalloc(mNumUtterances * sizeof(int))
421             for (int i = 0  i < mNumUtterances  i++)
422                 mpDelayUtterance[i] = src->mpDelayUtterance[i]
423         }
424         else
425         {
426             matFree(mpDelayUtterance)
427             mpDelayUtterance = NULL
428         }
429 
430         mNumSections = src->mNumSections
431         if (src->mpRefSections != NULL && mNumSections > 0)
432         {
433             delete[] mpRefSections
434             mpRefSections = new SECTION[mNumSections]
435             for (int i = 0  i < mNumSections  i++)
436                 mpRefSections[i].CopyFrom(src->mpRefSections[i])
437         }
438         else
439         {
440             delete[] mpRefSections
441             mpRefSections = NULL
442         }
443         if (src->mpDegSections != NULL && mNumSections > 0)
444         {
445             delete[] mpDegSections
446             mpDegSections = new SECTION[mNumSections]
447             for (int i = 0  i < mNumSections  i++)
448                 mpDegSections[i].CopyFrom(src->mpDegSections[i])
449         }
450         else
451         {
452             delete[] mpDegSections
453             mpDegSections = NULL
454         }
455 
456         mSNRRefdB = src->mSNRRefdB
457         mSNRDegdB = src->mSNRDegdB
458         mNoiseLevelRef = src->mNoiseLevelRef
459         mNoiseLevelDeg = src->mNoiseLevelDeg
460         mSignalLevelRef = src->mSignalLevelRef
461         mSignalLevelDeg = src->mSignalLevelDeg
462         mNoiseThresholdRef = src->mNoiseThresholdRef
463         mNoiseThresholdDeg = src->mNoiseThresholdDeg
464 
465         if (src->mpActiveFrameFlags != NULL && mNumFrames > 0)
466         {
467             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 FILE* pLogFile=0
677
678 CTempAlignment::CTempAlignment()
679 {
680     mpFeatureList = 0
681     mpFeatureList2 = 0
682     mpDelaySearch = 0
683     mpActiveFrameDetection = 0
684     mppSignals = 0
685     mpDelayInSamplesPerFrame = 0
686     mpReliabilityPerFrame = 0
687     mpResults = 0
688     mpReparsePoints = 0
689     mStartOffset = 0
690
691     mpSmartBufferPool = new SmartBufferPool(7)
692 }
693
694 CTempAlignment::~CTempAlignment()
695 {
696     delete mpSmartBufferPool
697 }
698
699 bool CTempAlignment::Init(CProcessData* pProcessData, CDelaySearch* pDelaySearch,
    CActiveFrameDetection* pActiveFrameDetection)
700 {
701     bool rc = true
702
703     Free()
704
705     mpDelaySearch = pDelaySearch
706     mpActiveFrameDetection = pActiveFrameDetection
707     pProcessData->mWindowSize = pProcessData->mpWindowSize[0]
708     pProcessData->mOverlap = pProcessData->mpOverlap[0]
709     pProcessData->Init(0, 1.0)
710     mppSignals = new CTASignal* [2]
711     for (int i=0  i<2  i++)
712         mppSignals[i] = 0
713     pProcessData->mNumSignals = 0
714     mProcessData = *pProcessData
715     mpFeatureList = 0
716     mpFeatureList2 = 0
717     mpDelayInSamplesPerFrame = 0
718     mpReliabilityPerFrame = 0
719
720     mpResults = new OTA_RESULT
721     if (mpResults)
722     {
723         mpResults->mNumUtterances = 0
724         mpResults->mpDelayUtterance = 0
725         mpResults->mpStartSampleUtterance = 0
726         mpResults->mpStopSampleUtterance = 0
727         mpResults->mpRefSections = 0
728         mpResults->mpDegSections = 0
729         mpResults->mpActiveFrameFlags = 0
730         mpResults->mAvgReliability = 0.0f
731     }
732     else rc = false
733
734     pLogFile=pProcessData->mpLogFile
735
```

```
736     return rc
737 }
738
739 void CTempAlignment::Free()
740 {
741     if (mpFeatureList) delete mpFeatureList
742     if (mpFeatureList2) delete mpFeatureList2
743     mpFeatureList = NULL
744
745     for (int i=0  mppSignals && i<2  i++)
746         if (mppSignals[i])
747         { delete mppSignals[i]  mppSignals[i]=0  }
748     if (mppSignals)
749         { delete[] mppSignals  mppSignals=0  }
750
751     if (mpDelayInSamplesPerFrame)
752         matFree(mpDelayInSamplesPerFrame)
753     mpDelayInSamplesPerFrame = NULL
754
755     if (mpResults)
756     {
757         delete mpResults
758         mpResults = 0
759     }
760     if (mpReparsePoints) delete[] mpReparsePoints
761     mpReparsePoints = 0
762 }
763
764
765 //Find the pattern pA in the buffer pB. Returns the position of the max and sets pReliability
766 //to the correlation at the maximum.
767 //The calculated delay is the delay where section A is found in section B, relative to the start of
    section B.
768 //If sections exceed the buffer limits, a new and sufficiently long buffer will be allocated and
    used.
769 //This buffer will be zero padded as needed. Passing negative section starts or ends that lie beyond
    the
770 //last valid data are therefore allowed. Care must however be taken if this happens for section A
    since
771 //the zero padding may skew the found delay.
772 //If MaxDelay is set to 0, the maximum possible search range will be used.
773 //NOTE 1: The selected feature must exist and must have been initialised properly before calling
    this.
774
775 int CTempAlignment::FindSectionAInSectionB(SECTION* pA, SECTION*pB, CFeatureVector* pVecA,
    CFeatureVector* pVecB, OTA_FLOAT* pReliability, int HistoLen, int HistoShift, int MaxDelay, bool
    PlotMe)
776 {
777     return FindSectionAInSectionB(pA, pB, pVecA->mpVector, pVecA->mSize, pVecB->mpVector,
    pVecB->mSize, pReliability, HistoLen, HistoShift, MaxDelay, PlotMe)
778 }
779
780 int CTempAlignment::FindSectionAInSectionB(SECTION* pA, SECTION*pB, OTA_FLOAT* pSigA, int LenA,
    OTA_FLOAT* pSigB, int LenB, OTA_FLOAT* pReliability, int HistoLen, int HistoShift, int MaxDelay,
    bool PlotMe)
781 {
782
783     int DelayOfA
784     OTA_FLOAT* pCCF=0
785     OTA_FLOAT R1=-1
786     OTA_FLOAT R2=-1
787
788     SECTION SecA = *pA
789     SECTION SecB = *pB
790
791     assert(LenA>2)
792     assert(LenB>2)
793     assert(SecA.Start<=LenA)
```

```
794     assert(SecB.Start<=LenB)
795     assert(SecA.Start<SecA.End-2)
796     assert(SecB.Start<SecB.End-2)
797
798     int OffsetA=0
799     OTA_FLOAT* pUsedSigA=pSigA
800     bool MustDeletepUsedSigA=false
801     if (SecA.Start<0 || SecA.End+HistoLen*HistoShift>LenA)
802     {
803         int NewLen = SecA.End-SecA.Start+(HistoLen*HistoShift)
804         pUsedSigA = matxMalloc(NewLen)
805         MustDeletepUsedSigA = true
806         int SrcRangeStart = (((0) > (SecA.Start)) ? (0) : (SecA.Start))
807         int SrcRangeEnd = (((LenA) < (SecA.End+(HistoLen*HistoShift))) ? (LenA) :
    (SecA.End+(HistoLen*HistoShift)))
808         int DestPos=0
809         if (SecA.Start<0)
810         {
811             matbZero(pUsedSigA, -SecA.Start)
812             DestPos += -SecA.Start
813             OffsetA = -SecA.Start
814         }
815         SecA.Start = 0
816         matbCopy(pSigA+SrcRangeStart, pUsedSigA+DestPos, SrcRangeEnd-SrcRangeStart)
817         DestPos += SrcRangeEnd-SrcRangeStart
818         if (DestPos<NewLen) matbZero(pUsedSigA+DestPos, NewLen-DestPos)
819         pSigA = pUsedSigA
820         LenA = NewLen
821         SecA.End = LenA-(HistoLen*HistoShift)
822     }
823
824     int OffsetB=0
825     OTA_FLOAT* pUsedSigB=pSigB
826     bool MustDeletepUsedSigB=false
827     if (SecB.Start<0 || SecB.End+(HistoLen*HistoShift)>LenB)
828     {
829         int NewLen = SecB.End-SecB.Start+(HistoLen*HistoShift)
830         pUsedSigB = matxMalloc(NewLen)
831         MustDeletepUsedSigB = true
832         int SrcRangeStart = (((0) > (SecB.Start)) ? (0) : (SecB.Start))
833         int SrcRangeEnd = (((LenB) < (SecB.End+(HistoLen*HistoShift))) ? (LenB) :
    (SecB.End+(HistoLen*HistoShift)))
834         int DestPos=0
835         if (SecB.Start<0)
836         {
837             matbZero(pUsedSigB, -SecB.Start)
838             DestPos += -SecB.Start
839             OffsetB = SecB.Start
840         }
841         SecB.Start = 0
842
843         matbCopy(pSigB+SrcRangeStart, pUsedSigB+DestPos, SrcRangeEnd-SrcRangeStart)
844         DestPos += SrcRangeEnd-SrcRangeStart
845         if (DestPos<NewLen) matbZero(pUsedSigB+DestPos, NewLen-DestPos)
846         pSigB = pUsedSigB
847         LenB = NewLen
848         SecB.End = LenB-(HistoLen*HistoShift)
849     }
850
851     int SpaceToEndA = (((0) > (LenA-SecA.Start)) ? (0) : (LenA-SecA.Start))
852     int SpaceToEndB = (((0) > (LenB-SecB.Start)) ? (0) : (LenB-SecB.Start))
853     int NumFFramesA = (((SpaceToEndB) < (SecA.End-SecA.Start)) ? (SpaceToEndB) :
    (SecA.End-SecA.Start))
854     int NumFFramesB = (((SpaceToEndB) < (SecB.End-SecB.Start)) ? (SpaceToEndB) :
    (SecB.End-SecB.Start))
855
856     if (NumFFramesA>=NumFFramesB)
857     {
```

```
858         NumFFramesB = (((SpaceToEndB) < (SecB.End-SecB.Start)) ? (SpaceToEndB) :
    (SecB.End-SecB.Start))
859         NumFFramesA = (((SpaceToEndA) < (SecA.End-SecA.Start)) ? (SpaceToEndA) :
    (SecA.End-SecA.Start))
860         if (MaxDelay==0)
861         {
862             MaxDelay = SpaceToEndA-(HistoLen*HistoShift)
863             MaxDelay = (((MaxDelay) < (NumFFramesA-NumFFramesB)) ? (MaxDelay) :
    (NumFFramesA-NumFFramesB))
864         }
865
866         int DelayOfB
867         pCCF = new OTA_FLOAT[NumFFramesA]
868         DelayOfB = FindDelay((MAT_HANDLE)mProcessData.mpMathlibHandle, pSigB+SecB.Start,
    NumFFramesB, pSigA+SecA.Start, NumFFramesA, HistoLen, HistoShift, MaxDelay, &R1, PlotMe)
869         DelayOfB = DelayOfB - OffsetA - OffsetB
870         DelayOfA = -DelayOfB
871         if(pReliability) *pReliability = R1
872     }
873     else
874     {
875
876         if (MaxDelay==0)
877         {
878             MaxDelay = SpaceToEndB-(HistoLen*HistoShift)
879             MaxDelay = (((MaxDelay) < (NumFFramesB-NumFFramesA)) ? (MaxDelay) :
    (NumFFramesB-NumFFramesA))
880         }
881
882         pCCF = new OTA_FLOAT[NumFFramesB]
883         DelayOfA = FindDelay((MAT_HANDLE)mProcessData.mpMathlibHandle, pSigA+SecA.Start,
    NumFFramesA, pSigB+SecB.Start, NumFFramesB, HistoLen, HistoShift, MaxDelay, &R1, PlotMe)
884         DelayOfA = DelayOfA + OffsetA + OffsetB
885         if(pReliability) *pReliability = R1
886     }
887
888     if (MustDeletepUsedSigA) matFree(pUsedSigA)
889     if (MustDeletepUsedSigB) matFree(pUsedSigB)
890
891     return DelayOfA
892
893 }
894
895
896 OTA_FLOAT CTempAlignment::GetSampleRateRatioDiff(int* pActiveFrameFlags, long* DelayVector, int
    DelayVecLen)
897 {
898     int MaxDelayStep = 0.1*mProcessData.mStepSize
899
900     long* DelayVecVirtual = (long*)matMalloc(sizeof(long)*DelayVecLen)
901     int* ActiveFrameFlagsVirtual = (int*)matMalloc(sizeof(int)*DelayVecLen)
902
903     ActiveFrameFlagsVirtual[0] = pActiveFrameFlags[0]
904     DelayVecVirtual[0] = DelayVector[0]
905     int StartFrame = 1
906     for (int i=1  i<DelayVecLen && !pActiveFrameFlags[i]  i++)
907     {
908         ActiveFrameFlagsVirtual[i] = 0
909         DelayVecVirtual[i] = DelayVector[i]
910         StartFrame++
911     }
912
913     int SilenceStart = StartFrame-1
914     bool InSilence = false
915     for (int i=StartFrame  i<DelayVecLen  i++)
916     {
917         if (pActiveFrameFlags[i])
918         {
```

```
            if (InSilence)
            {
                if (SilenceStart==0) SilenceStart++

                long AvgDelayStep = 0
                long ScaleFactor = 64

                long SumDelays=0
                int NumDelays=0
                for (int j=SilenceStart  j<i  j++)
                {
                    SumDelays += (DelayVector[j] - DelayVector[j-1])  * ScaleFactor
                    NumDelays++
                }
                long AvgDelayStepPerFrame=SumDelays / (i-SilenceStart)

                for (int j=SilenceStart  j<i  j++)
                {
                    ActiveFrameFlagsVirtual[j] = 1
                    DelayVecVirtual[j] = (DelayVecVirtual[j-1]*ScaleFactor + AvgDelayStepPerFrame) /
ScaleFactor
                }

                ActiveFrameFlagsVirtual[i] = 1
                DelayVecVirtual[i] = DelayVector[i]

            }
            else
            {
                ActiveFrameFlagsVirtual[i] = pActiveFrameFlags[i]
                DelayVecVirtual[i] = DelayVector[i]
            }
            InSilence = false
        }
        else
        {
            if (InSilence)
            {
                ActiveFrameFlagsVirtual[i] = pActiveFrameFlags[i]
            }
            else
            {
                SilenceStart = i
            }
            InSilence = true
        }
    }

    if (InSilence)
        DelayVecLen = SilenceStart

    int LastValid = 0
    int NewLen = 1
    long LastDelay = DelayVecVirtual[0]
    DelayVecVirtual[0] = 0

    int SumSkippedSteps=0
    int NumSkippedSteps=0
    int SumSteps=0
    int NumSteps=0

    for (int i=1  i<DelayVecLen  i++)
    {
        ActiveFrameFlagsVirtual[NewLen] = ActiveFrameFlagsVirtual[i]
        int FramesSinceLastValid=i-LastValid
        long Step = DelayVecVirtual[i]-LastDelay
        LastDelay = DelayVecVirtual[i]
        if (!ActiveFrameFlagsVirtual[i] || !ActiveFrameFlagsVirtual[i-1] || abs(Step)>MaxDelayStep)
```

```
        {
            if (abs(Step)<18*MaxDelayStep && (ActiveFrameFlagsVirtual[i] &&
ActiveFrameFlagsVirtual[i-1]))
            {
                SumSkippedSteps+=Step
                NumSkippedSteps++
            }
        }
        else
        {
            DelayVecVirtual[NewLen] = DelayVecVirtual[LastValid] + Step
            SumSteps += Step
            LastValid = NewLen
            NewLen++
        }
    }

    OTA_FLOAT rc=-1

    OTA_FLOAT Threshold = mMacroFrameSize*0.0035*(NewLen+NumSkippedSteps)

    int SumAllSteps = abs(SumSkippedSteps+SumSteps)
    if (abs(SumAllSteps)>(int)Threshold || NewLen>3.5*NumSkippedSteps)
    {
        rc = GetSampleRateRatioAbs(ActiveFrameFlagsVirtual, DelayVecVirtual, NewLen)
    }
    else
    {
        rc = -1
    }

    matFree(DelayVecVirtual)
    matFree(ActiveFrameFlagsVirtual)
    return rc
}

OTA_FLOAT CTempAlignment::GetSampleRateRatioAbs(int* pActiveFrameFlags, long* DelayVector, int
DelayVecLen)
{

    OTA_FLOAT const MIN_RATIO_OF_FRAMES_FITTING_LINEAR_MODEL = 0.9

    long delaySpan = matMax(DelayVector, DelayVecLen) - matMin(DelayVector, DelayVecLen)
    if (delaySpan < MSecondsToSamples(32))
        return 1.0

    OTA_FLOAT const MAX_DIST_LINE_SAMPLES = ((((int)(0.2 * delaySpan)) < (MSecondsToSamples(32))) ?
((int)(0.2 * delaySpan)) : (MSecondsToSamples(32)))

    int const MIN_FRAMES_INTER_SENTENCE_PAUSE = 15
    int const MIN_FRAMES_IN_SENTENCE = 30
    int const NUM_FRAMES_TOLERANCE = 4

    OTA_FLOAT SamplerateRatio = -1.0
    bool isConstantSampleRate = true

    int numPartialSR = 0

    OTA_FLOAT xy_sum = 0.0, x_sum = 0.0, y_sum = 0.0, x2_sum = 0.0
    int numActFrames = 0
    for (int i=0  i<DelayVecLen  i++)
    {
        if (pActiveFrameFlags[i])
        {
            numActFrames++
            xy_sum += (OTA_FLOAT)i * DelayVector[i]
            x_sum += (OTA_FLOAT)i
            y_sum += DelayVector[i]
```

```
1051            x2_sum += pow((OTA_FLOAT)i,2.0)
1052        }
1053    }
1054    OTA_FLOAT a = ((OTA_FLOAT)numActFrames*xy_sum - x_sum*y_sum) / ((OTA_FLOAT)numActFrames*x2_sum -
    pow(x_sum,2.0))
1055    OTA_FLOAT b = (y_sum - a*x_sum) / (OTA_FLOAT)numActFrames
1056
1057    OTA_FLOAT* expectedDelay = new OTA_FLOAT[DelayVecLen]
1058    OTA_FLOAT* delayEstimateError = new OTA_FLOAT[DelayVecLen]
1059
1060        int numFramesFittingLine=0
1061        for (int i=0  i<DelayVecLen  i++)
1062        {
1063            expectedDelay[i] = a * (OTA_FLOAT)i + b
1064            if (pActiveFrameFlags[i])
1065            {
1066                delayEstimateError[i] = abs(DelayVector[i] - expectedDelay[i])
1067                if (delayEstimateError[i] <= MAX_DIST_LINE_SAMPLES)
1068                    numFramesFittingLine++
1069            }
1070            else
1071                delayEstimateError[i] = 0
1072        }
1073
1074        if ((OTA_FLOAT)numFramesFittingLine/(OTA_FLOAT)numActFrames >=
    MIN_RATIO_OF_FRAMES_FITTING_LINEAR_MODEL)
1075        {
1076            SamplerateRatio = 1 - a/(OTA_FLOAT)mProcessData.mStepSize
1077
1078        }
1079
1080    delete[] expectedDelay  expectedDelay = NULL
1081    delete[] delayEstimateError  delayEstimateError = NULL
1082
1083    if (abs(1-SamplerateRatio) < 0.001)
1084        SamplerateRatio = 1.0
1085
1086    return SamplerateRatio
1087 }
1088
1089
1090
1091 //Get the coarse delay between the  two input signals.
1092 //The delay is positive if the degraded signal comes before the Reference signal
1093 //The first StartFrame frames are skipped to avoid calculating the delay on a silent intervall
1094 //The begining of the ref signal is searched in the deg signal.
1095
1096 bool CTempAlignment::GetCoarseAvgDelayAtStart(int FeatureIndex, int Channel, int StartFrameLong, int
    StartFrameShort, OTA_FLOAT CoarseAlignSegmentLengthInMs, int MaxDelay, int *AvgDelayInFrames,
    OTA_FLOAT* CorrAtMax, int LongSig, int ShortSig)
1097 {
1098
1099    bool rc = true
1100
1101    CFeatureVector* FVectors[2]
1102    FVectors[LongSig] = mpFeatureList->GetFVector(FeatureIndex, LongSig, Channel,
    FeatureIndex==0?1:0)
1103    FVectors[ShortSig] = mpFeatureList->GetFVector(FeatureIndex, ShortSig, Channel,
    FeatureIndex==0?1:0)
1104
1105    int SegmentLength = (int)(CoarseAlignSegmentLengthInMs * mProcessData.mSamplerate / 1000 /
    mProcessData.mStepSize)
1106
1107    int LagLeft = 0
1108    int Lag = MaxDelay
1109    int DegShift = 0
1110
1111    {
```

```
1112        int MaxIndex=0
1113        SECTION SecA
1114        SECTION SecB
1115        SecA.Start = StartFrameShort+DegShift
1116        SecA.End   = SecA.Start + SegmentLength
1117        SecB.Start = StartFrameLong-LagLeft
1118        SecB.End   = SecB.Start + SegmentLength + Lag
1119
1120        if (SecA.Start<((int)FVectors[ShortSig]->mSize)-10 &&
    SecB.Start<((int)FVectors[LongSig]->mSize)-10)
1121        {
1122            MaxIndex = FindSectionAInSectionB(&SecA, &SecB, FVectors[ShortSig], FVectors[LongSig],
    CorrAtMax, 50, 1)
1123            MaxIndex = MaxIndex - SecA.Start + SecB.Start
1124            *AvgDelayInFrames = -MaxIndex
1125        }
1126        else
1127        {
1128
1129            *AvgDelayInFrames = -10000
1130            *CorrAtMax = -1
1131        }
1132
1133    }
1134
1135    return rc
1136 }
1137
1138 //Get an initial estimate of the delay at each reparse point.
1139 //This estimate is used as a starting point and refined with every alignment iteration.
1140 //The degraded signal is searched in the ref signal.
1141 bool CTempAlignment::GetInitialDelayInSamplesForOneDownsamplingStep(int Point, SEGMENT* pSegment,
    int DownsamplingStep, int* FoundDelayInSamples, OTA_FLOAT* RAtDelay, int MaxInitialDelay)
1142 {
1143    bool rc = true
1144
1145    mProcessData.Init(2, (float)DownsamplingStep)
1146
1147    if (mProcessData.mpLogFile)
1148
1149
1150    MaxInitialDelay /= mProcessData.mStepSize
1151
1152    rc = mpFeatureList->Create(mppSignals, &mProcessData, OTA_FLTYPE_INITIAL_SEARCH, pSegment)
1153    if (rc)
1154    {
1155        int NumFFramesRef = mpFeatureList->GetFVector(0, 0, 0)->mSize
1156
1157        int NextIndex=0
1158        if (rc)
1159        {
1160            const int NumSegmentFactors = 2
1161            const int NumWindowShifts = 2
1162            int f, i, j, FoundDelay
1163            OTA_FLOAT CorrAtMax=0
1164            int FeatureIndex=0
1165            OTA_FLOAT* Correlations = new
    OTA_FLOAT[NumSegmentFactors*NumWindowShifts*mpFeatureList->mNumFeatures]
1166            int* Delays = new int[NumSegmentFactors*NumWindowShifts*mpFeatureList->mNumFeatures]
1167            int* Windows = new int[NumWindowShifts]
1168            int* Segments = new int[NumSegmentFactors]
1169
1170            int WindowShiftUnit=(int)((0.7*mProcessData.mSamplerate)/mProcessData.mStepSize)
1171
1172            for (j=0  j<NumWindowShifts  j++)
1173            {
1174
1175                Windows[j] = (int)((0.1*mProcessData.mSamplerate)/mProcessData.mStepSize) +
```

```
    j*WindowShiftUnit
1176
1177                for (i=0  i<NumSegmentFactors  i++)
1178                {
1179                    OTA_FLOAT SegmentFactor=0.5*(i+1)
1180                    Segments[i] =
    (int)(mProcessData.mP.mTAPara.CoarseAlignSegmentLengthInMs*SegmentFactor)
1181
1182                    for (f=0  f<mpFeatureList->mNumFeatures  f++)
1183                    {
1184
1185                        if (GetCoarseAvgDelayAtStart(f, 0, Windows[j], Windows[j], Segments[i],
    MaxInitialDelay, &FoundDelay, &CorrAtMax, 0, 1))
1186
1187                        {
1188                            Correlations[NextIndex] = CorrAtMax
1189                            Delays[NextIndex] = FoundDelay
1190                        }
1191                        else
1192                        {
1193                            Correlations[NextIndex] = 0
1194                            Delays[NextIndex] = 0
1195                        }
1196                        NextIndex++
1197                    }
1198                }
1199            }
1200
1201            int BestFeature = 0
1202            OTA_FLOAT BestR = Correlations[0]
1203            int Offset = (pSegment[0].Start - pSegment[1].Start) / mProcessData.mStepSize
1204            for (f=0  f<NextIndex  f++)
1205            {
1206
1207                if (BestR<Correlations[f]+0.03)
1208                {
1209                    if (fabs(Correlations[f]-BestR) > 0.03 || abs(Delays[f]+Offset)<
    abs(Delays[BestFeature]+Offset))
1210                    {
1211                        BestFeature = f
1212                        BestR = Correlations[f]
1213                    }
1214                }
1215
1216            }
1217            //Convert the delay from frames to samples and relate it to the degraded signal
1218            *FoundDelayInSamples = -Delays[BestFeature]*mProcessData.mStepSize
1219            *RAtDelay = BestR
1220
1221        }
1222    }
1223    else
1224    {
1225
1226
1227        *FoundDelayInSamples = 0
1228        *RAtDelay = -1
1229    }
1230
1231    return rc
1232 }
1233
1234 //Search through both input signals and identify the initial delay.
1235 //The first OffsetInSamples samples are skipped (but may be used for negative delays)
1236 long CTempAlignment::GetInitialDelayInSamples(int Point, SEGMENT* pSegments, int* pStartSampleRef,
    int *pStartSampleDeg, int* pActiveFrameFlags, OTA_FLOAT* pReliability, int
    *WorstResolutionInSamples, int SectionOffset, int MaxInitialDelay)
1237 {
```

```
1238
1239      bool rc=true
1240      int i
1241      int Delays[2]
1242      OTA_FLOAT Rs[2]
1243      *WorstResolutionInSamples = -1
1244      for (i=0  i<2  i++)
1245      {
1246          Delays[i] = 0  Rs[i] = 0
1247          GetInitialDelayInSamplesForOneDownsamplingStep(Point, pSegments, i+1, &Delays[i], &Rs[i],
      MaxInitialDelay)
1248          if (mProcessData.mStepSize>*WorstResolutionInSamples)
1249              *WorstResolutionInSamples = mProcessData.mStepSize
1250      }
1251      int BestIteration=0
1252      int BestDelay = Delays[0]
1253      OTA_FLOAT BestR = Rs[0]
1254      for (i=1  i<2  i++)
1255      {
1256
1257          if (BestR<Rs[i]+0.03)
1258          {
1259              if (fabs(Rs[i]-BestR) > 0.03 || abs(Delays[i]+SectionOffset)<
      abs(BestDelay+SectionOffset))
1260              {
1261                  BestDelay = Delays[i]
1262                  BestR = Rs[i]
1263                  BestIteration = i
1264              }
1265          }
1266
1267      }
1268
1269      if (BestR<0.6)
1270      {
1271          BestDelay = 0
1272          BestIteration = -1
1273          if (mProcessData.mpLogFile)
1274
1275      }
1276      else if (abs(BestDelay)>MSecondsToSamples(200) && BestR<0.8)
1277      {
1278          BestDelay = 0
1279          BestIteration = -1
1280          if (mProcessData.mpLogFile)
1281
1282      }
1283
1284      *pReliability = BestR
1285      if (BestIteration<0) *pReliability-=0.2
1286
1287      if (mProcessData.mpLogFile)
1288
1289
1290      return BestDelay
1291  }
1292
1293  //For each reparse point identify the delay in samples
1294  //Here we also set the ref start point and do also refine both startpoints.
1295  int CTempAlignment::GetInitialDelaysInSamples(REPARSE_POINT* ParsePoints, int NumParsePoints, int*
      pActiveFrameFlags, int OverallDelayEstimate, OTA_FLOAT OverallDelayEstimateReliability, int
      *AccuracyInSamples)
1296  {
1297      int Point
1298      int LastDelayInSamples=0
1299      SEGMENT Segments[2]
1300
1301      for (Point=0  Point<NumParsePoints  Point++)
```

```
1302     {
1303
1304
1305         if (ParsePoints[Point].Reliability >
     mProcessData.mP.mTAPara.mCorrForSkippingInitialDelaySearch)
1306
1307         {
1308             ParsePoints[Point].DelayInSamples = -ParsePoints[Point].Deg.Start +
     ParsePoints[Point].Ref.Start
1309
1310         }
1311         else
1312         {
1313             Segments[1].Start = ParsePoints[Point].Deg.Start
1314             Segments[1].Start = (((0) > ((((mppSignals[1]->mSignalLength) < (Segments[1].Start)) ?
     (mppSignals[1]->mSignalLength) : (Segments[1].Start)))) ? (0) :
     ((((mppSignals[1]->mSignalLength) < (Segments[1].Start)) ?
     (mppSignals[1]->mSignalLength) : (Segments[1].Start))))
1315             Segments[1].End   = Segments[1].Start + 4*mProcessData.mSamplerate
1316
1317             int MaxInitialDelay = 0
1318
1319             Segments[0].Start = ParsePoints[Point].Ref.Start - 600/1000*mProcessData.mSamplerate
1320             Segments[0].Start = (((0) > (Segments[0].Start-0.1*mProcessData.mSamplerate)) ? (0) :
     (Segments[0].Start-0.1*mProcessData.mSamplerate))
1321
1322             if (Point && Segments[0].Start <
     ParsePoints[Point-1].Deg.End+ParsePoints[Point-1].DelayInSamples)
1323                 Segments[0].Start = (((0) >
     (ParsePoints[Point-1].Deg.End+ParsePoints[Point-1].DelayInSamples)) ? (0) :
     (ParsePoints[Point-1].Deg.End+ParsePoints[Point-1].DelayInSamples))
1324
1325             MaxInitialDelay = (OTA_FLOAT)600*mProcessData.mSamplerate/1000.0
1326
1327             Segments[0].End = Segments[0].Start +  (600+5000)/1000*mProcessData.mSamplerate
1328
1329             Segments[0].End = (((Segments[0].End) < (mppSignals[0]->mSignalLength-1)) ?
     (Segments[0].End) : (mppSignals[0]->mSignalLength-1))
1330             Segments[1].End = (((Segments[1].End) < (mppSignals[1]->mSignalLength-1)) ?
     (Segments[1].End) : (mppSignals[1]->mSignalLength-1))
1331
1332             OTA_FLOAT Reliability
1333             int DelayOffset = Segments[0].Start - Segments[1].Start
1334             long DelayInSamples = GetInitialDelayInSamples(Point, Segments,
     &ParsePoints[Point].Ref.Start, &ParsePoints[Point].Deg.Start, pActiveFrameFlags,
     &Reliability, AccuracyInSamples, DelayOffset, MaxInitialDelay)
1335
1336             DelayInSamples += DelayOffset
1337
1338             if (Reliability>ParsePoints[Point].Reliability)
1339             {
1340                 ParsePoints[Point].DelayInSamples = DelayInSamples
1341                 ParsePoints[Point].Reliability = Reliability
1342
1343             }
1344             else
1345
1346         }
1347
1348     }
1349
1350     return NumParsePoints
1351 }
1352
1353
1354 //If one of the two signals is much longer, it may also contain more active sections than the other.
1355 //Here we are looking for a section which gives a good overall match and we will discard all others
1356 //afterwards.
```

```
1357 //Returns the detected start sample of the active part of the ref signal (minus some guard
     intervall).
1358 int CTempAlignment::FindUsedSectionOfRefSignal()
1359 {
1360     int SectionStart = 0
1361     const int HistoLen = 1
1362     const int HistoShift = 0
1363     SECTION SecA, SecB
1364     mProcessData.Init(2, 1.0)
1365 
1366     if (mProcessData.mpLogFile)
1367 
1368 
1369     if (mppSignals[0]->mSignalLength>2*mppSignals[1]->mSignalLength)
1370     {
1371 
1372         mpFeatureList->Create(mppSignals, &mProcessData, OTA_FLTYPE_INITIAL_SEARCH)
1373         int NumFFramesRef = mpFeatureList->GetFVector(0, 0, 0)->mSize
1374         int NumFFramesDeg = mpFeatureList->GetFVector(0, 1, 0)->mSize
1375         OTA_FLOAT* pRef = mpFeatureList->GetFVector(0, 0, 0, 1)->mpVector
1376         OTA_FLOAT* pDeg = mpFeatureList->GetFVector(0, 1, 0, 1)->mpVector
1377 
1378         int DegDelay
1379         OTA_FLOAT CorrAll
1380         SecA.Start = 0  SecA.End = NumFFramesDeg
1381         SecB.Start = 0  SecB.End = NumFFramesRef
1382         if (SecA.End-SecA.Start>=MSecondsToFrames(1000))
1383         {
1384             DegDelay = FindSectionAInSectionB(&SecA, &SecB, pDeg, NumFFramesDeg, pRef,
     NumFFramesRef, &CorrAll, HistoLen, HistoShift)
1385             DegDelay = DegDelay - SecA.Start+SecB.Start
1386             if (CorrAll>0.8)
1387                 SectionStart = (((0) > (DegDelay-MSecondsToFrames(300))) ? (0) :
     (DegDelay-MSecondsToFrames(300)))
1388             else
1389         }
1390     }
1391     else if (mppSignals[1]->mSignalLength>2*mppSignals[0]->mSignalLength)
1392     {
1393 
1394         mpFeatureList->Create(mppSignals, &mProcessData, OTA_FLTYPE_INITIAL_SEARCH)
1395         int NumFFramesRef = mpFeatureList->GetFVector(0, 0, 0)->mSize
1396         int NumFFramesDeg = mpFeatureList->GetFVector(0, 1, 0)->mSize
1397         OTA_FLOAT* pRef = mpFeatureList->GetFVector(0, 0, 0, 1)->mpVector
1398         OTA_FLOAT* pDeg = mpFeatureList->GetFVector(0, 1, 0, 1)->mpVector
1399 
1400         int DegDelay
1401         OTA_FLOAT CorrAll
1402         SecA.Start = 0  SecA.End = NumFFramesRef
1403         SecB.Start = 0  SecB.End = NumFFramesDeg
1404         if (SecA.End-SecA.Start>=MSecondsToFrames(1000))
1405         {
1406             DegDelay = FindSectionAInSectionB(&SecA, &SecB, pRef, NumFFramesRef, pDeg,
     NumFFramesDeg, &CorrAll, HistoLen, HistoShift)
1407             DegDelay = DegDelay - SecA.Start+SecB.Start
1408             if (CorrAll>0.8)
1409                 SectionStart = -(((0) > (DegDelay-MSecondsToFrames(300))) ? (0) :
     (DegDelay-MSecondsToFrames(300)))
1410             else
1411         }
1412     }
1413     else
1414 
1415     SectionStart = FramesToSamples(SectionStart)
1416 
1417     return SectionStart
1418 }
1419 
```

```
1420
1421
1422 //Estimate the overall delay. This may be completely off, but in most cases it is a good starting
     point.
1423 //Especially for low SNR values this may help identifying the correct reparse sections.
1424 //Also, if the correlation is high enough, we may skip/speed up some later alignment steps.
1425 //
1426 //the end of the last active ref section. The deg signal is used entirely.
1427 //For the first half, the ref signal starts with the first active section and ranges till the middle
     of the signal,
1428 //the deg section starts at 0 and ends at 75% of the signal length.
1429 //For the second half, the ref signal starts in the middle and ends after the last active section.
     The deg
1430 //section begins at 25% signal length and ends at the end.
1431 //
1432 bool CTempAlignment::EstimateOverallDelaySimpleLimits(int* OverallDelayEstimateInSamples,
     OTA_FLOAT* OverallDelayEstimateReliability,
1433                                                       int* OverallDelayEstimateInSamples1st,
     OTA_FLOAT* OverallDelayEstimateReliability1st,
1434                                                       int* OverallDelayEstimateInSamples2nd,
     OTA_FLOAT* OverallDelayEstimateReliability2nd,
1435                                                       int* Resolution, bool IncreasedResolution)
1436 {
1437     return EstimateOverallDelaySimpleLimits(OverallDelayEstimateInSamples,
     OverallDelayEstimateReliability, OverallDelayEstimateInSamples1st,
     OverallDelayEstimateReliability1st,
1438                                             OverallDelayEstimateInSamples2nd,
     OverallDelayEstimateReliability2nd, Resolution,
     IncreasedResolution, 0, 0)
1439 }
1440
1441 bool CTempAlignment::EstimateOverallDelaySimpleLimits(int* OverallDelayEstimateInSamples,
     OTA_FLOAT* OverallDelayEstimateReliability,
1442                                                       int* OverallDelayEstimateInSamples1st,
     OTA_FLOAT* OverallDelayEstimateReliability1st,
1443                                                       int* OverallDelayEstimateInSamples2nd,
     OTA_FLOAT* OverallDelayEstimateReliability2nd,
1444                                                       int* Resolution, bool IncreasedResolution, int
     InitialEstimate, int
     InitialEstimateResolution)
1445 {
1446     int HistoLen = 10
1447     int HistoShift = 1
1448     int SignalPartDelayIterations = 2
1449
1450     int SearchRange = InitialEstimateResolution
1451
1452     SECTION SecA, SecB
1453     if (IncreasedResolution)
1454     {
1455         mProcessData.Init(2, 1.0/128.)
1456         HistoLen = 1
1457         HistoShift = 0
1458         SignalPartDelayIterations = 1
1459     }
1460     else
1461     {
1462         mProcessData.Init(2, 1.0)
1463         HistoLen = 10
1464         HistoShift = 1
1465         SignalPartDelayIterations = 2
1466
1467         InitialEstimateResolution = 0
1468         InitialEstimate = 0
1469     }
1470
1471     if (mProcessData.mpLogFile)
1472
```

```
1473
1474        mpFeatureList->Create(mppSignals, &mProcessData, OTA_FLTYPE_INITIAL_SEARCH)
1475        int NumFFramesRef = mpFeatureList->GetFVector(0, 0, 0)->mSize
1476        int NumFFramesDeg = mpFeatureList->GetFVector(0, 1, 0)->mSize
1477        OTA_FLOAT* pRef = mpFeatureList->GetFVector(0, 0, 0, 1)->mpVector
1478        OTA_FLOAT* pDeg = mpFeatureList->GetFVector(0, 1, 0, 1)->mpVector
1479        *Resolution = mProcessData.mStepSize
1480
1481        int StartFrameRef = mpActiveFrameDetection->GetStartFrame(0, 0, mProcessData.mStepSize, 0)
1482        int LastFrameRef  = mpActiveFrameDetection->GetLastActiveFrame(0, 0, mProcessData.mStepSize, 0)
1483        int StartFrameDeg = 0
1484        int LastFrameDeg  = NumFFramesDeg
1485
1486        if (InitialEstimateResolution>0)
1487        {
1488            StartFrameDeg = StartFrameRef + (-InitialEstimate - SearchRange) / *Resolution
1489            StartFrameDeg = (((0) > (StartFrameDeg)) ? (0) : (StartFrameDeg))
1490            LastFrameDeg  = LastFrameRef  + (-InitialEstimate + SearchRange) / *Resolution
1491            LastFrameDeg = (((LastFrameDeg) < (NumFFramesDeg-1)) ? (LastFrameDeg) : (NumFFramesDeg-1))
1492
1493            if (LastFrameRef-StartFrameRef>LastFrameDeg-StartFrameDeg)
1494                LastFrameRef = LastFrameDeg - SearchRange / *Resolution
1495        }
1496
1497        int ms50InFrames=MSecondsToFrames(25)
1498
1499        int DegDelay
1500        OTA_FLOAT CorrAll
1501        SecA.Start = StartFrameRef  SecA.End = LastFrameRef
1502        SecB.Start = StartFrameDeg  SecB.End = LastFrameDeg
1503        if (SecA.End-SecA.Start<MSecondsToFrames(500))
1504        {
1505
1506            *OverallDelayEstimateInSamples = 0
1507            *OverallDelayEstimateReliability = 0
1508            *OverallDelayEstimateReliability1st = 0
1509            *OverallDelayEstimateReliability2nd = 0
1510            *OverallDelayEstimateInSamples1st = 0
1511            *OverallDelayEstimateInSamples2nd = 0
1512            return false
1513        }
1514
1515
1516        DegDelay = FindSectionAInSectionB(&SecA, &SecB, pRef, NumFFramesRef, pDeg, NumFFramesDeg,
    &CorrAll, HistoLen, HistoShift)
1517        DegDelay = DegDelay - StartFrameRef + StartFrameDeg
1518
1519        int DelayTemp
1520        OTA_FLOAT CorrTemp
1521
1522        int SecBStartForIteration[5]
1523        int EndOffset = LastFrameRef-LastFrameDeg
1524
1525        int Delay1st = 0
1526        OTA_FLOAT Corr1st = 0
1527
1528        DelayTemp = 0
1529        CorrTemp = 0
1530
1531        SecBStartForIteration[0] = (((0) > (StartFrameDeg - ms50InFrames)) ? (0) : (StartFrameDeg -
    ms50InFrames))
1532        SecBStartForIteration[1] = (((0) > (StartFrameRef + DegDelay - ms50InFrames)) ? (0) :
    (StartFrameRef + DegDelay - ms50InFrames))
1533
1534        SecA.End = NumFFramesRef/2
1535        if (InitialEstimateResolution>0)
1536        {
1537            SecB.End = (int)(SecA.End - EndOffset)
```

```
1538      }
1539      else
1540      {
1541          SecB.End = (int)(NumFFramesDeg*0.75)
1542      }
1543
1544      for(int iteration = 0  iteration < SignalPartDelayIterations  iteration++)
1545      {
1546          SecB.Start = SecBStartForIteration[iteration]
1547          if (SecB.Start<SecB.End && SecA.Start<SecA.End)
1548          {
1549              if ((OTA_FLOAT)(SecA.End-SecA.Start) / (OTA_FLOAT)(SecB.End-SecB.Start)>0.99)
1550                  SecA.End = SecA.Start + (int)(0.99*(OTA_FLOAT)(SecB.End-SecB.Start))
1551              if (SecA.End-SecA.Start<MSecondsToFrames(500))
1552              {
1553
1554              }
1555              else
1556              {
1557
1558                  DelayTemp = FindSectionAInSectionB(&SecA, &SecB, pRef, NumFFramesRef, pDeg,
      NumFFramesDeg, &CorrTemp, HistoLen)
1559                  DelayTemp += -SecA.Start + SecB.Start
1560
1561                  if(Corr1st < CorrTemp)
1562                  {
1563                      Corr1st = CorrTemp
1564                      Delay1st = DelayTemp
1565                  }
1566              }
1567          }
1568      }
1569
1570      int Delay2nd = 0
1571      OTA_FLOAT Corr2nd = 0
1572
1573      DelayTemp = 0
1574      CorrTemp = 0
1575
1576      if (InitialEstimateResolution>0)
1577      {
1578          int StartOffset = StartFrameRef - StartFrameDeg
1579          StartFrameRef = (int)(0.5*NumFFramesRef)
1580          StartFrameDeg = StartFrameRef - StartOffset
1581          SecB.End = LastFrameDeg
1582
1583      }
1584      else
1585      {
1586          StartFrameRef = (int)(0.5*NumFFramesRef)
1587          StartFrameDeg = (int)(0.25*NumFFramesDeg)
1588          SecB.End = NumFFramesDeg
1589      }
1590
1591      SecA.Start = StartFrameRef
1592      SecA.End = LastFrameRef
1593
1594      SecBStartForIteration[0] = (((0) > (StartFrameDeg)) ? (0) : (StartFrameDeg))
1595      SecBStartForIteration[1] = (((0) > (StartFrameRef + DegDelay - ms50InFrames)) ? (0) :
      (StartFrameRef + DegDelay - ms50InFrames))
1596
1597      for(int iteration = 0  iteration < SignalPartDelayIterations  iteration++)
1598      {
1599          SecB.Start = SecBStartForIteration[iteration]
1600          if (SecB.Start<SecB.End && SecA.Start<SecA.End)
1601          {
1602              if ((OTA_FLOAT)(SecA.End-SecA.Start) / (OTA_FLOAT)(SecB.End-SecB.Start)>0.99)
1603                  SecA.End = SecA.Start + 0.99*(OTA_FLOAT)(SecB.End-SecB.Start)
```

```
1604
1605            if (SecA.End-SecA.Start<MSecondsToFrames(500))
1606            {
1607
1608            }
1609            else
1610            {
1611
1612                DelayTemp = FindSectionAInSectionB(&SecA, &SecB, pRef, NumFFramesRef, pDeg,
    NumFFramesDeg, &CorrTemp, HistoLen)
1613                DelayTemp += -SecA.Start + SecB.Start
1614
1615                if(Corr2nd < CorrTemp)
1616                {
1617                    Corr2nd = CorrTemp
1618                    Delay2nd = DelayTemp
1619                }
1620            }
1621        }
1622    }
1623
1624    int Tolerance = ms50InFrames
1625    int MaxLength = MSecondsToFrames(15000)
1626    if (LastFrameRef>MaxLength && CorrAll<0.75)
1627        Tolerance = MSecondsToFrames(10)
1628
1629    if (abs(DegDelay-Delay1st)>Tolerance)
1630        *OverallDelayEstimateReliability = 0
1631    else if (abs(DegDelay-Delay2nd)>Tolerance)
1632        *OverallDelayEstimateReliability = 0
1633    else
1634        *OverallDelayEstimateReliability = CorrAll
1635
1636    if (CorrAll>0.94)
1637        *OverallDelayEstimateReliability = CorrAll
1638
1639    *OverallDelayEstimateInSamples = FramesToSamples( -DegDelay)
1640
1641    *OverallDelayEstimateReliability1st = Corr1st
1642    *OverallDelayEstimateInSamples1st = FramesToSamples( -Delay1st)
1643
1644    *OverallDelayEstimateReliability2nd = Corr2nd
1645    *OverallDelayEstimateInSamples2nd = FramesToSamples( -Delay2nd)
1646
1647    bool DelayIsReliableandConst=false
1648    const OTA_FLOAT MinCorr = 0.5
1649    if (Delay1st==Delay2nd && Delay1st==DegDelay && CorrAll>MinCorr && Corr1st>MinCorr &&
    Corr2nd>MinCorr)
1650        DelayIsReliableandConst = true,
1651
1652    return DelayIsReliableandConst
1653 }
1654
1655
1656
1657 int CTempAlignment::GetNearestStart(int Signal, int StartSample, int OffsetInSamples, bool
    IgnoreActivityFlags)
1658 {
1659
1660    assert(StartSample>=0)
1661    assert(StartSample+OffsetInSamples>=0)
1662
1663    int NumMacroFramesRef = mpActiveFrameDetection->GetMaxFrames(Signal, 0)
1664    int RefStartFrame
1665
1666    if (!IgnoreActivityFlags)
1667    {
1668        int* pActiveFrameFlagsRef = new int[NumMacroFramesRef]
```

```
1669
1670        NumMacroFramesRef = mpActiveFrameDetection->GetActiveFrameFlags(Signal, 0,
    mProcessData.mStepSize, pActiveFrameFlagsRef, NumMacroFramesRef)
1671
1672        RefStartFrame = (StartSample+OffsetInSamples) / mProcessData.mStepSize
1673        RefStartFrame = (((RefStartFrame) < (NumMacroFramesRef)) ? (RefStartFrame) :
    (NumMacroFramesRef))
1674        if (RefStartFrame==0) RefStartFrame = 1
1675        if (!pActiveFrameFlagsRef[RefStartFrame] )
1676        {
1677            while (RefStartFrame<NumMacroFramesRef && !pActiveFrameFlagsRef[RefStartFrame])
1678                RefStartFrame++
1679
1680            while (RefStartFrame && !pActiveFrameFlagsRef[RefStartFrame])
1681                RefStartFrame--
1682
1683            int LastEnd = RefStartFrame
1684            while (LastEnd && !pActiveFrameFlagsRef[LastEnd])
1685                LastEnd--
1686            RefStartFrame = LastEnd + (RefStartFrame-LastEnd)/2
1687        }
1688        else
1689        {
1690            while (RefStartFrame && pActiveFrameFlagsRef[RefStartFrame])
1691                RefStartFrame--
1692
1693            int LastEnd = RefStartFrame
1694            while (LastEnd && !pActiveFrameFlagsRef[LastEnd])
1695                LastEnd--
1696            RefStartFrame = LastEnd + (RefStartFrame-LastEnd)/2
1697
1698        }
1699
1700        if (RefStartFrame==NumMacroFramesRef)
1701        {
1702            RefStartFrame = (((0) > (StartSample-OffsetInSamples)) ? (0) :
    (StartSample-OffsetInSamples))/mProcessData.mStepSize
1703            if (RefStartFrame>=NumMacroFramesRef)
1704                RefStartFrame = StartSample / mProcessData.mStepSize
1705
1706            if (RefStartFrame>=NumMacroFramesRef)
1707                RefStartFrame = 0
1708        }
1709
1710        delete[] pActiveFrameFlagsRef
1711    }
1712    else
1713    {
1714        RefStartFrame = (((0) > (StartSample-OffsetInSamples)) ? (0) :
    (StartSample-OffsetInSamples))/mProcessData.mStepSize
1715        if (RefStartFrame>=NumMacroFramesRef)
1716            RefStartFrame = 0
1717    }
1718
1719    if (RefStartFrame>1) RefStartFrame-= 2
1720    else RefStartFrame = 0
1721
1722    return mpActiveFrameDetection->GetStartSample(Signal, 0, RefStartFrame*mProcessData.mStepSize)
1723 }
1724
1725 //Do a coarse localisation of reparse points by searching for the next longer active section after
1726 //an initial inactive section.
1727 int CTempAlignment::SearchActiveSegments(REPARSE_POINT* ParsePoints, int MaxParsePoints, int*
    pActiveFrameFlags)
1728 {
1729    int NumParsePointsDetected=0
1730    int i=0
1731
```

```
1732      while (i<mNumMacroFrames && !pActiveFrameFlags[i])  i++
1733      ParsePoints[NumParsePointsDetected].Deg.Start = FramesToSamples(i)
1734
1735      while(i<mNumMacroFrames)
1736      {
1737          i = GetNextPauseStartFrameIndex(pActiveFrameFlags, i, mNumMacroFrames)
1738
1739          ParsePoints[NumParsePointsDetected++].Deg.End = FramesToSamples(i)
1740
1741          while (i<mNumMacroFrames && !pActiveFrameFlags[i])  i++
1742          ParsePoints[NumParsePointsDetected].Deg.Start = FramesToSamples(i)
1743      }
1744
1745      if (!NumParsePointsDetected)
1746      {
1747          NumParsePointsDetected = 1
1748          ParsePoints[0].Deg.End = FramesToSamples(mNumMacroFrames)
1749      }
1750
1751      return NumParsePointsDetected
1752  }
1753
1754  //Do a coarse search for potential endpoints of the reparse points.
1755  void CTempAlignment::SearchInactiveSegments(REPARSE_POINT* ReparsePoints, int MaxReparsePoints, int*
      pActiveFrameFlags, bool WorkOnDegSignal)
1756  {
1757      int MinPauseDurationInFrames = MSecondsToFrames(500)
1758      for (int r=0  r<MaxReparsePoints  r++)
1759      {
1760          if (WorkOnDegSignal)
1761          {
1762              int End = SamplesToFrames(ReparsePoints[r].Deg.Start)
1763              while (End<mNumMacroFrames && !pActiveFrameFlags[End]) End++
1764              End = GetNextPauseStartFrameIndex(pActiveFrameFlags, End, mNumMacroFrames)
1765              ReparsePoints[r].Deg.End = FramesToSamples(End)
1766          }
1767          else
1768          {
1769              int End = SamplesToFrames(ReparsePoints[r].Ref.Start)
1770              while (End<mNumMacroFrames && !pActiveFrameFlags[End]) End++
1771
1772              End = GetNextPauseStartFrameIndex(pActiveFrameFlags, End, mNumMacroFrames)
1773              ReparsePoints[r].Ref.End = FramesToSamples(End)
1774          }
1775      }
1776  }
1777
1778  int CTempAlignment::GetNextPauseStartFrameIndex(int* pVec, int StartIdx, int VecLen)
1779  {
1780      int MinPauseDurationInFrames = MSecondsToFrames(500)
1781      bool EndFound=false
1782      int End = StartIdx
1783
1784      while (!EndFound && End<VecLen)
1785      {
1786          while (End<mNumMacroFrames && pVec[End]) End++
1787          int RequiredEnd=End+MinPauseDurationInFrames
1788          while (End<mNumMacroFrames && !pVec[End] && RequiredEnd) RequiredEnd--
1789          if (!RequiredEnd)
1790              EndFound=true
1791          else
1792              if (End<mNumMacroFrames)
1793                  End += MinPauseDurationInFrames-RequiredEnd
1794      }
1795      return End
1796  }
1797
1798  inline void DeleteReparsePoint(int IndexToDelete, REPARSE_POINT*ReparsePoints, int* Num)
```

```
1799 {
1800     for (int d=IndexToDelete  d<*Num-1  d++)
1801         ReparsePoints[d] = ReparsePoints[d+1]
1802     *Num = *Num-1
1803 }
1804
1805 inline void DuplicateReparsePoint(int IndexToDup, REPARSE_POINT*ReparsePoints, int* Num)
1806 {
1807     for (int d=*Num  d>=IndexToDup  d--)
1808         ReparsePoints[d+1] = ReparsePoints[d]
1809     *Num = *Num+1
1810 }
1811
1812 inline void DiscardReparseSectionDeg(int IndexToIgnore, REPARSE_POINT* ReparsePoints, int*
    NumReparsePoints)
1813 {
1814     int i = IndexToIgnore
1815     while (i+1<(*NumReparsePoints))
1816     {
1817       ReparsePoints[i].Deg.End = ReparsePoints[i+1].Ref.End
1818       ReparsePoints[i].Deg.Start = ReparsePoints[i+1].Deg.Start
1819       i++
1820     }
1821     *NumReparsePoints = *NumReparsePoints-1
1822 }
1823
1824 inline void SplitReparsePointRef(int IndexToSplit, REPARSE_POINT* ReparsePoints, int* Num, int
    SplitPos)
1825 {
1826     DuplicateReparsePoint(IndexToSplit, ReparsePoints, Num)
1827
1828     int SamplesCutOff = ReparsePoints[IndexToSplit].Ref.End-SplitPos
1829     ReparsePoints[IndexToSplit].Ref.End-=SamplesCutOff
1830     ReparsePoints[IndexToSplit+1].Ref.Start = SplitPos
1831     ReparsePoints[IndexToSplit+1].IsFromSplitting = true
1832 }
1833
1834 inline int CTempAlignment::FixInterruptedDegSection(int NumReparsePointsRef, REPARSE_POINT*
    ReparsePointsRef, int NumReparsePointsDeg, REPARSE_POINT* ReparsePointsDeg, int TolerancePlus, int
    ToleranceMinus, int MinLenDiff)
1835 {
1836     bool Repeat=false
1837     bool Modified = false
1838     OTA_FLOAT CorrAll,CorrAll1
1839     CorrAll = 0
1840     CorrAll1 = 0
1841     int DegDelay = 0
1842     bool Split = true
1843     OTA_FLOAT* pRef = mpFeatureList->GetFVector(0, 0, 0, 1)->mpVector
1844     OTA_FLOAT* pDeg = mpFeatureList->GetFVector(0, 1, 0, 1)->mpVector
1845     int NumFFramesRef = mpFeatureList->GetFVector(0, 0, 0)->mSize
1846     int NumFFramesDeg = mpFeatureList->GetFVector(0, 1, 0)->mSize
1847
1848     if (NumReparsePointsDeg>NumReparsePointsRef) Repeat = true
1849     while (Repeat)
1850     {
1851         Repeat = false
1852         int r = 0
1853         for (int d=1  d<NumReparsePointsDeg && r<NumReparsePointsRef  d++, r++)
1854         {
1855             int Diff = ReparsePointsRef[r].Ref.Len() - ReparsePointsDeg[d].Deg.Len() -
    ReparsePointsDeg[d-1].Deg.Len()
1856             int LenDiff = ReparsePointsDeg[d].Deg.Len()-ReparsePointsRef[r].Ref.Len()
1857             if ((LenDiff>MinLenDiff&& Diff>0 && Diff<TolerancePlus) || (Diff<0 &&
    Diff>ToleranceMinus))
1858             {
1859                 int ShortUtt = -1
1860                 if (ReparsePointsDeg[d-1].Deg.Len()<MSecondsToSamples(200))
```

```
                    ShortUtt = d-1
                if (ReparsePointsDeg[d].Deg.Len()<MSecondsToSamples(200))
                    ShortUtt = d
                if (ShortUtt!=-1)
                {
                    CorrAll1 = 1
                    SECTION SecDeg,SecRef
                    SECTION SecA = ReparsePointsDeg[ShortUtt].Deg
                    SecA.Start = SamplesToFrames(SecA.Start)
                    SecA.End = SamplesToFrames(SecA.End)
                    SECTION SecB = ReparsePointsRef[r].Ref
                    SecB.Start = SamplesToFrames(SecB.Start)
                    SecB.End = SamplesToFrames(SecB.End)
                    if ((SecA.End-SecA.Start)>2  && (SecB.End-SecB.Start)>2 && NumFFramesDeg>2 && NumFFramesRef>2 && (SecB.Start<=NumFFramesRef))
                        DegDelay = FindSectionAInSectionB(&SecA, &SecB, pDeg, NumFFramesDeg, pRef, NumFFramesRef, &CorrAll1,1,1)
                    else CorrAll1=0
                    CorrAll = (((CorrAll1) < (1)) ? (CorrAll1) : (1))
                    if (CorrAll<0.6 || abs(DegDelay) > 3)
                        Split = false

                }

                if (Split)
                {
                    SplitReparsePointRef(r, ReparsePointsRef, &NumReparsePointsRef, ReparsePointsDeg[d-1].Deg.Len()+ReparsePointsRef[r].Ref.Start)
                Repeat = true
                Modified = true
                }
                else
                {
                    DiscardReparseSectionDeg(ShortUtt, ReparsePointsDeg, &NumReparsePointsDeg)
                Repeat = true
                }
                break
            }
        }
    }

    return NumReparsePointsRef
}

//Return 1 if a match could be found, -1 if the sections were modified, 0 if nothing could be done
int CTempAlignment::FindMatchingSection(REPARSE_POINT* ReparsePointsRef, int NumReparsePointsRef, REPARSE_POINT* ReparsePointsDeg, int NumReparsePointsDeg, OTA_FLOAT SNRdB, int SectionIndex, int* pRefPointsRemaining, int* pDegPointsRemaining)
{
    int rc=0
    int d, r
    r=SectionIndex

    if (r<NumReparsePointsDeg)
    {
        int LengthRef = ReparsePointsRef[r].Ref.End - ReparsePointsRef[r].Ref.Start
        int LengthDeg = ReparsePointsDeg[r].Deg.End - ReparsePointsDeg[r].Deg.Start
        if (abs(LengthDeg-LengthRef)<MSecondsToSamples(200) || r==NumReparsePointsRef)
        {

            rc = 1
        }
        else if (LengthRef>LengthDeg)
        {
            int ms200 = MSecondsToSamples(200)
            int ms250 = MSecondsToSamples(250)
            int ms275 = MSecondsToSamples(275)
            int ms350 = MSecondsToSamples(350)
```

```
1924
1925            int DistanceForCombination = ms250
1926            if (SNRdB<8.0) DistanceForCombination *=2
1927
1928            int StartDiff = ReparsePointsRef[r].Ref.Start-ReparsePointsDeg[r].Deg.Start
1929            int LengthOfTwo=0
1930            int LengthOfThree=0
1931            if (r<NumReparsePointsDeg-1)
1932                LengthOfTwo = ReparsePointsDeg[r+1].Deg.End - ReparsePointsDeg[r].Deg.Start
1933            if (r<NumReparsePointsDeg-2)
1934                LengthOfThree = ReparsePointsDeg[r+2].Deg.End - ReparsePointsDeg[r].Deg.Start
1935
1936            if (r<NumReparsePointsDeg-1 && (OTA_FLOAT)LengthRef/(OTA_FLOAT)LengthDeg>2.0 &&
    LengthDeg<ms250 &&
1937                    abs(StartDiff) -
    abs(ReparsePointsRef[r].Ref.Start-ReparsePointsDeg[r+1].Deg.Start)>ms200
1938                   )
1939            {
1940
1941                DeleteReparsePoint(r, ReparsePointsDeg, &NumReparsePointsDeg)
1942            }
1943
1944            else if (abs(LengthOfTwo-LengthRef)<DistanceForCombination)
1945            {
1946                if (r<NumReparsePointsDeg-1 && abs(ReparsePointsDeg[r+2].Deg.End -
    ReparsePointsDeg[r+1].Deg.Start-LengthRef)<ms250
1947                    && abs(StartDiff) >
    abs(ReparsePointsRef[r].Ref.Start-ReparsePointsDeg[r+1].Deg.Start))
1948                {
1949
1950                    DeleteReparsePoint(r, ReparsePointsDeg, &NumReparsePointsDeg)
1951                }
1952                else
1953                {
1954
1955                    ReparsePointsDeg[r].Deg.End = ReparsePointsDeg[r+1].Deg.End
1956                    ReparsePointsDeg[r].IsVirtualPoint = true
1957                    DeleteReparsePoint(r+1, ReparsePointsDeg, &NumReparsePointsDeg)
1958                }
1959            }
1960            else if (r<NumReparsePointsDeg-2 && abs(LengthOfThree-LengthRef)<ms200)
1961            {
1962
1963                ReparsePointsDeg[r].Deg.End = ReparsePointsDeg[r+2].Deg.End
1964                ReparsePointsDeg[r].IsVirtualPoint = true
1965                for (d=r+3  d<NumReparsePointsDeg  d++)
1966                    ReparsePointsDeg[d-2] = ReparsePointsDeg[d]
1967                NumReparsePointsDeg-=2
1968            }
1969
1970            else if (r<NumReparsePointsDeg-1 && (OTA_FLOAT)LengthRef/(OTA_FLOAT)LengthDeg>2.0 &&
    LengthDeg<ms200
1971                &&
    abs(ReparsePointsRef[r].Ref.Start-ReparsePointsDeg[r+1].Deg.Start)<MSecondsToSamples
    (1000) )
1972            {
1973
1974                DeleteReparsePoint(r, ReparsePointsDeg, &NumReparsePointsDeg)
1975            }
1976            else if (r<NumReparsePointsDeg-1 && (OTA_FLOAT)LengthRef/(OTA_FLOAT)LengthDeg>2.0
1977                                        && abs(ReparsePointsDeg[r+1].Deg.End -
    ReparsePointsDeg[r+1].Deg.Start-LengthRef)<ms200)
1978            {
1979
1980                DeleteReparsePoint(r, ReparsePointsDeg, &NumReparsePointsDeg)
1981            }
1982            else if (r==NumReparsePointsDeg-1 ||
    ReparsePointsDeg[r+1].Deg.Start-ReparsePointsDeg[r].Deg.End>MSecondsToSamples(1000))
```

```
1983            {
1984
1985                if (1 && ReparsePointsDeg[r].Deg.End>1 &&
    abs(ReparsePointsDeg[r].Deg.Start-ReparsePointsRef[r].Ref.Start) >
    abs(ReparsePointsDeg[r].Deg.End-ReparsePointsRef[r].Ref.End)
1986                    && ( (r>0)? ReparsePointsDeg[r].Deg.Start >
    ReparsePointsDeg[r-1].Deg.End+MSecondsToFrames(50) : 1 ) )
1987                {
1988
1989                    ReparsePointsDeg[r].Deg.Start = ReparsePointsDeg[r].Deg.End- LengthRef
1990                    ReparsePointsDeg[r].Deg.Start = (((ReparsePointsDeg[r].Deg.Start) > (0)) ?
    (ReparsePointsDeg[r].Deg.Start) : (0))
1991
1992                }
1993                else
1994                {
1995
1996                    ReparsePointsDeg[r].Deg.End = ReparsePointsDeg[r].Deg.Start+LengthRef
1997                }
1998            }
1999            else if (0 && r<NumReparsePointsDeg-1 && (OTA_FLOAT)LengthRef/(OTA_FLOAT)LengthDeg>5)
2000            {
2001
2002                DeleteReparsePoint(r, ReparsePointsDeg, &NumReparsePointsDeg)
2003            }
2004            else
2005            {
2006
2007                if (1 && r<NumReparsePointsDeg-1 && r>0
2008                    && ms350 < abs(ReparsePointsRef[r-1].Ref.Start - ReparsePointsDeg[r-1].Deg.Start
    -(ReparsePointsRef[r].Ref.Start - ReparsePointsDeg[r].Deg.Start)) )
2009                {
2010
2011                    DeleteReparsePoint(r, ReparsePointsDeg, &NumReparsePointsDeg)
2012                }
2013                else
2014                {
2015                    if (abs(LengthOfTwo-LengthRef)<DistanceForCombination*1.5)
2016                    {
2017
2018                        int a = (LengthRef - LengthOfTwo) / 2
2019                        ReparsePointsDeg[r].Deg.Start = (((0) > (ReparsePointsDeg[r].Deg.Start+a)) ?
    (0) : (ReparsePointsDeg[r].Deg.Start+a))
2020                        ReparsePointsDeg[r].Deg.End   = ReparsePointsDeg[r].Deg.Start + LengthRef
2021                        DeleteReparsePoint(r+1, ReparsePointsDeg, &NumReparsePointsDeg)
2022                    }
2023                    else if (1 && ReparsePointsDeg[r].Deg.Start>1 &&
    abs(ReparsePointsDeg[r].Deg.Start-ReparsePointsRef[r].Ref.Start) >
    abs(ReparsePointsDeg[r].Deg.End-ReparsePointsRef[r].Ref.End) )
2024                    {
2025
2026                        ReparsePointsDeg[r].Deg.Start = ReparsePointsDeg[r].Deg.End- LengthRef
2027                        ReparsePointsDeg[r].Deg.Start = (((ReparsePointsDeg[r].Deg.Start) > (0)) ?
    (ReparsePointsDeg[r].Deg.Start) : (0))
2028
2029                    }
2030                    else
2031                    {
2032
2033                        ReparsePointsDeg[r].Deg.End = ReparsePointsDeg[r].Deg.Start+LengthRef
2034                    }
2035                }
2036            }
2037            rc = -1
2038        }
2039        else
2040
2041        {
```

```
2042            int ms200 = MSecondsToSamples(200)
2043            int ms250 = MSecondsToSamples(250)
2044            int ms350 = MSecondsToSamples(350)
2045            int StartDiff = ReparsePointsRef[r].Ref.Start-ReparsePointsDeg[r].Deg.Start
2046            int LengthOfTwo=0
2047            int LengthOfThree=0
2048            if (r<NumReparsePointsRef-1)
2049                LengthOfTwo = ReparsePointsRef[r+1].Ref.End - ReparsePointsRef[r].Ref.Start
2050            if (r<NumReparsePointsRef-2)
2051                LengthOfTwo = ReparsePointsRef[r+2].Ref.End - ReparsePointsRef[r].Ref.Start
2052
2053            if (abs(LengthOfTwo-LengthDeg)<ms250)
2054            {
2055                if (r<NumReparsePointsRef-1 && abs(ReparsePointsRef[r+2].Ref.End -
    ReparsePointsRef[r+1].Ref.Start-LengthDeg)<ms250
2056                     && abs(StartDiff) >
    abs(ReparsePointsDeg[r].Deg.Start-ReparsePointsRef[r+1].Ref.Start))
2057                {
2058
2059                    DeleteReparsePoint(r, ReparsePointsRef, &NumReparsePointsRef)
2060                }
2061                else
2062                {
2063
2064                    ReparsePointsRef[r].Ref.End = ReparsePointsRef[r+1].Ref.End
2065                    ReparsePointsRef[r].IsVirtualPoint = true
2066                    DeleteReparsePoint(r+1, ReparsePointsRef, &NumReparsePointsRef)
2067                }
2068            }
2069            else if (r<NumReparsePointsRef-2 && abs(LengthOfThree-LengthDeg)<ms200)
2070            {
2071
2072                ReparsePointsRef[r].Ref.End = ReparsePointsRef[r+2].Ref.End
2073                ReparsePointsRef[r].IsVirtualPoint = true
2074                for (d=r+2  d<NumReparsePointsDeg  d++)
2075                    ReparsePointsRef[d] = ReparsePointsRef[d+2]
2076                NumReparsePointsRef-=2
2077            }
2078
2079            else
2080            {
2081                int LengthOfTwoRef=0
2082                if (r<NumReparsePointsRef-1)
2083                    LengthOfTwoRef = ReparsePointsRef[r+1].Ref.End - ReparsePointsRef[r].Ref.Start
2084                bool IsGoodFit = (abs(LengthOfTwoRef-LengthDeg)<ms250)
2085                if (NumReparsePointsDeg<NumReparsePointsRef || IsGoodFit)
2086                {
2087
2088                    for (d=NumReparsePointsDeg  d>r  d--)
2089                        ReparsePointsDeg[d] = ReparsePointsDeg[d-1]
2090                    ReparsePointsDeg[r+1].IsVirtualPoint = true
2091                    int PauselLenRef = ReparsePointsRef[r+1].Ref.Start-ReparsePointsRef[r].Ref.End
2092                    ReparsePointsDeg[r].Deg.End = ReparsePointsDeg[r].Deg.Start+LengthRef
2093                    ReparsePointsDeg[r+1].Deg.Start = ReparsePointsDeg[r].Deg.End+PauselLenRef
2094                    if (ReparsePointsDeg[r+1].Deg.Start>=ReparsePointsDeg[r+1].Deg.End)
2095                        ReparsePointsDeg[r+1].Deg.Start = ReparsePointsDeg[r].Deg.End +1
2096                    NumReparsePointsDeg++
2097                }
2098                else
2099                {
2100
2101                    ReparsePointsDeg[r].Deg.End = ReparsePointsDeg[r].Deg.Start+LengthRef
2102                }
2103            }
2104            rc = -1
2105        }
2106    }
2107    else
```

```
    {
        int LengthRef = ReparsePointsRef[r].Ref.End - ReparsePointsRef[r].Ref.Start

        if (LengthRef<MSecondsToSamples(80))
        {

            DeleteReparsePoint(r, ReparsePointsRef, &NumReparsePointsRef)
            rc = -1
        }
        else if (LengthRef<MSecondsToSamples(500) && r>0)
        {

            int AddLen = ReparsePointsRef[r].Ref.End - ReparsePointsRef[r-1].Ref.End
            ReparsePointsRef[r].Ref.End = ReparsePointsRef[r-1].Ref.End
            ReparsePointsDeg[r-1].Ref.End += AddLen
            NumReparsePointsRef--
            rc = -1
        }
        else
        {

            int Offset = 0
            if (r) Offset = ReparsePointsDeg[r-1].Deg.Start - ReparsePointsRef[r-1].Ref.Start
            ReparsePointsDeg[r].Deg.Start = ReparsePointsRef[r].Ref.Start + Offset
            ReparsePointsDeg[r].Deg.End = ReparsePointsRef[r].Ref.End + Offset
            ReparsePointsDeg[r].Reliability = -1
            NumReparsePointsDeg++
            rc = -1
        }
    }

    //Combine any sections which may be overlapping now
    while (r<NumReparsePointsDeg-1 && ReparsePointsDeg[r+1].Deg.Start<ReparsePointsDeg[r].Deg.End)
    {
        ReparsePointsDeg[r].Deg.End = ReparsePointsDeg[r+1].Deg.End
        DeleteReparsePoint(r+1, ReparsePointsDeg, &NumReparsePointsDeg)
    }

    while (r<NumReparsePointsRef-1 && ReparsePointsRef[r+1].Ref.Start<ReparsePointsRef[r].Ref.End)
    {
        ReparsePointsRef[r].Ref.End = ReparsePointsRef[r+1].Ref.End
        DeleteReparsePoint(r+1, ReparsePointsRef, &NumReparsePointsRef)
    }

    *pDegPointsRemaining = NumReparsePointsDeg
    *pRefPointsRemaining = NumReparsePointsRef
    return rc
}

void CTempAlignment::FindMatchingSectionAInBWithCorrelation(REPARSE_POINT* ReparsePointsA, int
NumReparsePointsA, CFeatureVector* pVecA, REPARSE_POINT* ReparsePointsB, int NumReparsePointsB,
CFeatureVector* pVecB, int r, int* pDelay, OTA_FLOAT* pReliability)
{
    int Delay
    OTA_FLOAT Reliability
    SECTION SectionA, SectionB


    int ALen = ReparsePointsA[r].Ref.Len()
    SectionA.Start = (int)(SamplesToFrames(ReparsePointsA[r].Ref.Start+0.05*ALen))
    SectionA.End   = (int)(SamplesToFrames(ReparsePointsA[r].Ref.End-0.05*ALen))
    ALen = FramesToSamples(SectionA.Len())

    SectionB.Start = (((0) > ((int)SamplesToFrames(ReparsePointsB[r].Deg.Start - 0.1*ALen))) ? (0) :
((int)SamplesToFrames(ReparsePointsB[r].Deg.Start - 0.1*ALen)))
    SectionB.End = SamplesToFrames(ReparsePointsB[r].Deg.End)

    if(SectionA.Len() > 0 && SectionB.Len() > 0)
```

```
2173     {
2174
2175         int MaxInactivity = (int)SamplesToFrames(0.2*ALen)
2176         int InactivityCount=0
2177         while ( SectionB.End<pVecB->mSize && InactivityCount<MaxInactivity)
2178         {
2179             while (SectionB.End<pVecB->mSize && pVecB->mpVector[SectionB.End]>500) SectionB.End++
2180
2181             InactivityCount=0
2182             while (SectionB.End<pVecB->mSize && pVecB->mpVector[SectionB.End]<=500 &&
     InactivityCount++<MaxInactivity) SectionB.End++
2183         }
2184
2185         //If the gap between this and the next active B section is less than 700ms,
2186         //A check is made whether the next deg section shall be included in the search as well.
2187
2188         if (r<NumReparsePointsB-1 &&
     ReparsePointsB[r+1].Deg.Start-ReparsePointsB[r].Deg.End<MSecondsToSamples(700))
2189         {
2190             int NextStop = SamplesToFrames(ReparsePointsB[r+1].Deg.End)
2191             if (NextStop-SectionB.Start > SectionA.End-SectionA.Start)
2192                 SectionB.End = (((SectionB.End) < (SamplesToFrames(ReparsePointsB[r+1].Deg.End))) ?
     (SectionB.End) : (SamplesToFrames(ReparsePointsB[r+1].Deg.End)))
2193             if (SectionB.End>pVecB->mSize)  SectionB.End = pVecB->mSize
2194         }
2195         else SectionB.End = (((SectionB.End) <
     (SamplesToFrames(ReparsePointsB[r].Deg.End)+MSecondsToFrames(500))) ? (SectionB.End) :
     (SamplesToFrames(ReparsePointsB[r].Deg.End)+MSecondsToFrames(500)))
2196
2197         int AFrames = pVecA->mSize
2198
2199         if (SectionA.End-SectionA.Start>1 && AFrames / (SectionA.End-SectionA.Start)<15 &&
     SectionB.End-SectionB.Start>32 &&
2200             SectionB.End>SectionB.Start && SectionA.End>SectionA.Start)
2201         {
2202             //If the ref section is longer than the deg section, then extend ref and reverse the
     search.
2203             if (SectionB.End-SectionB.Start<SectionA.End-SectionA.Start)
2204             {
2205
2206                 SectionB.End = SamplesToFrames(ReparsePointsB[r].Deg.End)
2207                 SectionA.Start = SamplesToFrames(ReparsePointsA[r].Ref.Start)
2208                 SectionA.End   = SamplesToFrames(ReparsePointsA[r].Ref.End)
2209                 Delay = SectionB.Start - SectionA.Start
2210                 Delay = FindSectionAInSectionB(&SectionB, &SectionA, pVecB, pVecA, &Reliability, 10)
2211                 Delay = Delay -SectionB.Start +SectionA.Start
2212                 Delay = -Delay
2213             }
2214             else
2215             {
2216
2217                 Delay = SectionA.Start - SectionB.Start
2218                 Delay = FindSectionAInSectionB(&SectionA, &SectionB, pVecA, pVecB, &Reliability, 10)
2219                 Delay = Delay -SectionA.Start +SectionB.Start
2220             }
2221         }
2222         else
2223         {
2224             Reliability=0
2225             Delay = 0
2226         }
2227     }
2228     else
2229     {
2230
2231         Reliability=0
2232         Delay = 0
2233     }
```

```
2234      *pReliability = Reliability
2235      *pDelay = Delay
2236  }
2237
2238  REPARSE_POINT CTempAlignment::AllocateSectionWithCorrelationBasedOnRefInfo(int Offset, int Len, int
      r, REPARSE_POINT* ReparsePointsRef, int NumReparsePointsRef, CFeatureVector* pVecRefLin,
      REPARSE_POINT* ReparsePointsDeg, int NumReparsePointsDeg, CFeatureVector* pVecDegLin, OTA_FLOAT
      SNRdB, OTA_FLOAT NoiseLevel, bool* pSectionAllocated, bool* pSectionAccepted)
2239  {
2240      REPARSE_POINT Corr1ReparsePointDeg
2241      OTA_FLOAT     Corr1Reliability=-1
2242      int           Corr1Delay = 0
2243      bool          Corr1SectionAllocated=false
2244      bool          Corr1SectionAccepted=false
2245
2246      CFeatureVector* pRef = pVecRefLin
2247      CFeatureVector* pDeg = pVecDegLin
2248      int VecLenRefSamples = FramesToSamples(pRef->mSize)
2249      int VecLenDegSamples = FramesToSamples(pDeg->mSize)
2250
2251      Corr1ReparsePointDeg.DelayInSamples = 0
2252      Corr1ReparsePointDeg.Reliability = -1
2253
2254      SECTION SecRef = ReparsePointsRef[r].Ref
2255      SECTION SecDeg
2256      if (r<NumReparsePointsRef)
2257      {
2258
2259
2260          Corr1ReparsePointDeg.Ref = ReparsePointsRef[r].Ref
2261          if (r<NumReparsePointsDeg)
2262          {
2263              Corr1ReparsePointDeg = ReparsePointsDeg[r]
2264          }
2265          else
2266          {
2267              Corr1ReparsePointDeg.Deg.Start = 0
2268              if (r)  Corr1ReparsePointDeg.Deg.Start = ReparsePointsDeg[r-1].Deg.End
2269              Corr1ReparsePointDeg.Deg.End = VecLenDegSamples
2270          }
2271
2272          SecRef.Start += Offset
2273          if (Len>0) SecRef.End = SecRef.Start+Len
2274
2275          SecDeg = SecRef
2276
2277          SecDeg.Start -= mOverallDelayEstimate
2278          SecDeg.Start = ((((((SecDeg.Start) < (0)) ? (SecDeg.Start) : (0))) > (SecDeg.Start -
      MSecondsToSamples(700))) ? ((((SecDeg.Start) < (0)) ? (SecDeg.Start) : (0))) : (SecDeg.Start
      - MSecondsToSamples(700)))
2279
2280          if (r<NumReparsePointsRef-1)
2281          {
2282              int k
2283              SecDeg.End = ReparsePointsRef[r+1].Ref.Start - mOverallDelayEstimate
2284              if (ReparsePointsRef[r+1].Ref.Start - ReparsePointsRef[r].Ref.End<
      MSecondsToSamples(500))
2285                  SecDeg.End += MSecondsToSamples(500)
2286
2287              int End = ReparsePointsRef[r].Ref.End - mOverallDelayEstimate
2288              for (k=r  k<NumReparsePointsDeg && ReparsePointsDeg[k].Deg.Start<End  k++)
2289
2290              if (k>0 && k<NumReparsePointsDeg && SecDeg.End>ReparsePointsDeg[k].Deg.Start)
2291                  SecDeg.End = ReparsePointsDeg[k-1].Deg.End +
      (ReparsePointsDeg[k].Deg.Start-ReparsePointsDeg[k-1].Deg.End)/2
2292          }
2293          else SecDeg.End = VecLenDegSamples
2294
```

```
2295        if (SecDeg.Start<0)
2296        {
2297            SecRef.Start -= SecDeg.Start
2298
2299            if (SecRef.Len() < SamplesToMSeconds(1000))
2300                SecRef.End -= SecDeg.Start
2301            SecDeg.End -= SecDeg.Start
2302            SecDeg.Start = 0
2303        }
2304
2305        int TooLong = SecDeg.End-FramesToSamples(pDeg->mSize)
2306        if (TooLong>0)
2307        {
2308            SecRef.End -= TooLong
2309            SecRef.Start -= TooLong
2310            SecDeg.End += TooLong
2311            SecDeg.Start -= TooLong
2312        }
2313
2314        if (SecRef.Start<0) SecRef.Start = 0
2315        if (SecRef.End>VecLenRefSamples) SecRef.End = VecLenRefSamples
2316        if (SecDeg.Start<0) SecDeg.Start = 0
2317        if (SecDeg.End>VecLenDegSamples) SecDeg.End = VecLenDegSamples
2318
2319        if (r==0) SecDeg.Start = 0
2320        if (r==NumReparsePointsRef-1) SecDeg.End = VecLenDegSamples
2321
2322        if (SecDeg.Len()<SecRef.Len())
2323
2324
2325        int HistoShiftFrames=4
2326        int HistoShiftSamples = FramesToSamples(HistoShiftFrames)
2327
2328        int HistoLen=12
2329        int lenghtIndicator
2330        lenghtIndicator = (SamplesToMSeconds(VecLenRefSamples)/2800)
2331
2332        if (lenghtIndicator>4)  HistoLen = (8 + lenghtIndicator)
2333        if (HistoLen>18) HistoLen=18
2334
2335        int MaxSpaceForHistogram = (VecLenRefSamples-SecRef.Len()-SecRef.Start)/HistoShiftSamples
2336        MaxSpaceForHistogram = (((MaxSpaceForHistogram) <
    ((VecLenDegSamples-SecDeg.Len()-SecDeg.Start)/HistoShiftSamples)) ? (MaxSpaceForHistogram) :
    ((VecLenDegSamples-SecDeg.Len()-SecDeg.Start)/HistoShiftSamples))
2337
2338        if (MaxSpaceForHistogram<HistoLen)
2339        {
2340            int TotalShiftSamples = HistoShiftSamples * HistoLen
2341            if(SecRef.Start>=TotalShiftSamples) SecRef.Start -= TotalShiftSamples
2342            SecRef.End -= TotalShiftSamples
2343            if(SecDeg.Start>=TotalShiftSamples) SecDeg.Start -= TotalShiftSamples
2344            SecDeg.End -= TotalShiftSamples
2345
2346            HistoLen = (((HistoLen) <
    ((VecLenRefSamples-SecRef.Len()-SecRef.Start)/HistoShiftSamples)) ? (HistoLen) :
    ((VecLenRefSamples-SecRef.Len()-SecRef.Start)/HistoShiftSamples))
2347            HistoLen = (((HistoLen) <
    ((VecLenDegSamples-SecDeg.Len()-SecDeg.Start)/HistoShiftSamples)) ? (HistoLen) :
    ((VecLenDegSamples-SecDeg.Len()-SecDeg.Start)/HistoShiftSamples))
2348        }
2349
2350
2351
2352        int DelayOffset = (SecRef.Start-SecDeg.Start)
2353
2354        Corr1Delay = FindDelayStrict((MAT_HANDLE)mProcessData.mpMathlibHandle,
2355
2356                    pRef->mpVector+SamplesToFrames(SecRef.Start), SamplesToFrames(SecRef.Len()),
```

```
                    pDeg->mpVector+SamplesToFrames(SecDeg.Start), SamplesToFrames(SecDeg.Len()),
                    HistoLen, HistoShiftFrames, 0, &Corr1Reliability)
        Corr1Delay = FramesToSamples(Corr1Delay)
        Corr1Delay = -Corr1Delay+DelayOffset

    }

    if (!ReparsePointsRef[r].IsFromSplitting)
    {
        SecDeg.Start = ReparsePointsRef[r].Ref.Start - Corr1Delay
        SecDeg.End = ReparsePointsRef[r].Ref.End - Corr1Delay
    }
    else
    {
        SecRef.Start = ReparsePointsDeg[r].Deg.Start + Corr1Delay
        SecDeg.Start = ReparsePointsDeg[r].Deg.Start
        SecDeg.End = (((ReparsePointsDeg[r].Deg.End) > (ReparsePointsRef[r].Ref.End - Corr1Delay)) ?
(ReparsePointsDeg[r].Deg.End) : (ReparsePointsRef[r].Ref.End - Corr1Delay))
    }

    if (SecRef.Start<0)
    {
        SecRef.Start = 0
    }

    if (SecDeg.Start<0)
    {
        SecDeg.Start = 0
    }
    if (SecDeg.End>FramesToSamples(pDeg->mSize))
    {
        SecDeg.End = FramesToSamples(pDeg->mSize)
    }
    if (SecRef.End>FramesToSamples(pRef->mSize))
    {
        assert(-1)
        SecDeg.End -= SecRef.End - FramesToSamples(pRef->mSize)
    }

    if (SecRef.End>SecRef.Start && SecDeg.End>SecDeg.Start)
    {

        Corr1ReparsePointDeg.Deg = SecDeg
        Corr1ReparsePointDeg.Ref = ReparsePointsRef[r].Ref
        Corr1ReparsePointDeg.Reliability = Corr1Reliability
        Corr1SectionAllocated = true
    }

    bool DegInfoOk=false
    if (r<NumReparsePointsDeg &&
abs(ReparsePointsDeg[r].Deg.Len()-ReparsePointsRef[r].Ref.Len())<MSecondsToSamples(250) )
        DegInfoOk = true

    if (Corr1SectionAllocated )
    {
        int Offset
        if (r<NumReparsePointsDeg && DegInfoOk)
            Offset = Corr1ReparsePointDeg.Deg.Start-ReparsePointsDeg[r].Deg.Start
        else
        {
            if (r)
            {
                int DelayChange = ReparsePointsDeg[r-1].DelayInSamples-Corr1Delay
                int StartWithOldDelay = Corr1ReparsePointDeg.Deg.Start+DelayChange
                int OldPause = StartWithOldDelay - ReparsePointsDeg[r-1].Deg.End
                Offset = (int)(-DelayChange*0.5)

                Offset = MSecondsToSamples(50)
```

```
2423            }
2424            else Offset = 0
2425        }
2426        Corr1ReparsePointDeg.Deg.Start -= Offset
2427
2428        Corr1ReparsePointDeg.Ref.Start -= Offset
2429
2430        if ( (SNRdB<10.0 && Corr1Reliability>=0.5) || Corr1Reliability>0.6)
2431            Corr1SectionAccepted=true
2432    }
2433
2434    Corr1ReparsePointDeg.Reliability = Corr1Reliability
2435    Corr1ReparsePointDeg.DelayInSamples = Corr1Delay
2436
2437    *pSectionAccepted = Corr1SectionAccepted
2438    *pSectionAllocated = Corr1SectionAllocated
2439    return Corr1ReparsePointDeg
2440 }
2441
2442 REPARSE_POINT CTempAlignment::AllocateSectionWithCorrelationBasedOnVADInfo(int r, REPARSE_POINT*
    ReparsePointsRef, int NumReparsePointsRef, CFeatureVector* pVecRef, REPARSE_POINT* ReparsePointsDeg,
    int NumReparsePointsDeg, CFeatureVector* pVecDeg, OTA_FLOAT SNRdB, bool* pSectionAllocated, bool*
    pSectionAccepted)
2443 {
2444     REPARSE_POINT Corr2ReparsePointDeg
2445     OTA_FLOAT     Corr2Reliability=-1
2446     int           Corr2Delay=0
2447     bool          Corr2SectionAllocated=false
2448     bool          Corr2SectionAccepted=false
2449
2450     if (r<NumReparsePointsDeg)
2451     {
2452         Corr2ReparsePointDeg = ReparsePointsDeg[r]
2453         Corr2ReparsePointDeg.DelayInSamples = 0
2454         Corr2ReparsePointDeg.Reliability = -1
2455     }
2456
2457     if (r<NumReparsePointsRef && r<NumReparsePointsDeg)
2458     {
2459
2460
2461         int RefLen = ReparsePointsRef[r].Ref.End-ReparsePointsRef[r].Ref.Start
2462         int DegLen = ReparsePointsDeg[r].Deg.End-ReparsePointsDeg[r].Deg.Start
2463         if (0.9*RefLen < DegLen)
2464         {
2465
2466             FindMatchingSectionAInBWithCorrelation(ReparsePointsRef, NumReparsePointsRef, pVecRef,
    ReparsePointsDeg, NumReparsePointsDeg, pVecDeg, r, &Corr2Delay, &Corr2Reliability)
2467             Corr2Delay = -FramesToSamples(Corr2Delay)
2468
2469         }
2470         else
2471         {
2472
2473             FindMatchingSectionAInBWithCorrelation(ReparsePointsDeg, NumReparsePointsDeg, pVecDeg,
    ReparsePointsRef, NumReparsePointsRef, pVecRef, r, &Corr2Delay, &Corr2Reliability)
2474             Corr2Delay = FramesToSamples(Corr2Delay)
2475             Corr2Delay = Corr2Delay
2476
2477         }
2478
2479         if (ReparsePointsRef[r].IsFromSplitting)
2480         {
2481             ReparsePointsRef[r].Ref.Start = ReparsePointsDeg[r].Deg.Start + Corr2Delay
2482             ReparsePointsRef[r].Ref.End = ReparsePointsDeg[r].Deg.End + Corr2Delay
2483         }
2484
2485         int DegStart = ReparsePointsRef[r].Ref.Start - Corr2Delay
```

```
2486        if (DegStart<0)
2487        {
2488            if (ReparsePointsRef[r].Ref.End - ReparsePointsRef[r].Ref.Start < -DegStart)
2489            {
2490
2491                Corr2ReparsePointDeg.Deg.Start = 0
2492                Corr2ReparsePointDeg.Deg.End   = ReparsePointsRef[r].Ref.End   - Corr2Delay
2493                Corr2ReparsePointDeg.Ref       = ReparsePointsRef[r].Ref
2494                Corr2ReparsePointDeg.Ref.Start += -DegStart
2495                Corr2ReparsePointDeg.Reliability = Corr2Reliability
2496                Corr2SectionAllocated = true
2497            }
2498            else
2499            {
2500                Corr2SectionAllocated = false
2501            }
2502        }
2503        else
2504        {
2505
2506            Corr2ReparsePointDeg.Deg.Start = ReparsePointsRef[r].Ref.Start - Corr2Delay
2507            Corr2ReparsePointDeg.Deg.End   = ReparsePointsRef[r].Ref.End   - Corr2Delay
2508            Corr2ReparsePointDeg.Ref       = ReparsePointsRef[r].Ref
2509            Corr2ReparsePointDeg.Reliability = Corr2Reliability
2510            Corr2SectionAllocated = true
2511        }
2512
2513        int Offset = Corr2ReparsePointDeg.Deg.Start-ReparsePointsDeg[r].Deg.Start
2514
2515        if (r>0 && ReparsePointsRef[r].Ref.Start>ReparsePointsRef[r].Ref.End)
2516
2517        {
2518            Corr2ReparsePointDeg.Deg.Start -= Offset
2519
2520            Corr2ReparsePointDeg.Ref.Start -= Offset
2521        }
2522
2523        if (Corr2SectionAllocated && (SNRdB<10.0 && Corr2Reliability>=0.4) || Corr2Reliability>=0.8)

2524        {
2525            Corr2SectionAccepted = true
2526        }
2527    }
2528    else Corr2Reliability=0
2529
2530    Corr2ReparsePointDeg.Reliability = Corr2Reliability
2531    Corr2ReparsePointDeg.DelayInSamples = Corr2Delay
2532
2533    *pSectionAccepted = Corr2SectionAccepted
2534    *pSectionAllocated = Corr2SectionAllocated
2535    return Corr2ReparsePointDeg
2536 }
2537
2538 void CTempAlignment::CheckSectionLimits(REPARSE_POINT* ReparsePointsDeg, int NumReparsePointsRef,
    int *NumReparsePointsDeg)
2539 {
2540
2541    for (int r=0  r<NumReparsePointsRef  r++)
2542    {
2543        if (ReparsePointsDeg[r].Deg.End>mppSignals[1]->mSignalLength)
2544        {
2545
2546            ReparsePointsDeg[r].Deg.End=mppSignals[1]->mSignalLength
2547        }
2548        if (ReparsePointsDeg[r].Ref.End>mppSignals[0]->mSignalLength)
2549        {
2550
2551            ReparsePointsDeg[r].Ref.End=mppSignals[0]->mSignalLength
```

```
2552         }
2553
2554         if (ReparsePointsDeg[r].Deg.Start<0)
2555         {
2556
2557             ReparsePointsDeg[r].Ref.Start -= ReparsePointsDeg[r].Deg.Start
2558             ReparsePointsDeg[r].Deg.Start = 0
2559         }
2560
2561         if (ReparsePointsDeg[r].Deg.End<ReparsePointsDeg[r].Deg.Start)
2562         {
2563
2564             DeleteReparsePoint(r, ReparsePointsDeg, NumReparsePointsDeg)
2565             NumReparsePointsRef--
2566             continue
2567         }
2568         if (ReparsePointsDeg[r].Ref.Start<0)
2569         {
2570
2571             if (-ReparsePointsDeg[r].Ref.Start<MSecondsToSamples(300))
2572                 ReparsePointsDeg[r].Deg.Start -= ReparsePointsDeg[r].Ref.Start
2573             ReparsePointsDeg[r].Ref.Start = 0
2574         }
2575         if (ReparsePointsDeg[r].Ref.End<ReparsePointsDeg[r].Ref.Start)
2576         {
2577
2578             DeleteReparsePoint(r, ReparsePointsDeg, NumReparsePointsDeg)
2579             NumReparsePointsRef--
2580             continue
2581         }
2582
2583     }
2584 }
2585
2586 void CTempAlignment::AllocateSectionsFromDelayEstimate(REPARSE_POINT* ReparsePointsDeg,
    REPARSE_POINT* ReparsePointsRef, int NumReparsePointsRef, int *NumReparsePointsDeg, int
    DelayEstimate, OTA_FLOAT DelayEstimateReliability)
2587 {
2588     int i
2589
2590
2591     for (i=0  i<NumReparsePointsRef  i++)
2592     {
2593         ReparsePointsDeg[i].Deg.Start = ReparsePointsRef[i].Ref.Start-DelayEstimate
2594         ReparsePointsDeg[i].Deg.End   = ReparsePointsRef[i].Ref.End-DelayEstimate
2595         ReparsePointsDeg[i].Ref = ReparsePointsRef[i].Ref
2596         ReparsePointsDeg[i].Reliability = DelayEstimateReliability
2597         if (ReparsePointsDeg[i].Deg.Start<0)
2598         {
2599             ReparsePointsDeg[i].Ref.Start -= ReparsePointsDeg[i].Deg.Start
2600             ReparsePointsDeg[i].Deg.Start = 0
2601             if (ReparsePointsDeg[i].Ref.Start>=ReparsePointsDeg[i].Ref.End)
2602             {
2603
2604                 DeleteReparsePoint(i, ReparsePointsRef, &NumReparsePointsRef)
2605                 i--
2606             }
2607         }
2608     }
2609     *NumReparsePointsDeg = NumReparsePointsRef
2610 }
2611
2612 //Identify the position of all reparse points.
2613 //Only the deg startpoint is located roughly, the ref startpoint is set later
2614 //during the delay search when we know the delay of each preceeding active section.
2615 //This will not yet fill in the Delay element of the REPARSE_POINT structs.
2616 //Returns the number of found reparse points.
2617 int CTempAlignment::IdentifyReparsePoints(REPARSE_POINT* ReparsePointsDeg, int MaxParsePoints, int*
```

```
    pActiveFrameFlagsDeg, int OverallDelayEstimate, OTA_FLOAT OverallDelayEstimateReliability,
2618                                          int OverallDelayEstimate1st, OTA_FLOAT
    OverallDelayEstimateReliability1st, int
    OverallDelayEstimate2nd, OTA_FLOAT
    OverallDelayEstimateReliability2nd)
2619 {
2620     bool Done = false
2621     int i
2622     int NumReparsePointsDeg=-1
2623     int OriginalNumReparsePointsDeg=-1
2624     bool TakeRefPointsOnly = false
2625     int OriginalDegSectionLen[100]
2626
2627
2628
2629     for (i=0  i<MaxParsePoints  i++)
2630         ReparsePointsDeg[i].Reliability = -1
2631
2632     mProcessData.Init(1, 1.0)
2633
2634     OTA_FLOAT SigLevel, NoiseLevel, NoiseThreshold
2635     mpActiveFrameDetection->GetLevels(1, 0, mProcessData.mStepSize, &NoiseLevel, &SigLevel,
    &NoiseThreshold, 0)
2636     OTA_FLOAT SNRdB = 10*log10(SigLevel / NoiseLevel)
2637
2638
2639
2640
2641     REPARSE_POINT* ReparsePointsRef = new REPARSE_POINT[MaxParsePoints]
2642     int *pActiveFrameFlagsRef = new int[mNumMacroFrames]
2643     mpActiveFrameDetection->GetActiveFrameFlags(0, 0, mProcessData.mStepSize, pActiveFrameFlagsRef,
    mNumMacroFrames)
2644     int NumReparsePointsRef = SearchActiveSegments(ReparsePointsRef, MaxParsePoints,
    pActiveFrameFlagsRef)
2645     for (i=0  i<NumReparsePointsRef  i++)
2646         ReparsePointsRef[i].Ref = ReparsePointsRef[i].Deg
2647
2648     if (OverallDelayEstimateReliability>0.6)
2649
2650     {
2651         AllocateSectionsFromDelayEstimate(ReparsePointsDeg, ReparsePointsRef, NumReparsePointsRef,
    &NumReparsePointsDeg, OverallDelayEstimate, OverallDelayEstimateReliability)
2652         OriginalNumReparsePointsDeg = NumReparsePointsDeg
2653         for (i=0  i<NumReparsePointsDeg  i++)
2654             OriginalDegSectionLen[i] = ReparsePointsDeg[i].Deg.Len()
2655
2656         Done = true
2657
2658     }
2659
2660     //Do a search for active segments in the deg signal. These may be very inaccurate due to the
    poor signal quality.
2661     if (!Done)
2662     {
2663
2664         NumReparsePointsDeg = SearchActiveSegments(ReparsePointsDeg, 100, pActiveFrameFlagsDeg)
2665         for (i=0  i<NumReparsePointsDeg  i++)
2666             ReparsePointsDeg[i].Deg.Start = ReparsePointsDeg[i].Deg.Start
2667
2668         for (i=0  i<NumReparsePointsDeg  i++)
2669             ReparsePointsDeg[i].IsVirtualPoint = false
2670
2671     }
2672
2673     if (!Done)
2674     {
2675         mpFeatureList->Create(mppSignals, &mProcessData, OTA_FLTYPE_INITIAL_SEARCH)
2676
```

```
2677        if (SNRdB<11)
2678        {
2679
2680
2681            int LastStart = ReparsePointsDeg[NumReparsePointsDeg-1].Deg.Start
2682            for (i=0  i<NumReparsePointsDeg  i++)
2683                ReparsePointsDeg[i].Deg.Start = GetNearestStart(1,
    ReparsePointsDeg[i].Deg.Start+MSecondsToFrames(250), 0, false)-MSecondsToFrames(250)
2684
2685            SearchInactiveSegments(ReparsePointsDeg, NumReparsePointsDeg, pActiveFrameFlagsDeg,
    true)
2686
2687            for (i=0  i<NumReparsePointsDeg  i++)
2688            {
2689                while (i<NumReparsePointsDeg-1 &&
    ReparsePointsDeg[i+1].Deg.Start<ReparsePointsDeg[i].Deg.End)
2690                {
2691                    ReparsePointsDeg[i].Deg.End = ReparsePointsDeg[i+1].Deg.End
2692                    DeleteReparsePoint(i+1, ReparsePointsDeg, &NumReparsePointsDeg)
2693                }
2694            }
2695
2696            if (ReparsePointsDeg[NumReparsePointsDeg-1].Deg.End <
    ReparsePointsDeg[NumReparsePointsDeg-1].Deg.Start)
2697                ReparsePointsDeg[NumReparsePointsDeg-1].Deg.Start = LastStart
2698        }
2699
2700        NumReparsePointsRef = FixInterruptedDegSection(NumReparsePointsRef, ReparsePointsRef,
    NumReparsePointsDeg, ReparsePointsDeg, MSecondsToSamples(400), -MSecondsToSamples(50),
    MSecondsToSamples(200))
2701        DumpReparseSections(NumReparsePointsRef, ReparsePointsRef, NumReparsePointsDeg,
    ReparsePointsDeg, 1)
2702
2703        for (i=0  i<NumReparsePointsDeg  i++)
2704        {
2705            ReparsePointsDeg[i].IsVirtualPoint = false
2706            ReparsePointsDeg[i].Ref = ReparsePointsDeg[i].Deg
2707        }
2708        for (i=0  i<NumReparsePointsRef  i++)
2709        {
2710            ReparsePointsRef[i].Deg = ReparsePointsRef[i].Ref
2711        }
2712
2713        OriginalNumReparsePointsDeg = NumReparsePointsDeg
2714        for (i=0  i<NumReparsePointsDeg  i++)
2715            OriginalDegSectionLen[i] = ReparsePointsDeg[i].Deg.Len()
2716
2717        //Assign ref and deg segments to each other
2718
2719
2720        {
2721            int LastPoint=-1
2722            int LoopCount=0
2723            int r=0
2724            OTA_FLOAT Reliability, SectionCorrelation=0
2725            int Delay
2726            CFeatureVector* pVecRefLog = mpFeatureList->GetFVector(0, 0, 0, 1)
2727            CFeatureVector* pVecDegLog = mpFeatureList->GetFVector(0, 1, 0, 1)
2728            CFeatureVector* pVecRefLin = mpFeatureList->GetFVector(0, 0, 0, 0)
2729            CFeatureVector* pVecDegLin = mpFeatureList->GetFVector(0, 1, 0, 0)
2730            for (r=0  r<NumReparsePointsRef && NumReparsePointsDeg>0  r++)
2731            {
2732
2733
2734
2735                //See if we find a nice match based on the VAD info of both signals. If that is the
    case, just
2736                //use this allocation.
```

```
2737                REPARSE_POINT VAD1ReparsePoint
2738                bool          VAD1SectionAccepted=false
2739                if (1 && r<NumReparsePointsDeg)
2740                {
2741                    OTA_FLOAT     VAD1Reliability=-1
2742                    int           VAD1Delay
2743                    const int MaxLenDiffSamples=MSecondsToSamples(120)
2744                    int LengthRef = ReparsePointsRef[r].Ref.End - ReparsePointsRef[r].Ref.Start
2745                    int LengthDeg = ReparsePointsDeg[r].Deg.End - ReparsePointsDeg[r].Deg.Start
2746                    if (abs(LengthDeg-LengthRef)<MaxLenDiffSamples || SNRdB>35.0 &&
    abs(LengthDeg-LengthRef)<4*MaxLenDiffSamples)
2747                    {
2748
2749                        VAD1ReparsePoint.Deg = ReparsePointsDeg[r].Deg
2750                        VAD1ReparsePoint.Ref = ReparsePointsRef[r].Ref
2751                        VAD1SectionAccepted = true
2752
2753                        int DelayOffset = (VAD1ReparsePoint.Ref.Start-VAD1ReparsePoint.Deg.Start)
2754                        SECTION SecA = VAD1ReparsePoint.Ref
2755                        SECTION SecB = VAD1ReparsePoint.Deg
2756                        SecA.Start = SamplesToFrames(VAD1ReparsePoint.Ref.Start)
2757                        SecA.End = SamplesToFrames(VAD1ReparsePoint.Ref.End)
2758                        SecB.Start = SamplesToFrames(VAD1ReparsePoint.Deg.Start)
2759                        SecB.End= SamplesToFrames(VAD1ReparsePoint.Deg.End)
2760
2761                        if (SecA.Len()>4)
2762                        {
2763                            int Unit = SamplesToFrames(MaxLenDiffSamples)
2764                            SecB.Start -= Unit
2765                            SecB.End   += 2*Unit
2766                            SecB.Start = (((0) > (SecB.Start)) ? (0) : (SecB.Start))
2767
2768                            int Correction = 1
2769                            if (SecA.End-SecA.Start>4*Unit+50)
2770                            {
2771                                SecA.Start += 2*Unit
2772                                SecA.End   -= 2*Unit
2773                                Correction *= 2
2774                            }
2775                            VAD1Delay  = FindSectionAInSectionB(&SecA, &SecB, pVecRefLog,
    pVecDegLog, &VAD1Reliability, 1, 1)
2776                            VAD1Delay -= Correction*Unit
2777
2778                        }
2779                        else
2780                        {
2781                            if (abs(SecA.Len()-SecB.Len())<1)
2782                            {
2783                                VAD1Delay = SecB.Start-SecA.Start
2784                                VAD1Reliability = 0.95
2785                            }
2786                            else
2787                            {
2788                                VAD1Delay = SecB.Start-SecA.Start
2789                                VAD1Reliability = 0.7
2790                            }
2791                        }
2792
2793                        VAD1Delay = FramesToSamples(VAD1Delay)
2794                        VAD1Delay = -VAD1Delay+DelayOffset
2795                        VAD1ReparsePoint.Ref.Start = VAD1ReparsePoint.Deg.Start + VAD1Delay
2796                        VAD1ReparsePoint.Ref.End = VAD1ReparsePoint.Deg.End + VAD1Delay
2797                        VAD1ReparsePoint.Reliability = VAD1Reliability
2798
2799                    }
2800                }
2801
2802                //Try finding the ref section in the deg section by using the ref VAD info,
```

```
    correlation and the global delay
2803                //estimate as starting points. This works for most cases where the delay between the
    first and
2804                //the second half of the sequence did not vary significantly.
2805                REPARSE_POINT Corr1ReparsePointDeg
2806                bool          Corr1SectionAllocated=false
2807                bool          Corr1SectionAccepted=false
2808                int           Corr1Delay
2809                OTA_FLOAT     Corr1Reliability=-1
2810                Corr1ReparsePointDeg = AllocateSectionWithCorrelationBasedOnRefInfo(0, 0, r,
    ReparsePointsRef, NumReparsePointsRef, pVecRefLin, ReparsePointsDeg,
    NumReparsePointsDeg, pVecDegLin, SNRdB, NoiseLevel, &Corr1SectionAllocated,
    &Corr1SectionAccepted)
2811                Corr1Reliability = Corr1ReparsePointDeg.Reliability
2812                Corr1Delay = Corr1ReparsePointDeg.DelayInSamples
2813
2814                if (Corr1Reliability<0.65 && ReparsePointsRef[r].Ref.Len()>MSecondsToSamples(500))
2815
2816                {
2817                    REPARSE_POINT Corr11ReparsePointDeg
2818                    bool          Corr11SectionAllocated=false
2819                    bool          Corr11SectionAccepted=false
2820                    int           Corr11Delay
2821                    OTA_FLOAT     Corr11Reliability=-1
2822                    int Offset = (int)(0.25*Corr1ReparsePointDeg.Ref.Len())
2823                    int Len = 0
2824
2825                    if (SNRdB<5)
2826                    {
2827                        int CenterPos=0
2828                        double MaxEnergy=0
2829                        const int RefLen = SamplesToFrames(ReparsePointsRef[r].Ref.Len())
2830                        MaxEnergy =
    matMaxExt(pVecRefLin->mpVector+SamplesToFrames(ReparsePointsRef[r].Ref.Start
    ), RefLen, &CenterPos)
2831                        Len = 0.25 * ReparsePointsRef[r].Ref.Len()
2832                        Offset = FramesToSamples((((0) > (CenterPos-Len/2)) ? (0) :
    (CenterPos-Len/2)))
2833                    }
2834
2835                    Corr11ReparsePointDeg = AllocateSectionWithCorrelationBasedOnRefInfo(Offset,
    Len, r, ReparsePointsRef, NumReparsePointsRef, pVecRefLin, ReparsePointsDeg,
    NumReparsePointsDeg, pVecDegLin, SNRdB, NoiseLevel, &Corr11SectionAllocated,
    &Corr11SectionAccepted)
2836                    Corr11Reliability = Corr11ReparsePointDeg.Reliability
2837                    Corr11Delay = Corr11ReparsePointDeg.DelayInSamples
2838
2839                    if (Corr11Reliability>Corr1Reliability)
2840                    {
2841                        Corr1ReparsePointDeg = Corr11ReparsePointDeg
2842                        Corr1Reliability = Corr11Reliability
2843                        Corr1Delay = Corr11Delay
2844                        Corr1SectionAllocated = Corr11SectionAllocated
2845                        Corr1SectionAccepted = Corr11SectionAccepted
2846                    }
2847                }
2848
2849                //Try finding the ref section in the deg section by using the correlation and the
    VAD info
2850                //as starting points. If this results in a better correlation for the section, then
    take that.
2851                //This method is successful if there are large delay differences between sections,
    but the
2852                //VAD worked reliably on the degraded signal.
2853                REPARSE_POINT Corr2ReparsePointDeg
2854                OTA_FLOAT     Corr2Reliability=-1
2855                int           Corr2Delay
2856                bool          Corr2SectionAllocated=false
```

```
2857                bool          Corr2SectionAccepted=false
2858                Corr2ReparsePointDeg = AllocateSectionWithCorrelationBasedOnVADInfo(r,
    ReparsePointsRef, NumReparsePointsRef, pVecRefLog, ReparsePointsDeg,
    NumReparsePointsDeg, pVecDegLog, SNRdB, &Corr2SectionAllocated,
    &Corr2SectionAccepted)
2859                Corr2Reliability = Corr2ReparsePointDeg.Reliability
2860                Corr2Delay = Corr2ReparsePointDeg.DelayInSamples
2861
2862                bool SectionAllocated=false
2863                {
2864                    int VersionToUse=-1
2865                    if (Corr1SectionAllocated && Corr2SectionAllocated)
2866                    {
2867                        if (Corr1Reliability>=Corr2Reliability) VersionToUse = 1
2868                        else                                    VersionToUse = 2
2869                    }
2870                    else if (Corr1SectionAllocated)             VersionToUse = 1
2871                    else if (Corr2SectionAllocated)             VersionToUse = 2
2872                    else if (Corr1Reliability>=Corr2Reliability) VersionToUse = 1
2873                    else                                         VersionToUse = 2
2874
2875                    switch(VersionToUse)
2876                    {
2877                        case 1:
2878                        {
2879
2880
2881                            if(mProcessData.mpLogFile)
2882                            {
2883                                if(r < NumReparsePointsRef-1)
2884                                    fprintf(mProcessData.mpLogFile, "Reliability %.10f,
    NumReparsePointsRef %d, ReparsePointsDeg[r].Deg.Len() %d,
    ReparsePointsRef[r].Ref.Len() %d,
    ReparsePointsRef[r+1].Ref.Len() %d\n",
2885                                                                    Corr1Reliability,
    NumReparsePointsRef,
    ReparsePointsDeg[r].Deg.Len(),
    ReparsePointsRef[r].Ref.Len(),
    ReparsePointsRef[r+1].Ref.Len())
2886                                else
2887                                    fprintf(mProcessData.mpLogFile, "Reliability %.10f,
    NumReparsePointsRef %d, ReparsePointsDeg[r].Deg.Len() %d,
    ReparsePointsRef[r].Ref.Len() %d\n",
2888                                                                    Corr1Reliability,
    NumReparsePointsRef,
    ReparsePointsDeg[r].Deg.Len(),
    ReparsePointsRef[r].Ref.Len())
2889
2890                            }
2891                            if (r<NumReparsePointsRef-1 && Corr1Reliability > 0.85
2892                                && ReparsePointsDeg[r].Deg.Len() > ReparsePointsRef[r].Ref.Len() +
    ReparsePointsRef[r+1].Ref.Len())
2893                            {
2894
2895                                DuplicateReparsePoint(r, ReparsePointsDeg, &NumReparsePointsDeg)
2896                                ReparsePointsDeg[r].Deg.End = ReparsePointsDeg[r].Deg.Start +
    ReparsePointsRef[r].Ref.Len()
2897                                ReparsePointsDeg[r+1].Deg.Start=ReparsePointsDeg[r].Deg.End +
    ReparsePointsRef[r+1].Ref.Start - ReparsePointsRef[r].Ref.End
2898                            }
2899
2900                            int VLen1 = ReparsePointsDeg[r].Deg.Len() +
    ReparsePointsDeg[r+1].Deg.Len()
2901                            int VLen2 = ReparsePointsRef[r].Ref.Len()
2902
2903                            if (r<NumReparsePointsDeg-1 &&
2904                                ReparsePointsRef[r].Ref.Len() > ReparsePointsDeg[r].Deg.Len() +
    ReparsePointsDeg[r+1].Deg.Len() &&
```

```
    (ReparsePointsDeg[r+1].Deg.Start-ReparsePointsDeg[r].Deg.End)<MSecon
    dsToSamples(500))
2905
2906                            {
2907
2908                                DuplicateReparsePoint(r, ReparsePointsRef, &NumReparsePointsRef)
2909                                ReparsePointsRef[r].Ref.End = ReparsePointsRef[r].Ref.Start +
    ReparsePointsDeg[r].Deg.Len()
2910                                ReparsePointsRef[r+1].Ref.Start=ReparsePointsRef[r].Ref.End + 1
2911                                Corr1ReparsePointDeg.Deg.End = Corr1ReparsePointDeg.Deg.Start +
    ReparsePointsDeg[r].Deg.Len()
2912                                Corr1ReparsePointDeg.Ref.End = Corr1ReparsePointDeg.Ref.Start +
    ReparsePointsDeg[r].Ref.Len()
2913                            }
2914
2915                            Reliability = Corr1Reliability
2916                            Delay = SamplesToFrames(Corr1Delay)
2917                            ReparsePointsDeg[r] = Corr1ReparsePointDeg
2918
2919                            if (r>=NumReparsePointsDeg)
2920                            {
2921                                NumReparsePointsDeg++
2922                            }
2923                            if (Corr1SectionAccepted)
2924                            {
2925                                SectionAllocated = true
2926
2927
2928                            }
2929
2930                            break
2931                        }
2932                        case 2:
2933                        {
2934
2935
2936                            int DegLen = ReparsePointsDeg[r].Deg.Len()
2937                            int NewLen = Corr2ReparsePointDeg.Ref.Len()
2938                            if (DegLen-NewLen > MSecondsToSamples(500))
2939                            {
2940
2941                                if (ReparsePointsDeg[r].Deg.Len() > ReparsePointsRef[r].Ref.Len() +
    ReparsePointsRef[r+1].Ref.Len())
2942
2943                                {
2944
2945                                    DuplicateReparsePoint(r, ReparsePointsDeg, &NumReparsePointsDeg)
2946
2947                                    ReparsePointsDeg[r+1].Deg.Start = Corr2ReparsePointDeg.Deg.End +
    1
2948
2949                                }
2950                            }
2951                            Reliability = Corr2Reliability
2952                            Delay = SamplesToFrames(Corr2Delay)
2953                            ReparsePointsDeg[r] = Corr2ReparsePointDeg
2954                            if (Corr2SectionAccepted)
2955                            {
2956                                SectionAllocated = true
2957
2958                            }
2959
2960                            if (r>=NumReparsePointsDeg)
2961                            {
2962                                NumReparsePointsDeg++
2963                            }
2964
2965                            break
```

```
                        }
                        default:
                        {

                            Reliability = 0
                            Delay = 0
                            SectionAllocated = false
                        }
                    }
                }

                //If all of the above failed we need to guess the alignment from the VAD info only.
                //This is just a last resort!
                //Since this may destroy some of the real reparse points and a different allocation method might be chosen later,
                //we must operate on a copy of the data.

                REPARSE_POINT ReparsePointVAD
                REPARSE_POINT CopyOfReparsePointsDeg[100]
                int CopyOfNumReparsePointsDeg=NumReparsePointsDeg
                bool FoundVADMatch=false

                for (i=0  i<NumReparsePointsDeg  i++)
                    CopyOfReparsePointsDeg[i] = ReparsePointsDeg[i]

                if (!SectionAllocated)
                {

                    int Res
                    int LoopCount=0
                    do
                    {
                        Res = FindMatchingSection(ReparsePointsRef, NumReparsePointsRef, CopyOfReparsePointsDeg, CopyOfNumReparsePointsDeg, SNRdB, r, &NumReparsePointsRef, &CopyOfNumReparsePointsDeg)
                        if (Res==1)
                        {
                            ReparsePointVAD = CopyOfReparsePointsDeg[r]
                            ReparsePointVAD.Ref = ReparsePointsRef[r].Ref
                            ReparsePointVAD.Reliability = Reliability=0

                            int RefStart = SamplesToFrames(ReparsePointVAD.Ref.Start)
                            int DegStart = SamplesToFrames(ReparsePointVAD.Deg.Start)
                            int DegEnd = SamplesToFrames(ReparsePointVAD.Deg.End)
                            int Len = (((DegEnd-DegStart) < (pVecRefLog->mSize-RefStart)) ? (DegEnd-DegStart) : (pVecRefLog->mSize-RefStart))
                            Len = (((Len) < (pVecDegLog->mSize-DegStart)) ? (Len) : (pVecDegLog->mSize-DegStart))
                            if (Len>0)
                                ReparsePointVAD.Reliability = matPearsonCorrelation(pVecDegLog->mpVector+DegStart, pVecRefLog->mpVector+RefStart, Len)
                            else ReparsePointVAD.Reliability = -1
                            FoundVADMatch = true
                        }
                        else ReparsePointVAD.Reliability = -1

                        LoopCount++
                        if (LoopCount>20)
                        {

                            OPTTHROW(CANT_ALIGN_SIGNALS)
                        }
                    } while (Res<0 && r<NumReparsePointsRef)


                }
```

```
3027
3028                if (!SectionAllocated)
3029                {
3030                    bool UseOverallDelay =
    (OverallDelayEstimateReliability>ReparsePointsDeg[r].Reliability &&
    ReparsePointsDeg[r].Reliability>ReparsePointVAD.Reliability) ? true : false
3031
3032
3033
3034
3035                    bool UseVADMatchDelay = FoundVADMatch && !UseOverallDelay &&
    ReparsePointVAD.Reliability>ReparsePointsDeg[r].Reliability ? true : false
3036
3037                    bool EmergencySolution = false
3038                    if (1 && ReparsePointsDeg[r].Reliability < 0.2 && ReparsePointVAD.Reliability <
    0.2 && OverallDelayEstimateReliability < 0.2)
3039                    {
3040                        int ReparseSectionLiesinHalf = -1
3041                        OTA_FLOAT OverallDelayEstimateReliabilityThisHalf
3042                        int OverallDelayEstimateThisHalf
3043
3044                        if (ReparsePointsDeg[r].Deg.End <= mppSignals[1]->mSignalLength/2)
3045                        {
3046                            ReparseSectionLiesinHalf = 0
3047                            OverallDelayEstimateReliabilityThisHalf =
    OverallDelayEstimateReliability1st
3048                            OverallDelayEstimateThisHalf = OverallDelayEstimate1st
3049                        }
3050                        else if(ReparsePointsDeg[r].Deg.Start > mppSignals[1]->mSignalLength/2)
3051                        {
3052                            ReparseSectionLiesinHalf = 1
3053                            OverallDelayEstimateReliabilityThisHalf =
    OverallDelayEstimateReliability2nd
3054                            OverallDelayEstimateThisHalf = OverallDelayEstimate2nd
3055                        }
3056
3057                        if(ReparseSectionLiesinHalf != -1 && OverallDelayEstimateReliabilityThisHalf
    > 0.5)
3058                        {
3059
3060                            ReparsePointsDeg[r].Deg.Start = ReparsePointsRef[r].Ref.Start -
    OverallDelayEstimateThisHalf
3061                            ReparsePointsDeg[r].Deg.End   = ReparsePointsRef[r].Ref.End -
    OverallDelayEstimateThisHalf
3062                            ReparsePointsDeg[r].Ref = ReparsePointsRef[r].Ref
3063                            ReparsePointsDeg[r].Reliability =
    OverallDelayEstimateReliabilityThisHalf
3064                            if (r>=NumReparsePointsDeg) NumReparsePointsDeg++
3065                            SectionAllocated = true
3066                            UseOverallDelay = false
3067                            UseVADMatchDelay = false
3068                            EmergencySolution = true
3069                        }
3070                    }
3071
3072                    //If the section was not allocated so far, choose the best possible approach
    now.
3073                    //We have three possible delays now:
3074                    //1. The overalDelayEstimate
3075                    //2. The delay from the section correlation (this is already used by default)
3076                    //3. The delay from VAD matching
3077
3078                    if(!EmergencySolution)
3079                    {
3080                        if (UseOverallDelay)
3081                        {
3082
3083                            ReparsePointsDeg[r].Deg.Start =
```

```
    ReparsePointsRef[r].Ref.Start-OverallDelayEstimate
                                ReparsePointsDeg[r].Deg.End   =
    ReparsePointsRef[r].Ref.End-OverallDelayEstimate
                                ReparsePointsDeg[r].Ref = ReparsePointsRef[r].Ref
                                ReparsePointsDeg[r].Reliability = OverallDelayEstimateReliability
                                if (r>=NumReparsePointsDeg) NumReparsePointsDeg++
                                SectionAllocated = true
                            }

                            if (UseVADMatchDelay)
                            {


                                for (i=0  i<CopyOfNumReparsePointsDeg  i++)
                                    ReparsePointsDeg[i] = CopyOfReparsePointsDeg[i]
                                NumReparsePointsDeg = CopyOfNumReparsePointsDeg

                                ReparsePointsDeg[r].Deg = ReparsePointVAD.Deg
                                ReparsePointsDeg[r].Ref = ReparsePointVAD.Ref
                                ReparsePointsDeg[r].Reliability = ReparsePointVAD.Reliability
                                if (r>=NumReparsePointsDeg) NumReparsePointsDeg++
                                SectionAllocated = true
                            }

                            if (!UseOverallDelay && !UseVADMatchDelay)
                            {

                                if (r>=NumReparsePointsDeg) NumReparsePointsDeg++
                                SectionAllocated = true
                            }
                        }
                    }

                    if (ReparsePointsDeg[r].Reliability<0.8 && VAD1SectionAccepted)
                    {

                        for (i=0  i<CopyOfNumReparsePointsDeg  i++)
                            ReparsePointsDeg[i] = CopyOfReparsePointsDeg[i]
                        NumReparsePointsDeg = CopyOfNumReparsePointsDeg
                        ReparsePointsDeg[r] = VAD1ReparsePoint
                        if (r>=NumReparsePointsDeg) NumReparsePointsDeg++
                        SectionAllocated = true
                    }

                    if (SectionAllocated)
                    {
                        for (i=r  i<NumReparsePointsDeg-1  i++)
                        {
                            if (ReparsePointsDeg[i].Deg.End>=ReparsePointsDeg[i+1].Deg.Start)
                            {

                                DeleteReparsePoint(i+1, ReparsePointsDeg, &NumReparsePointsDeg)
                                i--
                            }
                        }
                    }

                }
            }

            //Clean up found sections.

            if (NumReparsePointsDeg<=0)
            {

                for (i=0  i<NumReparsePointsRef  i++)
                {
                    ReparsePointsDeg[i].Deg = ReparsePointsRef[i].Ref
```

```
3150                ReparsePointsDeg[i].Ref = ReparsePointsRef[i].Ref
3151            }
3152        }
3153
3154        //Check for proper limits etc.
3155        //In theory the following checks should not be required...
3156        CheckSectionLimits(ReparsePointsDeg, NumReparsePointsRef, &NumReparsePointsDeg)
3157        if (NumReparsePointsRef<=0)
3158        {
3159
3160            ReparsePointsDeg[0].Ref.Start = ReparsePointsDeg[0].Deg.Start = 0
3161            ReparsePointsDeg[0].Ref.End   = ReparsePointsDeg[0].Deg.End   =
    FramesToSamples((((mpFeatureList->GetFVector(0, 0, 0, 0)->mSize) <
    (mpFeatureList->GetFVector(0, 1, 0, 0)->mSize)) ? (mpFeatureList->GetFVector(0, 0, 0,
    0)->mSize) : (mpFeatureList->GetFVector(0, 1, 0, 0)->mSize)))
3162            NumReparsePointsRef = NumReparsePointsDeg = 1
3163        }
3164        else
3165        {
3166
3167            bool LowSectionCorrelation=true
3168            for (i=0  i<NumReparsePointsRef  i++)
3169                if (ReparsePointsDeg[i].Reliability>0.57)
3170                    LowSectionCorrelation = false
3171
3172            bool BadLengthRatio=true
3173            int TotalOriginalLen=0
3174            int TotalFinalLen=0
3175            for (i=0  i<OriginalNumReparsePointsDeg  i++)
3176            {
3177                TotalOriginalLen += OriginalDegSectionLen[i]
3178                TotalFinalLen    += ReparsePointsDeg[i].Deg.Len()
3179            }
3180            OTA_FLOAT Ratio = (OTA_FLOAT)TotalFinalLen / (OTA_FLOAT)TotalOriginalLen+0.1
3181            if (Ratio<4 && Ratio>0.25)
3182                BadLengthRatio = false
3183
3184            bool BadSectionRatio=true
3185            OTA_FLOAT SectionRatio = (OTA_FLOAT)NumReparsePointsDeg/(OTA_FLOAT)MaxParsePoints
3186            if (SectionRatio<2 && SectionRatio>0.5)
3187                BadSectionRatio = false
3188
3189            if (LowSectionCorrelation && BadLengthRatio)
3190            {
3191
3192
3193                if (OverallDelayEstimateReliability>0)
3194                    AllocateSectionsFromDelayEstimate(ReparsePointsDeg, ReparsePointsRef,
    NumReparsePointsRef, &NumReparsePointsDeg, OverallDelayEstimate,
    OverallDelayEstimateReliability)
3195                else
3196                    AllocateSectionsFromDelayEstimate(ReparsePointsDeg, ReparsePointsRef,
    NumReparsePointsRef, &NumReparsePointsDeg, 0, 0)
3197
3198                CheckSectionLimits(ReparsePointsDeg, NumReparsePointsRef, &NumReparsePointsDeg)
3199            }
3200        }
3201
3202        NumReparsePointsDeg = NumReparsePointsRef
3203
3204        Done = true
3205    }
3206
3207    for (int r=0  r<NumReparsePointsDeg  r++)
3208        ReparsePointsDeg[r].DelayInSamples = ReparsePointsDeg[r].Ref.Start -
    ReparsePointsDeg[r].Deg.Start
3209
3210    return NumReparsePointsDeg
```

```
3211 }
3212
3213
3214 //Fill the initial delay vector with the average delay at the current feature frame resolution,
3215 //but only one for each macro frame. Delay changes are placed in the middle between two reparse
     points.
3216 void CTempAlignment::ReparseSections2DelayVector(OTA_FLOAT* ReliabilityPerFrame, int*
     pSearchRangePerMacroFrameLow, int* pSearchRangePerMacroFrameHigh, int DelayResolution)
3217 {
3218     int r, NextFrame
3219     CProcessData IterationData = mProcessData
3220     IterationData.Init(1, 1.0)
3221     int ReparsePointStepSize = IterationData.mStepSize
3222
3223     for (r=0  r<mNumReparsePoints-1  r++)
3224     {
3225         int StartMacroFrame = (mpReparsePoints[r].Deg.Start+ReparsePointStepSize/2) /
     ReparsePointStepSize
3226
3227         int LastMacroFrame  = (mpReparsePoints[r].Deg.End+ReparsePointStepSize/2) /
     ReparsePointStepSize
3228
3229         int StartOfNextMacroFrame = (mpReparsePoints[r+1].Deg.Start+ReparsePointStepSize/2) /
     ReparsePointStepSize
3230
3231         int DelayDiff = mpReparsePoints[r+1].DelayInSamples - mpReparsePoints[r].DelayInSamples

3232         int NewDelayFrame=0
3233         NewDelayFrame = ((mpReparsePoints[r+1].Deg.Start - mpReparsePoints[r].Deg.End + DelayDiff)/2
     + mpReparsePoints[r].Deg.End) / ReparsePointStepSize
3234         NewDelayFrame = (((NewDelayFrame) > (mpReparsePoints[r].Deg.End / ReparsePointStepSize)) ?
     (NewDelayFrame) : (mpReparsePoints[r].Deg.End / ReparsePointStepSize))
3235
3236         if (r==0)
3237         {
3238             int Delay = mpReparsePoints[0].DelayInSamples
3239             OTA_FLOAT Reliability = 0
3240             for (NextFrame=0  NextFrame<StartMacroFrame && NextFrame<mNumMacroFrames  NextFrame++)
3241             {
3242                 mpDelayInSamplesPerFrame[NextFrame] = Delay
3243                 mpReliabilityPerFrame[NextFrame] = ReliabilityPerFrame[NextFrame] = Reliability
3244                 pSearchRangePerMacroFrameHigh[NextFrame] = mpReparsePoints[0].MaxDelayVarInSamples
3245                 pSearchRangePerMacroFrameLow[NextFrame] = mpReparsePoints[0].MinDelayVarInSamples
3246             }
3247         }
3248         int Delay = mpReparsePoints[r].DelayInSamples
3249         OTA_FLOAT Reliability = mpReparsePoints[r].Reliability
3250         for (NextFrame=StartMacroFrame  NextFrame<LastMacroFrame && NextFrame<mNumMacroFrames
     NextFrame++)
3251         {
3252             mpDelayInSamplesPerFrame[NextFrame] = Delay
3253             mpReliabilityPerFrame[NextFrame] = ReliabilityPerFrame[NextFrame] = Reliability
3254             pSearchRangePerMacroFrameHigh[NextFrame] = mpReparsePoints[r].MaxDelayVarInSamples
3255             pSearchRangePerMacroFrameLow[NextFrame] = mpReparsePoints[r].MinDelayVarInSamples
3256         }
3257
3258         if (mpReparsePoints[r].Reliability >
     mProcessData.mP.mTAPara.mCorrForSkippingInitialDelaySearch)
3259         {
3260             for (int j=StartMacroFrame  j<StartMacroFrame+2 && j<LastMacroFrame && j<mNumMacroFrames
      j++)
3261             {
3262                 pSearchRangePerMacroFrameHigh[j] =  4*DelayResolution
3263                 pSearchRangePerMacroFrameLow[j]  = -4*DelayResolution
3264             }
3265         }
3266
3267         Reliability = mpReparsePoints[r].Reliability
```

```
3268        for (  NextFrame<NewDelayFrame && NextFrame<mNumMacroFrames  NextFrame++)
3269        {
3270            mpDelayInSamplesPerFrame[NextFrame] = Delay
3271            mpReliabilityPerFrame[NextFrame] = ReliabilityPerFrame[NextFrame] = Reliability
3272            pSearchRangePerMacroFrameHigh[NextFrame] = mpReparsePoints[r].MaxDelayVarInSamples
3273            pSearchRangePerMacroFrameLow[NextFrame] = mpReparsePoints[r].MinDelayVarInSamples
3274        }
3275
3276        Delay = mpReparsePoints[r+1].DelayInSamples
3277        Reliability = mpReparsePoints[r+1].Reliability
3278        for (  NextFrame<StartOfNextMacroFrame && NextFrame<mNumMacroFrames  NextFrame++)
3279        {
3280            mpDelayInSamplesPerFrame[NextFrame] = Delay
3281            mpReliabilityPerFrame[NextFrame] = ReliabilityPerFrame[NextFrame] = Reliability
3282            pSearchRangePerMacroFrameHigh[NextFrame] = mpReparsePoints[r+1].MaxDelayVarInSamples
3283            pSearchRangePerMacroFrameLow[NextFrame] = mpReparsePoints[r+1].MinDelayVarInSamples
3284        }
3285    }
3286
3287    int LastStartFrame = mNumReparsePoints>1 ? mpReparsePoints[mNumReparsePoints-1].Deg.Start /
    ReparsePointStepSize : 0
3288    LastStartFrame = (((mNumMacroFrames) < ((((0) > (LastStartFrame)) ? (0) : (LastStartFrame)))) ?
    (mNumMacroFrames) : ((((0) > (LastStartFrame)) ? (0) : (LastStartFrame))))
3289    for (NextFrame=LastStartFrame  NextFrame<mNumMacroFrames  NextFrame++)
3290    {
3291        mpDelayInSamplesPerFrame[NextFrame] = mpReparsePoints[mNumReparsePoints-1].DelayInSamples
3292        mpReliabilityPerFrame[NextFrame] = ReliabilityPerFrame[NextFrame] =
    mpReparsePoints[mNumReparsePoints-1].Reliability
3293        pSearchRangePerMacroFrameHigh[NextFrame] =
    mpReparsePoints[mNumReparsePoints-1].MaxDelayVarInSamples
3294        pSearchRangePerMacroFrameLow[NextFrame] =
    mpReparsePoints[mNumReparsePoints-1].MinDelayVarInSamples
3295    }
3296 }
3297
3298 //Get a list of the active frames and make sure that all sections between reparse points are marked
    as inactive.
3299 //This list is operating at the macro frame rate. MARGIN additional frames on either side of active
    segments are
3300 //set to inactive in order to avoid delay decissions based on frames which may show some border
    effects.
3301
3302 void CTempAlignment::SetActiveFrameFlags(int Stepsize, int* pActiveFrameFlags)
3303 {
3304    int i, r
3305    mpActiveFrameDetection->GetActiveFrameFlags(1, 0, Stepsize, pActiveFrameFlags, mNumMacroFrames)
3306    for (i=0  i<mNumMacroFrames && i<mpReparsePoints[0].Deg.Start/Stepsize+0  i++)
3307        pActiveFrameFlags[i] = 0
3308    for (r=1  r<mNumReparsePoints  r++)
3309        for (i=(((0) > (mpReparsePoints[r-1].Deg.End/Stepsize-0)) ? (0) :
    (mpReparsePoints[r-1].Deg.End/Stepsize-0))  i<mNumMacroFrames &&
    i<mpReparsePoints[r].Deg.Start/Stepsize+0  i++)
3310            pActiveFrameFlags[i] = 0
3311    for (i=(((0) > (mpReparsePoints[mNumReparsePoints-1].Deg.End/Stepsize-0)) ? (0) :
    (mpReparsePoints[mNumReparsePoints-1].Deg.End/Stepsize-0))  i<mNumMacroFrames  i++)
3312        pActiveFrameFlags[i] = 0
3313
3314    int SumFlags = matSum(pActiveFrameFlags, mNumMacroFrames)
3315    if (SumFlags==0)
3316        for (r=1  r<mNumReparsePoints  r++)
3317            matbSet(1, pActiveFrameFlags+mpReparsePoints[r].Deg.Start/Stepsize,
    (mpReparsePoints[r].Deg.End-mpReparsePoints[r].Deg.Start)/Stepsize)
3318
3319 }
3320
3321 //Do the fast prealignment method. This replaces:
3322 //- Overall delay estimation
3323 //- Reparse point identification
```

```
3324 //- Initial delay search
3325 //In addition this drasticall limits the searchrange required for the coarse alignment
3326 //and thus speeds up processing dramatically!
3327 bool CTempAlignment::SectionsFromSQPrealignment(void **pPAHandle)
3328 {
3329     bool rc = true
3330     int i, j, r
3331 
3332     assert(((CAudioSignal*)mppSignals[0])->mSampleRate ==
     ((CAudioSignal*)mppSignals[1])->mSampleRate)
3333     assert(((CAudioSignal*)mppSignals[0])->mNumChannels == 1 &&
3334            ((CAudioSignal*)mppSignals[0])->mNumChannels == 1)
3335 
3336     int              pa_sampleRate = (int) ((CAudioSignal*)mppSignals[0])->mSampleRate
3337     TA_SegList const *pa_segList   = NULL
3338 
3339     *pPAHandle = PreAlignment_Init(
3340         mppSignals[0]->GetDataVector(0), mppSignals[0]->mSignalLength,
3341         mppSignals[1]->GetDataVector(0), mppSignals[1]->mSignalLength,
3342         pa_sampleRate, 2, mProcessData.mpLogFile, mProcessData.mpMathlibHandle)
3343     if (*pPAHandle == NULL)
3344         return false
3345 
3346     pa_segList = PreAlignment_GetSegList(*pPAHandle)
3347     if (pa_segList == NULL)
3348     {
3349         PreAlignment_Free(pPAHandle)
3350         return false
3351     }
3352 
3353     OTA_FLOAT maxPauseLen = (OTA_FLOAT)0.1 / sqrt(((((((PreAlignment_GetDegSNR(*pPAHandle) /
     (OTA_FLOAT)40.0) > ((OTA_FLOAT)0.25)) ? (PreAlignment_GetDegSNR(*pPAHandle) / (OTA_FLOAT)40.0) :
     ((OTA_FLOAT)0.25))) < ((OTA_FLOAT)1.0)) ? ((((PreAlignment_GetDegSNR(*pPAHandle) /
     (OTA_FLOAT)40.0) > ((OTA_FLOAT)0.25)) ? (PreAlignment_GetDegSNR(*pPAHandle) / (OTA_FLOAT)40.0) :
     ((OTA_FLOAT)0.25))) : ((OTA_FLOAT)1.0)))
3354 
3355     r = 0
3356     bool  inSpeechSeg    = false
3357     int   consecLen=0, consecPauseLen, consecLenMatched
3358     XFLOAT avgWeightedDelay=0.0, avgWeightedRlblt=0.0
3359     int   IndexOfLongestSegment = -1
3360     for (i = 0  i < (int)pa_segList->size() && r < 100  i++)
3361     {
3362         if (!inSpeechSeg && pa_segList->at(i).segType == TA_SEG_PAUSE)
3363             continue
3364 
3365         if (inSpeechSeg && pa_segList->at(i).segType == TA_SEG_PAUSE)
3366         {
3367             for (j = i, consecPauseLen = 0
3368                  j < (int)pa_segList->size() && pa_segList->at(j).segType == TA_SEG_PAUSE
3369                  consecPauseLen += pa_segList->at(j).segLen, j++)
3370             if (j < (int)pa_segList->size() && consecPauseLen < (int)(pa_sampleRate*maxPauseLen +
     (OTA_FLOAT)0.5))
3371             {
3372                 i = j-1
3373                 continue
3374             }
3375         }
3376 
3377         if (pa_segList->at(i).segType == TA_SEG_PAUSE)
3378         {
3379 
3380             if (pa_segList->at(IndexOfLongestSegment).segType == TA_SEG_MATCHED &&
     pa_segList->at(IndexOfLongestSegment).segLen > (consecLen>>1))
3381             {
3382                 mpReparsePoints[r].DelayInSamples = pa_segList->at(IndexOfLongestSegment).refPos -
     pa_segList->at(IndexOfLongestSegment).degPos
3383                 mpReparsePoints[r].Reliability    =
```

```
    pa_segList->at(IndexOfLongestSegment).reliability
3384            }
3385            else
3386            {
3387                mpReparsePoints[r].DelayInSamples = RINT(avgWeightedDelay / consecLen)
3388                mpReparsePoints[r].Reliability    = avgWeightedRlblt / consecLen
3389            }
3390
3391            mpReparsePoints[r].MinDelayVarInSamples = -mpReparsePoints[r].DelayInSamples +
    mpReparsePoints[r].MinDelayVarInSamples
3392            mpReparsePoints[r].MaxDelayVarInSamples = mpReparsePoints[r].MaxDelayVarInSamples -
    mpReparsePoints[r].DelayInSamples
3393
3394            mpReparsePoints[r].Deg.End = pa_segList->at(i).degPos - 1
3395            mpReparsePoints[r].Ref.End = pa_segList->at(i).refPos - 1
3396
3397            r++
3398            inSpeechSeg = false
3399        }
3400
3401        else if (inSpeechSeg && i+1 < (int)pa_segList->size() &&
    (pa_segList->at(i+1).degPos-pa_segList->at(i+1).refPos) -
    (pa_segList->at(i).degPos-pa_segList->at(i).refPos) > MSecondsToSamples(250))
3402        {
3403            if (pa_segList->at(IndexOfLongestSegment).segType == TA_SEG_MATCHED &&
    pa_segList->at(IndexOfLongestSegment).segLen > (consecLen>>1))
3404            {
3405                mpReparsePoints[r].DelayInSamples = pa_segList->at(IndexOfLongestSegment).refPos -
    pa_segList->at(IndexOfLongestSegment).degPos
3406                mpReparsePoints[r].Reliability    =
    pa_segList->at(IndexOfLongestSegment).reliability
3407            }
3408            else
3409            {
3410                mpReparsePoints[r].DelayInSamples = RINT(avgWeightedDelay / consecLen)
3411                mpReparsePoints[r].Reliability    = avgWeightedRlblt / consecLen
3412            }
3413
3414            mpReparsePoints[r].MinDelayVarInSamples = -mpReparsePoints[r].DelayInSamples +
    mpReparsePoints[r].MinDelayVarInSamples
3415            mpReparsePoints[r].MaxDelayVarInSamples = mpReparsePoints[r].MaxDelayVarInSamples -
    mpReparsePoints[r].DelayInSamples
3416
3417            mpReparsePoints[r].Deg.End = pa_segList->at(i).degPos + pa_segList->at(i).segLen
3418            mpReparsePoints[r].Ref.End = pa_segList->at(i).refPos + pa_segList->at(i).segLen
3419
3420            r++
3421            inSpeechSeg = false
3422        }
3423
3424        else
3425        {
3426            if (!inSpeechSeg)
3427            {
3428                mpReparsePoints[r].Ref.Start = pa_segList->at(i).refPos
3429                mpReparsePoints[r].Deg.Start = pa_segList->at(i).degPos
3430                mpReparsePoints[r].DelayInSamples = pa_segList->at(i).refPos -
    pa_segList->at(i).degPos
3431                mpReparsePoints[r].MinDelayVarInSamples = mpReparsePoints[r].DelayInSamples
3432                mpReparsePoints[r].MaxDelayVarInSamples = mpReparsePoints[r].DelayInSamples
3433                consecLenMatched = consecLen = 0
3434                avgWeightedDelay = avgWeightedRlblt = 0.0f
3435                IndexOfLongestSegment = i
3436                inSpeechSeg = true
3437            }
3438            else if (pa_segList->at(i).segType != TA_SEG_MISSING)
3439            {
3440                int NewDelayStart = pa_segList->at(i).refPos - pa_segList->at(i).degPos
```

```
3441                int NewDelayEnd   = NewDelayStart
3442                mpReparsePoints[r].MinDelayVarInSamples =
    (((mpReparsePoints[r].MinDelayVarInSamples) < ((((NewDelayStart) < (NewDelayEnd)) ?
    (NewDelayStart) : (NewDelayEnd)))) ? (mpReparsePoints[r].MinDelayVarInSamples) :
    ((((NewDelayStart) < (NewDelayEnd)) ? (NewDelayStart) : (NewDelayEnd))))
3443                mpReparsePoints[r].MaxDelayVarInSamples =
    (((mpReparsePoints[r].MaxDelayVarInSamples) > ((((NewDelayStart) > (NewDelayEnd)) ?
    (NewDelayStart) : (NewDelayEnd)))) ? (mpReparsePoints[r].MaxDelayVarInSamples) :
    ((((NewDelayStart) > (NewDelayEnd)) ? (NewDelayStart) : (NewDelayEnd))))
3444            }
3445
3446            if (pa_segList->at(i).segType != TA_SEG_MISSING)
3447            {
3448                consecLen        += pa_segList->at(i).segLen
3449                avgWeightedDelay += pa_segList->at(i).segLen * (pa_segList->at(i).refPos -
    pa_segList->at(i).degPos)
3450                avgWeightedRlblt += pa_segList->at(i).segLen * pa_segList->at(i).reliability
3451                if (pa_segList->at(i).segLen > pa_segList->at(IndexOfLongestSegment).segLen)
3452                    IndexOfLongestSegment = i
3453            }
3454        }
3455    }
3456
3457    if (inSpeechSeg)
3458    {
3459        for (j = i-1  j >= 0 && pa_segList->at(j).segType == TA_SEG_MISSING  j--)
3460
3461        if (pa_segList->at(IndexOfLongestSegment).segType == TA_SEG_MATCHED &&
    pa_segList->at(IndexOfLongestSegment).segLen > (consecLen>>1))
3462        {
3463            mpReparsePoints[r].DelayInSamples = pa_segList->at(IndexOfLongestSegment).refPos -
    pa_segList->at(IndexOfLongestSegment).degPos
3464            mpReparsePoints[r].Reliability    = pa_segList->at(IndexOfLongestSegment).reliability
3465        }
3466        else
3467        {
3468            mpReparsePoints[r].DelayInSamples = RINT(avgWeightedDelay / consecLen)
3469            mpReparsePoints[r].Reliability    = avgWeightedRlblt / consecLen
3470        }
3471
3472        mpReparsePoints[r].MinDelayVarInSamples = -mpReparsePoints[r].DelayInSamples +
    mpReparsePoints[r].MinDelayVarInSamples
3473        mpReparsePoints[r].MaxDelayVarInSamples = mpReparsePoints[r].MaxDelayVarInSamples -
    mpReparsePoints[r].DelayInSamples
3474
3475        mpReparsePoints[r].Deg.End = pa_segList->at(j).degPos + pa_segList->at(j).segLen - 1
3476        mpReparsePoints[r].Deg.End = (((mppSignals[1]->mSignalLength-1) <
    (mpReparsePoints[r].Deg.End)) ? (mppSignals[1]->mSignalLength-1) :
    (mpReparsePoints[r].Deg.End))
3477        mpReparsePoints[r].Ref.End = pa_segList->at(i-1).refPos + pa_segList->at(i-1).segLen - 1
3478
3479        r++
3480    }
3481
3482    if (r <= 0)
3483    {
3484        PreAlignment_Free(pPAHandle)
3485        return false
3486    }
3487    else
3488    {
3489
3490        mNumReparsePoints = r
3491    }
3492
3493    return rc
3494 }
3495
```

```
3496 bool CTempAlignment::ResetSectionData()
3497 {
3498     bool rc = true
3499
3500     mNumReparsePoints = 1
3501     for (int i = 0  i < 100  i++)
3502     {
3503         mpReparsePoints[i].DelayInSamples = 0
3504         mpReparsePoints[i].Deg.Start = 0
3505         mpReparsePoints[i].Ref.Start = 0
3506         mpReparsePoints[i].Deg.End = mppSignals[1]->mSignalLength
3507         mpReparsePoints[i].Ref.End = mppSignals[0]->mSignalLength
3508     }
3509
3510     return rc
3511 }
3512
3513 //Derive the initial delay vector etc. directly from the segment lists created by
    SectionsFromSQPrealignment().
3514 //this is an alternative to ReparseSections2DelayVector(), but only available when
    SectionsFromSQPrealignment()
3515 //was called before!
3516 bool CTempAlignment::SegmentList2DelayVector(void** pPAHandle, void** pPA_vec, float
    DelayVecStepsize, int* pActiveFrameFlags, OTA_FLOAT* ReliabilityPerFrame, int*
    pSearchRangePerMacroFrameLow, int* pSearchRangePerMacroFrameHigh)
3517 {
3518     bool rc = true
3519
3520     int i, j, k, f
3521
3522     if (pPAHandle == NULL || pPA_vec == NULL)
3523         return false
3524
3525     TraversalVecType const *pa_vec = (TraversalVecType const*)(*pPA_vec)
3526     if (pa_vec == NULL)
3527     {
3528         PreAlignment_Free(pPAHandle)
3529         return false
3530     }
3531
3532     i = f = 0
3533     for (  i < (int)pa_vec->size() && pa_vec->at(i).type == MISSING_SPEECH  i++)
3534     if (i >= (int)pa_vec->size())
3535     {
3536         PreAlignment_Free(pPAHandle)
3537         *pPA_vec = NULL
3538         return false
3539     }
3540
3541     int* pSavedActiveFrameFlags = (int*)matMalloc(sizeof(int)*mNumMacroFrames)
3542     matbCopy(pActiveFrameFlags, pSavedActiveFrameFlags, mNumMacroFrames)
3543
3544     int pa_sampleRate = (int) ((CAudioSignal*)mppSignals[0])->mSampleRate
3545     int pa_delayInSamples, curDegPos = 0, firstSpeechPos, lastSpeechPos
3546     firstSpeechPos = pa_vec->at(i).refPos
3547     for (j = 0  j <= (int)pa_vec->size()-1 && (pa_vec->at(j).type == PAUSE || pa_vec->at(j).type ==
    INSERTED_SIG)  j++)
3548         firstSpeechPos = pa_vec->at(j).refPos
3549     lastSpeechPos  = pa_vec->back().refPos
3550     for (j = (int)pa_vec->size()-1  j >= 0 && (pa_vec->at(j).type == PAUSE || pa_vec->at(j).type ==
    INSERTED_SIG)  j--)
3551         lastSpeechPos  = pa_vec->at(j).refPos
3552
3553     //Add frames for extra leading silence in deg for which there is no info in pa_vec.
3554     if (pa_vec->at(i).degPos != 0)
3555     {
3556         pa_delayInSamples = pa_vec->at(i).refPos - pa_vec->at(i).degPos
3557         while (f < mNumMacroFrames && curDegPos < pa_vec->at(i).degPos)
```

```
3558        {
3559            pActiveFrameFlags  [f] = 0
3560            ReliabilityPerFrame[f] = (OTA_FLOAT)0
3561            pSearchRangePerMacroFrameLow [f] = 0
3562            pSearchRangePerMacroFrameHigh[f] = firstSpeechPos - (curDegPos+pa_delayInSamples)
3563            mpDelayInSamplesPerFrame[f] = pa_delayInSamples
3564            curDegPos += mMacroFrameSize
3565            f++
3566        }
3567    }
3568
3569    int firstPauseRefPos = -1, lastPauseRefPos = -1, insertedSigAct = -1, missingRefStart = -1,
    missingRefEnd = -1,
3570        minConsecMinPos = -1, maxConsecMaxPos = -1
3571    for (  f < mNumMacroFrames && i < (int)pa_vec->size()  i++)
3572    {
3573        if (pa_vec->at(i).type == MISSING_SPEECH)
3574        {
3575            for (j = i  j < (int)pa_vec->size() && pa_vec->at(j).type == MISSING_SPEECH  j++)
3576            j--
3577            missingRefStart = pa_vec->at(i).refPos
3578            missingRefEnd   = pa_vec->at(j).refPos + mMacroFrameSize
3579
3580            if (f > 0 && pActiveFrameFlags[f-1] == 1 && i > 0 &&
3581                (pa_vec->at(i-1).type == SEARCHABLE_SPEECH || pa_vec->at(i-1).type == FIXED_SPEECH)
    &&
3582                pa_vec->at(i-1).maxPos + mMacroFrameSize < missingRefEnd)
3583            {
3584                pSearchRangePerMacroFrameHigh[f-1] = (((pSearchRangePerMacroFrameHigh[f-1]) >
    ((missingRefEnd-mMacroFrameSize) - pa_vec->at(i-1).refPos)) ?
    (pSearchRangePerMacroFrameHigh[f-1]) : ((missingRefEnd-mMacroFrameSize) -
    pa_vec->at(i-1).refPos))
3585                ReliabilityPerFrame          [f-1] = (((ReliabilityPerFrame[f-1]) < (0.75f)) ?
    (ReliabilityPerFrame[f-1]) : (0.75f))
3586            }
3587
3588            firstPauseRefPos = lastPauseRefPos = -1
3589            i = j
3590            continue
3591        }
3592
3593        ReliabilityPerFrame[f] = pa_vec->at(i).reliability
3594        pActiveFrameFlags  [f] = pa_vec->at(i).degActivity && pa_vec->at(i).type != PAUSE ? 1 : 0
3595
3596        if (pa_vec->at(i).type == INSERTED_SIG)
3597        {
3598            if (insertedSigAct < 0)
3599            {
3600                minConsecMinPos = lastSpeechPos, maxConsecMaxPos = firstSpeechPos
3601                for (j = i  j > 0                    && pa_vec->at(j).type == INSERTED_SIG  j--)
3602                    minConsecMinPos = (((minConsecMinPos) < (pa_vec->at(j).minPos)) ?
    (minConsecMinPos) : (pa_vec->at(j).minPos))
3603                for (k = i  k < (int)pa_vec->size()-1 && pa_vec->at(k).type == INSERTED_SIG  k++)
3604                    maxConsecMaxPos = (((maxConsecMaxPos) > (pa_vec->at(k).maxPos)) ?
    (maxConsecMaxPos) : (pa_vec->at(k).maxPos))
3605
3606                if (pa_vec->at(j).type != PAUSE && pa_vec->at(k).type != PAUSE)
3607                {
3608
3609                        pActiveFrameFlags[f] = insertedSigAct = 1
3610                }
3611                else
3612                    pActiveFrameFlags[f] = insertedSigAct = 0
3613            }
3614            else
3615                pActiveFrameFlags[f] = insertedSigAct
3616
3617            if (pa_vec->at(i).maxPos < firstSpeechPos || pa_vec->at(i).minPos > lastSpeechPos)
```

```
3618                pActiveFrameFlags[f] = 0
3619        }
3620        else
3621            insertedSigAct = -1
3622
3623        //Set activity to 1 at utterance boundaries after a pause, as the delay may have changed.
3624        if (pa_vec->at(i).type == SEARCHABLE_SPEECH && pActiveFrameFlags[f] == 0)
3625        {
3626            for (j = i  j > 0                      && pa_vec->at(j).type == SEARCHABLE_SPEECH  j--)
3627            for (k = i  k < (int)pa_vec->size()-1 && pa_vec->at(k).type == SEARCHABLE_SPEECH  k++)
3628            if (pa_vec->at(j).type == PAUSE || pa_vec->at(k).type == PAUSE)
3629                pActiveFrameFlags[f] = 1
3630        }
3631
3632        if (pa_vec->at(i).type != PAUSE)
3633        {
3634            firstPauseRefPos = lastPauseRefPos = -1
3635            int minRefPos = pa_vec->at(i).minPos, maxRefPos = pa_vec->at(i).maxPos
3636            if (missingRefStart >= 0 && missingRefEnd >= 0)
3637            {
3638                minRefPos = (((minRefPos) < (missingRefStart)) ? (minRefPos) : (missingRefStart))
3639                ReliabilityPerFrame[f] = (((ReliabilityPerFrame[f]) < (0.75f)) ?
(ReliabilityPerFrame[f]) : (0.75f))
3640            }
3641            else if (insertedSigAct >= 0)
3642            {
3643                minRefPos = (((minRefPos) < (minConsecMinPos)) ? (minRefPos) : (minConsecMinPos))
3644                maxRefPos = (((maxRefPos) > (maxConsecMaxPos)) ? (maxRefPos) : (maxConsecMaxPos))
3645            }
3646            minRefPos = (((minRefPos) < (pa_vec->at(i).refPos)) ? (minRefPos) :
(pa_vec->at(i).refPos))
3647            maxRefPos = (((maxRefPos) > (pa_vec->at(i).refPos)) ? (maxRefPos) :
(pa_vec->at(i).refPos))
3648            pSearchRangePerMacroFrameLow [f] = -pa_vec->at(i).refPos + minRefPos
3649            pSearchRangePerMacroFrameHigh[f] =  maxRefPos - pa_vec->at(i).refPos
3650
3651            if (ReliabilityPerFrame[f]==0 && pSearchRangePerMacroFrameHigh[f]==0)
3652                pSearchRangePerMacroFrameHigh[f] += mProcessData.mStepSize
3653
3654            if (ReliabilityPerFrame[f]==0 && pSearchRangePerMacroFrameLow[f]==0)
3655                pSearchRangePerMacroFrameLow[f] -= mProcessData.mStepSize
3656
3657        }
3658        else
3659        {
3660            if (firstPauseRefPos < 0)
3661            {
3662                firstPauseRefPos = pa_vec->at(i).refPos
3663                for (int j = i  j < (int)pa_vec->size() && pa_vec->at(j).type == PAUSE  j++)
3664                    lastPauseRefPos = pa_vec->at(j).refPos
3665            }
3666            pSearchRangePerMacroFrameLow [f] = -pa_vec->at(i).refPos + firstPauseRefPos
3667            pSearchRangePerMacroFrameHigh[f] = lastPauseRefPos       - pa_vec->at(i).refPos
3668        }
3669
3670        mpDelayInSamplesPerFrame[f] = pa_vec->at(i).refPos - pa_vec->at(i).degPos
3671
3672        f++
3673
3674        missingRefStart = missingRefEnd = -1
3675    }
3676
3677    //Add frames for extra trailing silence in deg for which there is no info in pa_vec.
3678    for (i--  i >= 0 && pa_vec->at(i).type == MISSING_SPEECH  i--)
3679    if (pa_vec->at(i).degPos + mMacroFrameSize <= ((CAudioSignal*)mppSignals[1])->mSignalLength -
mMacroFrameSize)
3680    {
3681        curDegPos         = pa_vec->at(i).degPos + mMacroFrameSize
```

```
3682        pa_delayInSamples = mpDelayInSamplesPerFrame[f-1]
3683        while (f < mNumMacroFrames && curDegPos + mMacroFrameSize <=
    ((CAudioSignal*)mppSignals[1])->mSignalLength)
3684        {
3685            pActiveFrameFlags            [f] = 0
3686            ReliabilityPerFrame          [f] = (OTA_FLOAT)0
3687            pSearchRangePerMacroFrameLow [f] = -(curDegPos+pa_delayInSamples) + lastSpeechPos
3688            pSearchRangePerMacroFrameHigh[f] = 0
3689            mpDelayInSamplesPerFrame     [f] = pa_delayInSamples
3690            curDegPos += mMacroFrameSize
3691            f++
3692        }
3693    }
3694
3695    for (--f, i = f+1  i < mNumMacroFrames  i++)
3696    {
3697        pActiveFrameFlags            [i] = pActiveFrameFlags            [f]
3698        ReliabilityPerFrame          [i] = (OTA_FLOAT)0
3699        pSearchRangePerMacroFrameLow [i] = pSearchRangePerMacroFrameLow [f]
3700        pSearchRangePerMacroFrameHigh[i] = pSearchRangePerMacroFrameHigh[f]
3701        mpDelayInSamplesPerFrame     [i] = mpDelayInSamplesPerFrame     [f]
3702    }
3703
3704    int FPAResolution = ceil((OTA_FLOAT)mProcessData.mSamplerate / 8000.0)
3705    for (int i=0  i<mNumMacroFrames  i++)
3706    {
3707        pSearchRangePerMacroFrameLow [i] = (((pSearchRangePerMacroFrameLow [i]) < (-FPAResolution))
    ? (pSearchRangePerMacroFrameLow [i]) : (-FPAResolution))
3708        pSearchRangePerMacroFrameHigh[i] = (((pSearchRangePerMacroFrameHigh[i]) > (+FPAResolution))
    ? (pSearchRangePerMacroFrameHigh[i]) : (+FPAResolution))
3709    }
3710
3711    matbCopy(pSavedActiveFrameFlags, pActiveFrameFlags, mNumMacroFrames)
3712    matFree(pSavedActiveFrameFlags)
3713
3714    return rc
3715 }
3716
3717 class TimePos
3718 {
3719 public:
3720     TimePos(int size){privPos=new(OTA_FLOAT[size]) }
3721     ~TimePos(){delete[] privPos }
3722     OTA_FLOAT* get(){return privPos }
3723 public:
3724     OTA_FLOAT *privPos
3725 }
3726 OTA_FLOAT CTempAlignment::DetectTAtimeDrift(void const *pPA_vec, OTA_FLOAT frameStepInSec, int
    frameStepInSamples)
3727 {
3728
3729     if (pPA_vec == NULL)
3730         return (OTA_FLOAT)-1.0
3731
3732     TraversalVecType const *pa_vec = (TraversalVecType const*)pPA_vec
3733
3734     int vecLen = (int)pa_vec->size()
3735     if (vecLen <= 0 || frameStepInSec <= 0.0f || frameStepInSamples <= 0)
3736         return (OTA_FLOAT)-1.0
3737
3738     int   const DTD_MIN_NUM_FRAMES     = (((RINT((OTA_FLOAT)1.5 / frameStepInSec)) > (50)) ?
    (RINT((OTA_FLOAT)1.5 / frameStepInSec)) : (50))
3739     OTA_FLOAT const DTD_MIN_TIMEDRIFT      = (OTA_FLOAT)0.05
3740     OTA_FLOAT const DTD_MAX_RESIDUAL_SQERR = (OTA_FLOAT)1.5
3741
3742     OTA_FLOAT timeDrift = 0.0
3743
3744     TimePos timePosRef(vecLen),timePosDeg(vecLen)
```

```
3745
3746        bool awaitingNewSeg = true
3747        int i, cnt, refOffset = 0, degOffset = 0, end = 0
3748
3749        for (i = 1, cnt = 0  i < vecLen  i++)
3750        {
3751            if (pa_vec->at(i).type == MISSING_SPEECH || pa_vec->at(i).type == INSERTED_SIG ||
3752                pa_vec->at(i).type == PAUSE || !pa_vec->at(i).degActivity)
3753            {
3754                if (!awaitingNewSeg)
3755                    end = i-1
3756
3757                awaitingNewSeg = true
3758                continue
3759            }
3760
3761            if (awaitingNewSeg)
3762            {
3763                refOffset += pa_vec->at(i-1).refPos - pa_vec->at(end).refPos
3764                degOffset += pa_vec->at(i-1).degPos - pa_vec->at(end).degPos
3765                end = i
3766                awaitingNewSeg = false
3767            }
3768            if (!awaitingNewSeg)
3769            {
3770                timePosRef.get()[cnt] = (pa_vec->at(i).refPos - refOffset) /
    (OTA_FLOAT)(frameStepInSamples)
3771                timePosDeg.get()[cnt] = (pa_vec->at(i).degPos - degOffset) /
    (OTA_FLOAT)(frameStepInSamples)
3772                cnt++
3773            }
3774        }
3775
3776        if (cnt < DTD_MIN_NUM_FRAMES)
3777            return (OTA_FLOAT)0.0
3778
3779        OTA_FLOAT covariance = 0.0, variance = 0.0
3780        OTA_FLOAT meanDeg, meanRef, residualSQerr
3781        meanRef = matMean(timePosRef.get(), cnt)
3782        meanDeg = matMean(timePosDeg.get(), cnt)
3783        for (int i = 0  i < cnt  i++)
3784        {
3785            covariance += (timePosRef.get()[i] - meanRef) * (timePosDeg.get()[i] - meanDeg)
3786            variance   += pow(timePosRef.get()[i] - meanRef, 2)
3787        }
3788        covariance /= (OTA_FLOAT)cnt
3789        variance   /= (OTA_FLOAT)cnt
3790        OTA_FLOAT slopeDiff = (covariance+1e-10f)/(variance+1e-10f)
3791
3792        matbMpy1(slopeDiff, timePosRef.get(), cnt)
3793        meanRef = matMean(timePosRef.get(), cnt)
3794
3795        matbAdd1(meanDeg-meanRef, timePosRef.get(), cnt)
3796
3797        matbSub3(timePosRef.get(), timePosDeg.get(), timePosRef.get(), cnt)
3798        matbAbs1(timePosRef.get(),                                      cnt)
3799        residualSQerr = 0.0
3800        for (int i = 0  i < cnt  i++)
3801            residualSQerr += timePosRef.get()[i] * timePosRef.get()[i]
3802        residualSQerr /= (OTA_FLOAT)cnt
3803
3804        OTA_FLOAT slopeFac, errFac
3805        slopeFac = (fabs(slopeDiff - 1) - DTD_MIN_TIMEDRIFT) / DTD_MIN_TIMEDRIFT
3806        errFac   = sqrt((((1.0f - (residualSQerr / DTD_MAX_RESIDUAL_SQERR)) > (0.0f)) ? (1.0f -
    (residualSQerr / DTD_MAX_RESIDUAL_SQERR)) : (0.0f)))
3807        slopeFac = ((((((slopeFac) > (0.0f)) ? (slopeFac) : (0.0f))) < (1.0f)) ? ((((slopeFac) > (0.0f))
    ? (slopeFac) : (0.0f))) : (1.0f))
3808        errFac   = ((((((errFac) > (0.0f)) ? (errFac) : (0.0f))) < (1.0f)) ? ((((errFac) > (0.0f)) ?
```

```
    (errFac) : (0.0f))) : (1.0f))
3809
3810        return ((((((slopeFac*errFac) > ((OTA_FLOAT)0.0)) ? (slopeFac*errFac) : ((OTA_FLOAT)0.0))) <
    ((OTA_FLOAT)1.0)) ? ((((slopeFac*errFac) > ((OTA_FLOAT)0.0)) ? (slopeFac*errFac) :
    ((OTA_FLOAT)0.0))) : ((OTA_FLOAT)1.0))
3811
3812    }
3813
3814    bool CTempAlignment::RunPrealignment(int* pActiveFrameFlags, OTA_FLOAT* ReliabilityPerFrame, int*
    pSearchRangePerMacroFrameLow, int *pSearchRangePerMacroFrameHigh, int TArunIndex, bool
    &doRevertToOPTprealignment)
3815    {
3816        bool rc=true
3817        long MaxDelayVecLen = mppSignals[1]->mSignalLength
3818
3819        void *PAHandle = NULL
3820        TraversalVecType const *pa_vec = NULL
3821
3822        if (!doRevertToOPTprealignment)
3823        {
3824            rc = SectionsFromSQPrealignment(&PAHandle)
3825            fflush(mProcessData.mpLogFile)
3826
3827            OTA_FLOAT landmarkPA_matchQual = PreAlignment_GetMatchQuality(PAHandle)
3828
3829
3830            doRevertToOPTprealignment  = landmarkPA_matchQual < (OTA_FLOAT)0.75
3831
3832        }
3833
3834        if (!doRevertToOPTprealignment)
3835        {
3836            int   pa_sampleRate       = (int) ((CAudioSignal*)mppSignals[0])->mSampleRate
3837            OTA_FLOAT pa_frameLengthInSec = mMacroFrameSize / (OTA_FLOAT)pa_sampleRate
3838            pa_vec = PreAlignment_Traverse(PAHandle, pa_frameLengthInSec, pa_frameLengthInSec)
3839            if (pa_vec == NULL)
3840            {
3841                PreAlignment_Free(&PAHandle)
3842                doRevertToOPTprealignment = true
3843            }
3844
3845            OTA_FLOAT timeDriftFac = DetectTAtimeDrift((void*)pa_vec, pa_frameLengthInSec,
    mMacroFrameSize)
3846
3847            if (timeDriftFac > (OTA_FLOAT)0.30)
3848                doRevertToOPTprealignment = true
3849
3850
3851        }
3852
3853        if (!doRevertToOPTprealignment)
3854        {
3855
3856            rc = SegmentList2DelayVector(&PAHandle, (void**)&pa_vec, 1, pActiveFrameFlags,
    ReliabilityPerFrame, pSearchRangePerMacroFrameLow, pSearchRangePerMacroFrameHigh)
3857            fflush(mProcessData.mpLogFile)
3858
3859                int i
3860                for (i=0  i<mNumMacroFrames && !pActiveFrameFlags[i]  i++)
3861
3862                if (i<mNumMacroFrames)
3863                {
3864                    int Delay = mpDelayInSamplesPerFrame[i]
3865                    OTA_FLOAT Reliability = 0
3866                    int SL = pSearchRangePerMacroFrameLow[i]
3867                    int SH = pSearchRangePerMacroFrameHigh[i]
3868                    for (int j=0  j<i  j++)
3869                    {
```

```
3870                    mpDelayInSamplesPerFrame[j] = Delay
3871                    ReliabilityPerFrame[j] = Reliability
3872                    pSearchRangePerMacroFrameLow[j] = SL
3873                    pSearchRangePerMacroFrameHigh[j] = SH
3874                }
3875            }
3876
3877        PreAlignment_Free(&PAHandle)
3878        pa_vec    = NULL
3879        PAHandle = NULL
3880    }
3881    else
3882    {
3883        rc = true
3884
3885        PreAlignment_Free(&PAHandle)
3886        PAHandle = NULL
3887        pa_vec    = NULL
3888
3889
3890        ResetSectionData()
3891
3892        int WorstResolutionInSamples=0
3893
3894        {
3895            int StartFrame=0
3896            int* pActiveFrameFlagsRef = new int[MaxDelayVecLen]
3897            int NumRefFrames=mpActiveFrameDetection->GetActiveFrameFlags(0, 0,
    mProcessData.mStepSize, pActiveFrameFlagsRef, MaxDelayVecLen)
3898            while (StartFrame<NumRefFrames && pActiveFrameFlagsRef[StartFrame]<=0) StartFrame++
3899            GetDelayLimits(FramesToSamples(StartFrame), &mProcessData.mMaxStaticDelayInMs,
    &mProcessData.mMinStaticDelayInMs)
3900            delete[] pActiveFrameFlagsRef
3901        }
3902
3903        bool HighReliableInitialDelay=EstimateOverallDelaySimpleLimits(&mOverallDelayEstimate,
    &mOverallDelayEstimateReliability, &mOverallDelayEstimate1st,
    &mOverallDelayEstimateReliability1st,
3904                                                &mOverallDelayEstimate2nd,
    &mOverallDelayEstimateReliability2nd,
    &WorstResolutionInSamples, false)
3905        if (HighReliableInitialDelay)
3906        {
3907
3908            int       OverallDelayEstimate
3909            int       OverallDelayEstimate1st
3910            int       OverallDelayEstimate2nd
3911            OTA_FLOAT OverallDelayEstimateReliability
3912            OTA_FLOAT OverallDelayEstimateReliability1st
3913            OTA_FLOAT OverallDelayEstimateReliability2nd
3914            int WorstResolutionInSamples2
3915            if (0 && EstimateOverallDelaySimpleLimits(&OverallDelayEstimate,
    &OverallDelayEstimateReliability, &OverallDelayEstimate1st,
    &OverallDelayEstimateReliability1st,
3916                                                &OverallDelayEstimate2nd,
    &OverallDelayEstimateReliability2nd,
    &WorstResolutionInSamples2, true))
3917            {
3918                mOverallDelayEstimate = OverallDelayEstimate
3919                mOverallDelayEstimate1st = OverallDelayEstimate1st
3920                mOverallDelayEstimate2nd = OverallDelayEstimate2nd
3921                mOverallDelayEstimateReliability = OverallDelayEstimateReliability
3922                mOverallDelayEstimateReliability1st = OverallDelayEstimateReliability1st
3923                mOverallDelayEstimateReliability2nd = OverallDelayEstimateReliability2nd
3924                WorstResolutionInSamples = WorstResolutionInSamples2
3925            }
3926        }
3927
```

```
3928        mNumReparsePoints = IdentifyReparsePoints(mpReparsePoints, 100, pActiveFrameFlags,
    mOverallDelayEstimate, mOverallDelayEstimateReliability,
3929                                                  mOverallDelayEstimate1st,
    mOverallDelayEstimateReliability1st,
    mOverallDelayEstimate2nd,
    mOverallDelayEstimateReliability2nd)
3930
3931
3932        GetInitialDelaysInSamples(mpReparsePoints, mNumReparsePoints, pActiveFrameFlags,
    mOverallDelayEstimate, mOverallDelayEstimateReliability, &WorstResolutionInSamples)
3933        mStartSampleRef = mpReparsePoints[0].Ref.Start
3934        mStartSampleDeg = mpReparsePoints[0].Deg.Start
3935
3936        SetActiveFrameFlags(mMacroFrameSize, pActiveFrameFlags)
3937
3938        for (int r=0  r<mNumReparsePoints  r++)
3939        {
3940
3941            if (mpReparsePoints[r].Reliability>0.95 && HighReliableInitialDelay)
3942            {
3943
3944                mpReparsePoints[r].MinDelayVarInSamples = -4*WorstResolutionInSamples
3945                mpReparsePoints[r].MaxDelayVarInSamples = 4*WorstResolutionInSamples
3946
3947            }
3948            else
3949            {
3950                mpReparsePoints[r].MinDelayVarInSamples = mProcessData.mMinLowVarDelayInSamples
3951                mpReparsePoints[r].MaxDelayVarInSamples = mProcessData.mMaxHighVarDelayInSamples
3952            }
3953
3954        }
3955
3956        ReparseSections2DelayVector(ReliabilityPerFrame, pSearchRangePerMacroFrameLow,
    pSearchRangePerMacroFrameHigh, WorstResolutionInSamples)
3957
3958    }
3959
3960    return rc
3961 }
3962
3963
3964 //Input:
3965     //&IterationData
3966     // pActiveFrameFlags,  DelayVec
3967     //FrameWithLastValidDelay
3968     //Path, ReliabilityPerFrame
3969
3970 //Output:
3971 //  ReliabilityPerFrame, DelayVec
3972
3973 void CTempAlignment::LogDelayVec(const char* Title, int FeatureLength, long* DelayVec, OTA_FLOAT*
    ReliabilityPerFrame, int* pActiveFrameFlags)
3974 {
3975
3976 }
3977
3978 void CTempAlignment::CoarseAlignmentLog(FILE* pLogFile, long* DelayVec, int FeatureLength, int*
    Path, CProcessData* pIterationData, int* pRelativeDelayPerFrame, OTA_FLOAT* ReliabilityPerFrame)
3979 {
3980
3981 }
3982
3983 long CTempAlignment::modifyActiveFrameFlags(unsigned int numSimpleAnalysisFrames, int*
    pActiveFrameFlags, int* pActiveFrameFlagsSimplified)
3984 {
3985     bool activeBlock = false
3986     int activeBlockLength = 0
```

```
3987      matibZero(pActiveFrameFlagsSimplified, mNumMacroFrames)
3988
3989      long startFrameSimplified = -1
3990      for(int frame = 0  frame < mNumMacroFrames  frame++)
3991      {
3992          if(activeBlock)
3993          {
3994              if(pActiveFrameFlags[frame])
3995              {
3996                  if(activeBlockLength < numSimpleAnalysisFrames)
3997                      pActiveFrameFlagsSimplified[frame] = 1
3998                  activeBlockLength++
3999              }
4000              else
4001              {
4002
4003                  activeBlock = false
4004                  activeBlockLength = 0
4005              }
4006          }
4007          else
4008              if(pActiveFrameFlags[frame])
4009              {
4010                  activeBlock = true
4011                  if(startFrameSimplified == -1)
4012                      startFrameSimplified = frame
4013                  pActiveFrameFlagsSimplified[frame] = 1
4014                  activeBlockLength++
4015              }
4016      }
4017      return startFrameSimplified
4018  }
4019
4020  bool CTempAlignment::CoarseAlignmentFirstRun2(CProcessData* pIterationData, int* pActiveFrameFlags,
      int StartFrame, int NumFrames, int* DelayVecOffset, long* DelayVec, OTA_FLOAT* ReliabilityPerFrame)
4021  {
4022      int d
4023
4024      int AlertThresholdP =
      (int)(0.95*(pIterationData->mMaxHighVarDelay-pIterationData->mMinLowVarDelay))
4025      int AlertThresholdM =
      (int)(0.05*(pIterationData->mMaxHighVarDelay-pIterationData->mMinLowVarDelay))
4026      bool Alert=false
4027      int LastGoodFrame=-1
4028
4029      for (d=0  d<NumFrames-1  d++)
4030      {
4031          if (pActiveFrameFlags[d]
4032               && (DelayVecOffset[d]>AlertThresholdP || DelayVecOffset[d]<AlertThresholdM)
4033               && (DelayVecOffset[d+1]>AlertThresholdP || DelayVecOffset[d+1]<AlertThresholdM)
4034               && ReliabilityPerFrame[d]>0.7)
4035
4036          {
4037              LastGoodFrame = d-1
4038              Alert=true
4039              break
4040          }
4041      }
4042
4043      if (Alert && LastGoodFrame>=0 && pActiveFrameFlags[LastGoodFrame])
4044      {
4045          int UseDelay = DelayVec[LastGoodFrame]
4046          d = LastGoodFrame
4047          while (d<NumFrames && pActiveFrameFlags[d])
4048              DelayVec[d++] = UseDelay
4049      }
4050
4051      return Alert
```

```
4052 }
4053
4054 int CTempAlignment::CoarseAlignmentNewWindowSize(CProcessData* pIterationData, int Loop, int
     DegStep, int* pSearchRangeLow, int* pSearchRangeHigh, long* DelayVec, long* pDelayInSamples,
     PLOT_VECTOR* pVecs, int* pNumVecs)
4055 {
4056     int MinLowVarDelayInFF
4057     int MaxHighVarDelayInFF
4058     int LastWindowSize = pIterationData->mWindowSize
4059
4060     pIterationData->mMinLowVarDelayInSamples  = -LastWindowSize*2
4061     pIterationData->mMaxHighVarDelayInSamples =  LastWindowSize*2
4062
4063
4064     OTA_FLOAT IterationRatio =
     (pIterationData->mpWindowSize[Loop-1]-pIterationData->mpOverlap[Loop-1]) /
     (pIterationData->mpWindowSize[Loop]-pIterationData->mpOverlap[Loop])
4065
4066     pIterationData->Init(Loop, 1.0)
4067     DegStep *= IterationRatio
4068
4069     MinLowVarDelayInFF = pIterationData->mMinLowVarDelayInSamples / DegStep
4070     MaxHighVarDelayInFF = pIterationData->mMaxHighVarDelayInSamples / DegStep
4071
4072     OTA_FLOAT StepF = (OTA_FLOAT)pIterationData->mStepSize
4073
4074     for (int d=0  d<mNumMacroFrames  d++)
4075     {
4076         DelayVec[d] *= IterationRatio
4077
4078         mCAIntermediate.pSearchRangeLow[d] *= IterationRatio
4079         mCAIntermediate.pSearchRangeHigh[d] *= IterationRatio
4080
4081         int SearchRangeInSamplesIn = mCAIntermediate.pSearchRangeHighIn[d] -
     mCAIntermediate.pSearchRangeLowIn[d]
4082         if (SearchRangeInSamplesIn<StepF)
4083         {
4084             int SearchRangeInSamples = (mCAIntermediate.pSearchRangeHigh[d] -
     mCAIntermediate.pSearchRangeLow[d])* StepF
4085             if (SearchRangeInSamples>SearchRangeInSamplesIn)
4086             {
4087                 if (DelayVec[d]*StepF<mCAIntermediate.pSearchRangeLowIn[d] ||
     DelayVec[d]*StepF>mCAIntermediate.pSearchRangeHighIn[d])
4088                     DelayVec[d] = (long)floor(pDelayInSamples[d] / StepF)
4089
4090                 if (mCAIntermediate.pSearchRangeHighIn[d]!=0 &&
     mCAIntermediate.pSearchRangeLowIn[d]!=0)
4091                 {
4092                     int SearchRangeShift = pDelayInSamples[d]-DelayVec[d]*StepF
4093                     mCAIntermediate.pSearchRangeLow[d]  = mCAIntermediate.pSearchRangeLowIn[d] +
     SearchRangeShift
4094                     mCAIntermediate.pSearchRangeLow[d]  = (((mCAIntermediate.pSearchRangeLow[d]) <
     (0)) ? (mCAIntermediate.pSearchRangeLow[d]) : (0))
4095                     mCAIntermediate.pSearchRangeHigh[d] = mCAIntermediate.pSearchRangeHighIn[d] +
     SearchRangeShift
4096                     mCAIntermediate.pSearchRangeLow[d] =
     (int)floor(mCAIntermediate.pSearchRangeLow[d] /StepF)
4097                     mCAIntermediate.pSearchRangeHigh[d] = (int)ceil
     (mCAIntermediate.pSearchRangeHigh[d]/StepF)
4098                 }
4099             }
4100         }
4101
4102     }
4103
4104     return DegStep
4105 }
4106
```

```
4107 void CTempAlignment::CoarseAlignmentReduceSearchRangeAtSectionEnd(CProcessData* pIterationData)
4108 {
4109     int Frame=0
4110     int NumConstFramesBeforePause = PIMSecondsToFrames(150)
4111     int ReducedSearchRange  = PIMSecondsToFrames(32)
4112 
4113     int* pActiveFrameFlags=mCAIntermediate.pActiveFrameFlags
4114     int IsActiveSection = pActiveFrameFlags[Frame]
4115     for (Frame=0  Frame<mNumMacroFrames  Frame++)
4116     {
4117         if (IsActiveSection)
4118         {
4119             if(!pActiveFrameFlags[Frame])
4120             {
4121                 int LastDelay = mCAIntermediate.pDelayVec[(((0) > (Frame-NumConstFramesBeforePause)) ? (0) : (Frame-NumConstFramesBeforePause))]
4122 
4123                 for (int j=Frame-1  j>=Frame-NumConstFramesBeforePause && j>=0 && pActiveFrameFlags[j] j++)
4124                 {
4125 
4126                     int CenterIndex = -mCAIntermediate.pDelayVec[j] + LastDelay
4127                     mCAIntermediate.pSearchRangeLow[j]  = (((mCAIntermediate.pSearchRangeLow[j]) > (CenterIndex-ReducedSearchRange)) ? (mCAIntermediate.pSearchRangeLow[j]) : (CenterIndex-ReducedSearchRange))
4128                     mCAIntermediate.pSearchRangeHigh[j] = (((mCAIntermediate.pSearchRangeHigh[j]) < (CenterIndex+ReducedSearchRange)) ? (mCAIntermediate.pSearchRangeHigh[j]) : (CenterIndex+ReducedSearchRange))
4129 
4130                     mCAIntermediate.pOptionsApplied[j] |= APPL_REDUCED_SEARCHRANGE_AT_SECTION_END
4131                 }
4132             }
4133         }
4134 
4135         IsActiveSection = pActiveFrameFlags[Frame]
4136     }
4137 }
4138 
4139 bool CTempAlignment::CoarseAlignment(CProcessData* pIterationData, int* pActiveFrameFlags, int StartFrame, int* pSearchRangeLow, int* pSearchRangeHigh, long* DelayVec, OTA_FLOAT* ReliabilityPerFrame, bool doRevertToOPTprealignment, PLOT_VECTOR* pVecs, int* pNumVecs)
4140 {
4141 
4142 
4143     bool rc = true
4144     int StartPlotIteration=mProcessData.mStartPlotIteration
4145     int LastPlotIteration =mProcessData.mLastPlotIteration
4146     bool EnablePlotting=mProcessData.mEnablePlotting
4147     int i, f, d
4148 
4149     OTA_FLIST_TYPE FeatureListType = OTA_FLTYPE_COARSE_ALIGN
4150 
4151     long* pDelayInSamples = (long*)matMalloc(sizeof(long)*mNumMacroFrames)
4152     memcpy(pDelayInSamples, DelayVec, sizeof(long)*mNumMacroFrames)
4153     mCAIntermediate.pDelayVec = DelayVec
4154     mCAIntermediate.pActiveFrameFlags = pActiveFrameFlags
4155     int *Path = mCAIntermediate.pOptOffset
4156 
4157     memcpy(mCAIntermediate.pSearchRangeLow,  pSearchRangeLow,  sizeof(int)*mNumMacroFrames)
4158     memcpy(mCAIntermediate.pSearchRangeHigh, pSearchRangeHigh, sizeof(int)*mNumMacroFrames)
4159 
4160     mCAIntermediate.pSearchRangeLowIn = pSearchRangeLow
4161     mCAIntermediate.pSearchRangeHighIn = pSearchRangeHigh
4162 
4163     int Loop = 3
4164     pIterationData->Init(Loop, 1.0)
4165     int DegStep = (pIterationData->mpWindowSize[1]-pIterationData->mpOverlap[1]) / (pIterationData->mpWindowSize[Loop]-pIterationData->mpOverlap[Loop])
```

```
4166
4167      float StepF = (float)pIterationData->mStepSize
4168
4169      for (d=0  d<mNumMacroFrames  d++)
4170      {
4171          DelayVec[d] = (long)floor(pDelayInSamples[d] / StepF)
4172
4173          if (mCAIntermediate.pSearchRangeHigh[d]!=0 && mCAIntermediate.pSearchRangeLow[d]!=0)
4174          {
4175              int SearchRangeShift = pDelayInSamples[d]-DelayVec[d]*pIterationData->mStepSize
4176              mCAIntermediate.pSearchRangeLow[d] += SearchRangeShift
4177              mCAIntermediate.pSearchRangeLow[d] = (((mCAIntermediate.pSearchRangeLow[d]) < (0)) ?
    (mCAIntermediate.pSearchRangeLow[d]) : (0))
4178              mCAIntermediate.pSearchRangeHigh[d] += SearchRangeShift
4179          }
4180
4181          mCAIntermediate.pSearchRangeLow[d] =  (int)floor(mCAIntermediate.pSearchRangeLow[d] /StepF)
4182          mCAIntermediate.pSearchRangeHigh[d] = (int)ceil (mCAIntermediate.pSearchRangeHigh[d]/StepF)
4183      }
4184
4185      int* FrameWithLastValidDelay = new int[mNumMacroFrames]
4186      for (d=0  d<mNumMacroFrames  d++)
4187          FrameWithLastValidDelay[d] = d
4188      mCAIntermediate.pFrameWithLastValidDelay = FrameWithLastValidDelay
4189
4190      long MinDelay = matMin(DelayVec, mNumMacroFrames)
4191      long MaxDelay = matMax(DelayVec, mNumMacroFrames)
4192      long startFrameSimplified = -1
4193      int *pActiveFrameFlagsSimplified = 0
4194      OTA_FLOAT MinReliability = matMin(ReliabilityPerFrame, mNumMacroFrames)
4195      bool simplified = false
4196
4197      unsigned int numSimpleAnalysisFrames = (unsigned
    int)(400*(double)(pIterationData->mSamplerate)/1000.0/(double)mMacroFrameSize)
4198
4199      if (MinDelay==MaxDelay && MinReliability > 0.9)
4200      {
4201
4202          simplified = true
4203          FeatureListType = OTA_FLTYPE_COARSE_ALIGN_SIMPLIFIED
4204
4205          pActiveFrameFlagsSimplified = new int[mNumMacroFrames]
4206
4207          startFrameSimplified = modifyActiveFrameFlags(numSimpleAnalysisFrames, pActiveFrameFlags,
    pActiveFrameFlagsSimplified)
4208          mCAIntermediate.pActiveFrameFlags = pActiveFrameFlagsSimplified
4209      }
4210      else
4211          mCAIntermediate.pActiveFrameFlags = pActiveFrameFlags
4212
4213      OTA_FLOAT*  pDelayVecBackup=new OTA_FLOAT[mNumMacroFrames]
4214      int*        pFrameWithLastValidDelayBackup=new int[mNumMacroFrames]
4215      bool AllIterationsDone=false
4216      bool FirstRun=true
4217      bool GoToNextIteration=true
4218      bool RecalculateFeatures=true
4219      bool ReprocessCorrelationMatrix=false
4220      int DelayLimitsExceeded = 0
4221      while(rc && !AllIterationsDone)
4222      {
4223          if (!FirstRun)
4224              FeatureListType = OTA_FLTYPE_COARSE_ALIGN
4225
4226          mCAIntermediate.CurrentLoop = Loop
4227          GoToNextIteration=true
4228
4229          for (i=0  i<mNumMacroFrames  i++)
4230              pDelayVecBackup[i] = DelayVec[i]
```

```
4231        for (i=0  i<mNumMacroFrames  i++)
4232            pFrameWithLastValidDelayBackup[i] = FrameWithLastValidDelay[i]
4233
4234        GetPitchVector(1, 0, mCAIntermediate.pPitchVec, mNumMacroFrames, mMacroFrameSize)
4235
4236        if (RecalculateFeatures)
4237        {
4238
4239            rc = mpFeatureList->Create(mppSignals, pIterationData, FeatureListType)
4240            ReprocessCorrelationMatrix = true
4241        }
4242
4243        if (ReprocessCorrelationMatrix)
4244        {
4245            for (int i=0  i<mNumMacroFrames  i++)
4246                mCAIntermediate.pOptionsApplied[i]=0
4247
4248
4249            if (rc) rc = mpDelaySearch->CreateMatrix(mpFeatureList, &mCAIntermediate,
    pIterationData, mNumMacroFrames, DegStep)
4250
4251            mProcessData.mP.mViterbi.UseRelDistance = true
4252
4253            //Combine all matrices to the matrix[0]. CombineMatricesAndFeatures() should be
    overloaded by the
4254            //specific implementation. The default version does nothing.
4255            //After this method was called, the correlation matrix for feature 0 left
4256            //is the only one which is evaluated.
4257            if (rc) rc = mpDelaySearch->CombineMatricesAndFeatures(StartFrame, DegStep,
    &mCAIntermediate)
4258        }
4259
4260        int FeatureLength = mpDelaySearch->mpCorrMatrix[0][0].mNumMacroFrames
4261
4262        //If combining the matrices decided that some frames have invalid delays, then correct
4263        //the delay of those frames to match the delay of the last valid frame.
4264        //CombineMatricesAndFeatures() must have set the correlation vectors of those
4265        //frames to ...0 0 0 0 0 0 0 0 0 0... This will force the Viterbi algorithm
4266        //to keep the delay constant.
4267        //WARNING: This shifts delay jumps during inactive phases to the beginning of
4268        //the next active section!
4269
4270        int* pRelativeDelayPerFrame = mCAIntermediate.pRelativeDelayPerFrame
4271        pRelativeDelayPerFrame[0] = 0
4272        for (f=1  f<mNumMacroFrames  f++)
4273            pRelativeDelayPerFrame[f] = DelayVec[f] - DelayVec[f-1]
4274
4275        if(1 || mCAIntermediate.CurrentLoop==3)
4276            CoarseAlignmentReduceSearchRangeAtSectionEnd(pIterationData)
4277
4278        if (rc) rc = mpDelaySearch->CalcOptimumPath(DegStep, &mCAIntermediate, 0, 0,
    ReliabilityPerFrame)
4279
4280        OTA_FLOAT MinReliability=1.0
4281        for (i=0  i<FeatureLength  i++)
4282        {
4283            mpReliabilityPerFrame[i] = ReliabilityPerFrame[i]
4284            if (ReliabilityPerFrame[i]>0 && ReliabilityPerFrame[i]<MinReliability)
4285                MinReliability=ReliabilityPerFrame[i]
4286        }
4287
4288        CoarseAlignmentLog(pLogFile, DelayVec, FeatureLength, Path, pIterationData,
    pRelativeDelayPerFrame, ReliabilityPerFrame)
4289
4290        for (d=0  d<FeatureLength  d++)
4291            DelayVec[d] = DelayVec[d] + Path[d] + pIterationData->mMinLowVarDelay
4292
4293        mpDelaySearch->CleanupPath(&mCAIntermediate, DelayVec, FeatureLength,
```

```
    FrameWithLastValidDelay, DegStep)
4294        LogDelayVec("DelayVec per frame in CA after CleanupPath()", FeatureLength, DelayVec,
    ReliabilityPerFrame, pActiveFrameFlags)
4295
4296        if (FirstRun)
4297        {
4298            bool DoAgain = CoarseAlignmentFirstRun2(pIterationData,
    mCAIntermediate.pActiveFrameFlags, StartFrame, FeatureLength, Path, DelayVec,
    ReliabilityPerFrame)
4299            if (DoAgain && !simplified)
4300            {
4301                GoToNextIteration = false
4302                RecalculateFeatures = false
4303                ReprocessCorrelationMatrix=true
4304                DelayLimitsExceeded++
4305            }
4306            else GoToNextIteration = true
4307        }
4308
4309        if (GoToNextIteration)
4310        {
4311
4312            long MinDelay = matMin(DelayVec, mNumMacroFrames)
4313            long MaxDelay = matMax(DelayVec, mNumMacroFrames)
4314
4315            if ((MaxDelay-MinDelay)>1 || MinReliability < 0.9)
4316            {
4317                if(simplified)
4318                {
4319
4320
4321                    for (int i=0  i<mNumMacroFrames  i++)
4322                        DelayVec[i] = (long)pDelayVecBackup[i]
4323                    for (int i=0  i<mNumMacroFrames  i++)
4324                        FrameWithLastValidDelay[i] = pFrameWithLastValidDelayBackup[i]
4325
4326                    FeatureListType = OTA_FLTYPE_COARSE_ALIGN
4327                    GoToNextIteration = false
4328                    RecalculateFeatures = true
4329                    simplified = false
4330                    mCAIntermediate.pActiveFrameFlags = pActiveFrameFlags
4331                }
4332            }
4333        }
4334
4335        //Report large delay variations
4336        if (GoToNextIteration && mProcessData.mpLogFile)
4337        {
4338
4339
4340            for (d=1  d<FeatureLength  d++)
4341            {
4342                int DelayDifferenceInFrames = DelayVec[d]-DelayVec[d-1]
4343                if (abs(DelayDifferenceInFrames)>20)
4344
4345            }
4346        }
4347
4348        if (true)
4349        {
4350            if (!GoToNextIteration)
4351            {
4352                if (FirstRun && DelayLimitsExceeded>1)
4353                {
4354
4355
4356                    pIterationData->mMinLowVarDelayInSamples *= 2
4357                    pIterationData->mMaxHighVarDelayInSamples *= 2
```

```
4358
4359                    if (pIterationData->mMinLowVarDelayInSamples <
    -((int)(0.3*pIterationData->mSamplerate)))
4360                    {
4361                        pIterationData->mMinLowVarDelayInSamples =
    -((int)(0.3*pIterationData->mSamplerate))
4362                        GoToNextIteration=true
4363                    }
4364                    if (pIterationData->mMaxHighVarDelayInSamples >
    ((int)(0.3*pIterationData->mSamplerate)))
4365                    {
4366                        pIterationData->mMaxHighVarDelayInSamples =
    ((int)(0.3*pIterationData->mSamplerate))
4367                        GoToNextIteration=true
4368                    }
4369                    pIterationData->Init(Loop, 1.0)
4370                }
4371            }
4372        }
4373
4374        //*********************************************************
4375        //Choose the next iteration parameters or terminate the loop
4376        if (GoToNextIteration)
4377        {
4378            int MinLowVarDelayInFF
4379            int MaxHighVarDelayInFF
4380            int LastWindowSize = pIterationData->mWindowSize
4381
4382            pIterationData->mMinLowVarDelayInSamples  = -LastWindowSize*2
4383            pIterationData->mMaxHighVarDelayInSamples =  LastWindowSize*2
4384
4385            if (mProcessData.mEnablePlotting)
4386                SetVecInfoTimeSeries(&pVecs[(*pNumVecs)++], mpDelayInSamplesPerFrame,
    mNumMacroFrames, 0,0, (OTA_FLOAT)pIterationData->mStepSize, "Delay After CA
    Downsampling %d", pIterationData->mStepSize)
4387
4388            if (++Loop<8 && pIterationData->mpWindowSize[Loop]>0)
4389            {
4390                DegStep = CoarseAlignmentNewWindowSize(pIterationData, Loop, DegStep,
    pSearchRangeLow, pSearchRangeHigh, DelayVec, pDelayInSamples, pVecs, pNumVecs)
4391                RecalculateFeatures = true
4392            }
4393            else
4394            {
4395                AllIterationsDone = true
4396
4397            }
4398            FirstRun = false
4399        }
4400    }
4401
4402    {
4403        OTA_FLOAT StepF = (OTA_FLOAT)pIterationData->mStepSize
4404        for (int d=0  d<mNumMacroFrames  d++)
4405        {
4406            if (pSearchRangeLow[d]==0 && pSearchRangeHigh[d]==0)
4407                DelayVec[d] = (long)floor(pDelayInSamples[d] / StepF)
4408
4409            pSearchRangeLow[d] = (((pSearchRangeLow[d]) >
    (pIterationData->mMinLowVarDelayInSamples)) ? (pSearchRangeLow[d]) :
    (pIterationData->mMinLowVarDelayInSamples))
4410            pSearchRangeHigh[d] = (((pSearchRangeHigh[d]) <
    (pIterationData->mMaxHighVarDelayInSamples)) ? (pSearchRangeHigh[d]) :
    (pIterationData->mMaxHighVarDelayInSamples))
4411        }
4412    }
4413
4414    delete[] pDelayVecBackup
```

```
4415     delete[] FrameWithLastValidDelay
4416     delete[] pFrameWithLastValidDelayBackup
4417
4418     if (pActiveFrameFlagsSimplified)
4419     {
4420         delete[] pActiveFrameFlagsSimplified
4421         mCAIntermediate.pActiveFrameFlags = pActiveFrameFlags
4422     }
4423
4424     if (simplified)
4425     {
4426
4427         int RefFrame = startFrameSimplified + numSimpleAnalysisFrames/2
4428         for (d = startFrameSimplified + numSimpleAnalysisFrames  d < mNumMacroFrames  d++)
4429         {
4430             mpReliabilityPerFrame[d] = mpReliabilityPerFrame[RefFrame]
4431             DelayVec[d] = DelayVec[RefFrame]
4432             ReliabilityPerFrame[d] = ReliabilityPerFrame[RefFrame]
4433         }
4434     }
4435
4436
4437
4438     for (i=0  i<mNumMacroFrames  i++)
4439         DelayVec[i] = DelayVec[i] * pIterationData->mStepSize
4440
4441     if (pDelayInSamples) matFree(pDelayInSamples)
4442
4443
4444
4445     return rc
4446 }
4447
4448 bool inline IsSampleInRefSpeech(int SampleIdx, int NumReparsePoints, REPARSE_POINT* ReparsePoints)
4449 {
4450     bool rc = false
4451     for (int i=0  i<NumReparsePoints  i++)
4452         if (ReparsePoints[i].Ref.Start<SampleIdx && ReparsePoints[i].Ref.End>SampleIdx)
4453             return true
4454     return rc
4455 }
4456
4457 void CTempAlignment::DetermineInvalidSections()
4458 {
4459
4460     int* pFlags = mpResults->mpIgnoreFlags
4461     int* pActiveFrameFlags = mFAIntermediate.pActiveFrameFlags
4462
4463     const int SignificantChange = MSecondsToSamples(50)
4464     const int SmallChange = MSecondsToSamples(10)
4465     for (int i=1  i<mNumMacroFrames  i++)
4466     {
4467         if (i>1 && pActiveFrameFlags[i] && pActiveFrameFlags[i-1]
4468             && mpDelayInSamplesPerFrame[i-1]-mpDelayInSamplesPerFrame[i]>0)
4469         {
4470             int FramesToSearch = (mpDelayInSamplesPerFrame[i-1]-mpDelayInSamplesPerFrame[i]) /
     mProcessData.mStepSize+1
4471
4472             for (int k=(((0) > (i-FramesToSearch)) ? (0) : (i-FramesToSearch))  k<i  k++)
4473                 pFlags[k] |= 0x0001
4474         }
4475
4476         if (i>1 && pActiveFrameFlags[i] && pActiveFrameFlags[i-1]
4477             && mpDelayInSamplesPerFrame[i-1]-mpDelayInSamplesPerFrame[i]>SignificantChange)
4478         {
4479             int FramesToSearch = (mpDelayInSamplesPerFrame[i-1]-mpDelayInSamplesPerFrame[i]) /
     mProcessData.mStepSize+1
4480
```

```
4481            for (int k=(((0) > (i-FramesToSearch)) ? (0) : (i-FramesToSearch))  k<i  k++)
4482            {
4483                pFlags[k] |= 0x0002
4484
4485                if (pActiveFrameFlags[k] && mFAIntermediate.pReliabilityPerFrame[k]<0.7 &&
    abs(int(mpDelayInSamplesPerFrame[k]-mpDelayInSamplesPerFrame[i-1]))<SmallChange
4486                    &&
    mFAIntermediate.pRefEnergy[k+mpDelayInSamplesPerFrame[k]/mProcessData.mStepSize]
    >10.0*mFAIntermediate.pDegEnergy[k])
4487                    pFlags[k] |= 0x0004
4488            }
4489        }
4490    }
4491
4492    for (int r = 0  r<mNumReparsePoints-1  r++)
4493    {
4494        int DegPause = mpReparsePoints[r+1].Deg.Start-mpReparsePoints[r].Deg.Start
4495        int RefPause = mpReparsePoints[r+1].Ref.Start-mpReparsePoints[r].Ref.Start
4496        if (DegPause>RefPause)
4497        {
4498            int f=mpReparsePoints[r].Deg.End/mMacroFrameSize+1
4499            int NextStart = mpReparsePoints[r+1].Deg.Start/mMacroFrameSize
4500
4501            while (f<NextStart && pActiveFrameFlags[f])
4502                f++
4503
4504            for (  f<NextStart  f++)
4505            {
4506                if (IsSampleInRefSpeech(f*mMacroFrameSize + mpDelayInSamplesPerFrame[f],
    mNumReparsePoints, mpReparsePoints))
4507                    pFlags[f] |= (0x0008 | 0x0001)
4508            }
4509
4510            while (f<mNumMacroFrames && !pActiveFrameFlags[f])
4511            {
4512                if (IsSampleInRefSpeech(f*mMacroFrameSize + mpDelayInSamplesPerFrame[f],
    mNumReparsePoints, mpReparsePoints))
4513                    pFlags[f] |= (0x0008 | 0x0001)
4514                f++
4515            }
4516        }
4517    }
4518
4519    bool startedInsertion = false
4520    int indexStartedInsertion = 0
4521    for (int i = 1  i<mNumMacroFrames  i++)
4522    {
4523        if (pActiveFrameFlags[i] && pActiveFrameFlags[i - 1] && mpDelayInSamplesPerFrame[i - 1] -
    mpDelayInSamplesPerFrame[i] > 0)
4524        {
4525            if (startedInsertion == false)
4526            {
4527                indexStartedInsertion = i
4528                startedInsertion = true
4529            }
4530        }
4531        else
4532        {
4533            if (startedInsertion == true)
4534            {
4535                if (mpDelayInSamplesPerFrame[indexStartedInsertion - 1] -
    mpDelayInSamplesPerFrame[i] > MSecondsToSamples(40))
4536                {
4537                    for (int k = indexStartedInsertion - 1  k < i  k++)
4538                        pFlags[k] |= 0x0008
4539                }
4540                startedInsertion = false
4541            }
```

```
4542         }
4543     }
4544 
4545     for (int i=1  i<mNumMacroFrames  i++)
4546     {
4547         if (pFlags[i] & 0x0004)
4548         {
4549             int DelayToLastGood = mpDelayInSamplesPerFrame[i]-mProcessData.mStepSize
4550             while (i<mNumMacroFrames && (pFlags[i] & 0x0004))
4551             {
4552                 mpDelayInSamplesPerFrame[i++] = DelayToLastGood
4553                 DelayToLastGood -= mProcessData.mStepSize
4554             }
4555         }
4556     }
4557 
4558 }
4559 
4560 bool CTempAlignment::Run(unsigned long Control, OTA_RESULT* pResult, int TArunIndex)
4561 {
4562 
4563 
4564     bool rc = true
4565     DEBUG_TRY
4566 
4567     int i, r, d
4568 
4569     int NumSpareFrames=0
4570 
4571     mNumMacroFrames=0
4572     mProcessData.Init(1, 1)
4573     mMacroFrameSize = mProcessData.mStepSize
4574 
4575     unsigned long TriggerPoint = 0
4576     int TriggerPointInFrames = 0
4577     mpResults->FirstRefSample = 0
4578     mpResults->FirstDegSample = 0
4579 
4580     long MaxDelayVecLen = mppSignals[1]->mSignalLength+TriggerPoint
4581 
4582     mpReparsePoints = new REPARSE_POINT[100]
4583 
4584     OTA_FLOAT RelativeSamplerateDifference = 0
4585     int *pActiveFrameFlags = 0
4586 
4587     mOverallDelayEstimate=0
4588     mOverallDelayEstimateReliability = -1
4589     mpDelayInSamplesPerFrame = 0
4590     mpReliabilityPerFrame = 0
4591 
4592     mStartOffset = FindUsedSectionOfRefSignal()
4593 
4594     int StartOffsetInFrames = abs(mStartOffset) / mMacroFrameSize
4595 
4596     if (mStartOffset)
4597     {
4598         if (mStartOffset<0)
4599             mppSignals[1]->SetOffset(-mStartOffset)
4600         else
4601             mppSignals[0]->SetOffset(mStartOffset)
4602 
4603         NumSpareFrames += StartOffsetInFrames
4604     }
4605 
4606     ResetSectionData()
4607 
4608     OTA_FLOAT ERef = mppSignals[0]->GetEnergy(0)
4609     OTA_FLOAT EDeg = mppSignals[1]->GetEnergy(0)
```

```
4610
4611      OTA_FLOAT Factor = ERef / EDeg
4612      if (Factor<0.1) Factor = 0.1
4613      if (Factor>10.0) Factor = 10.0
4614
4615      mppSignals[1]->Amplify(0, sqrt(Factor))
4616
4617      pActiveFrameFlags = (int*)matCalloc(MaxDelayVecLen, sizeof(int))
4618
4619      mpActiveFrameDetection->Init(&mProcessData)
4620      mpActiveFrameDetection->Start(mppSignals)
4621      mProcessData.Init(1, 1)
4622      mNumMacroFrames = mpActiveFrameDetection->GetActiveFrameFlags(1, 0, mProcessData.mStepSize,
    pActiveFrameFlags, MaxDelayVecLen)
4623
4624
4625      mpFeatureList2->Create(mppSignals, &mProcessData, OTA_ENERGYPERFRAME)
4626
4627      mpDelayInSamplesPerFrame = (long*)matCalloc((mNumMacroFrames+NumSpareFrames), sizeof(long))
4628
4629      mpReliabilityPerFrame = (OTA_FLOAT*)matCalloc((mNumMacroFrames+NumSpareFrames),
    sizeof(OTA_FLOAT))
4630
4631      OTA_FLOAT* ReliabilityPerFrame = (OTA_FLOAT*)matCalloc((mNumMacroFrames+NumSpareFrames),
    sizeof(OTA_FLOAT))
4632
4633      int* pSearchRangePerMacroFrameLow = (int*)matCalloc((mNumMacroFrames+NumSpareFrames),
    sizeof(int))
4634      int* pSearchRangePerMacroFrameHigh = (int*)matCalloc((mNumMacroFrames+NumSpareFrames),
    sizeof(int))
4635
4636      mpResults->mpIgnoreFlags = (int*)matMalloc(sizeof(int)*(mNumMacroFrames + NumSpareFrames))
4637      mpResults->mNumIngoreFlags = mNumMacroFrames + NumSpareFrames
4638      matbSet(0, mpResults->mpIgnoreFlags, mNumMacroFrames + NumSpareFrames)
4639
4640      bool doRevertToOPTprealignment = TArunIndex > 0
4641      if(mppSignals[1]->mSignalLength/mProcessData.mSamplerate > MAX_SPEECH_DURATION)
    doRevertToOPTprealignment=true
4642      if(mppSignals[0]->mSignalLength/mProcessData.mSamplerate > MAX_SPEECH_DURATION)
    doRevertToOPTprealignment=true
4643
4644      int StartFrame = 0
4645
4646      DEBUG_TRY
4647
4648      rc = RunPrealignment(pActiveFrameFlags, ReliabilityPerFrame, pSearchRangePerMacroFrameLow,
    pSearchRangePerMacroFrameHigh, TArunIndex, doRevertToOPTprealignment)
4649
4650      if (rc)
4651      {
4652          TACheckTimeMatEval(mh, 2, &ClockCycles, &TimeDiffInitalDelay)
4653          TACheckTimeMatInit(mh, 2)
4654
4655          StartFrame = (((0) > (mpReparsePoints[0].Deg.Start / mMacroFrameSize)) ? (0) :
    (mpReparsePoints[0].Deg.Start / mMacroFrameSize))
4656
4657      }
4658      else
4659      {
4660
4661      }
4662
4663      DEBUG_CATCH
4664
4665      //The initial delay is known now, mpDelayInSamplesPerFrame is filled with it,
4666      //pActiveFrameFlags contains for each macro frame an indication
4667      //whether the frame is active or not and StartFrame points to the first
4668      //really active frame. IterationData contains the window parameters
```

```
4669     //etc. for the following coarse alignment and the schedule for the
4670     //iterative increase of the delay resolution.
4671     //mpDelayInSamplesPerFrame must now be refined and ReliabilityPerFrame must be set
4672     //to the correlation found for each frame.
4673     //The coarse alignment will shift delay jumps during silence to the beginning of
4674     //the next active section. This must be corrected later in order to avoid problems
4675     //when converting deg delays to ref delays. Otherwise it will result in lost or
4676     //repeated frames.
4677
4678     CProcessData IterationData = mProcessData
4679     DEBUG_TRY
4680
4681     if (rc)
4682
4683     {
4684         mCAIntermediate.Init(mNumMacroFrames)
4685         mCAIntermediate.AvgEnergyRef = ERef
4686         mCAIntermediate.AvgEnergyDeg = EDeg
4687
4688         int Size = mpFeatureList2->GetFVector(0, 0, 0)->mSize
4689         int MinSize = (((Size) < (mNumMacroFrames)) ? (Size) : (mNumMacroFrames))
4690         matbCopy(mpFeatureList2->GetFVector(0, 0, 0)->mpVector, mCAIntermediate.pRefEnergy, MinSize)
4691         if (MinSize<mNumMacroFrames)
4692             matbSet(0.0, mCAIntermediate.pRefEnergy+MinSize, mNumMacroFrames-MinSize)
4693
4694         Size = mpFeatureList2->GetFVector(0, 1, 0)->mSize
4695         MinSize = (((Size) < (mNumMacroFrames)) ? (Size) : (mNumMacroFrames))
4696         matbCopy(mpFeatureList2->GetFVector(0, 1, 0)->mpVector, mCAIntermediate.pDegEnergy, MinSize)
4697         if (MinSize<mNumMacroFrames)
4698             matbSet(0.0, mCAIntermediate.pDegEnergy+MinSize, mNumMacroFrames-MinSize)
4699
4700         rc = CoarseAlignment(&IterationData, pActiveFrameFlags, StartFrame,
     pSearchRangePerMacroFrameLow, pSearchRangePerMacroFrameHigh, mpDelayInSamplesPerFrame,
     ReliabilityPerFrame, doRevertToOPTprealignment, pVecs, &NumVecs)
4701
4702
4703
4704         if (rc)
4705         {
4706
4707             //Shift delay changes from the start of active sections to the center of inactive
     intervalls.
4708             //To avoid problems with transition effects at the start of the section, the delay of
     the second
4709             //frame of the section is taken.
4710
4711
4712             mProcessData.Init(1, 1.0)
4713             for (r=1  r<mNumReparsePoints  r++)
4714             {
4715                 int DegEndSample = mpReparsePoints[r-1].Deg.End
4716                 int RefEndSample = mpReparsePoints[r-1].Ref.End
4717
4718                 int DegStartSample = mpReparsePoints[r].Deg.Start
4719                 int RefStartSample = mpReparsePoints[r].Ref.Start
4720
4721                 int CheckFrame = SamplesToFrames(DegStartSample)
4722                 if (CheckFrame<mNumMacroFrames-1 && CheckFrame>1)
4723                 {
4724                     int DelayAfter = mpDelayInSamplesPerFrame[CheckFrame+1]
4725
4726                     int ChangePosDeg
4727                     if (RefStartSample-RefEndSample<DegStartSample-DegEndSample)
4728                     {
4729                         ChangePosDeg = (((0) > (RefEndSample +
     (int)(0.5*(RefStartSample-RefEndSample)) - DelayAfter)) ? (0) :
     (RefEndSample + (int)(0.5*(RefStartSample-RefEndSample)) - DelayAfter))
4730
```

```
4731
4732                    }
4733                    else
4734                    {
4735                        ChangePosDeg = DegEndSample + (int)(0.5*(DegStartSample-DegEndSample))
4736
4737                    }
4738
4739                    for (i=SamplesToFrames(ChangePosDeg)  i<SamplesToFrames(DegStartSample)+1 &&
    i<mNumMacroFrames  i++)
4740                        mpDelayInSamplesPerFrame[i] = DelayAfter
4741                }
4742            }
4743
4744    DEBUG_CATCH
4745
4746    DEBUG_TRY
4747
4748    if (rc)
4749    {
4750        OTA_FLOAT LagToSamples=1
4751
4752        long* pDelayInSamplesPerFrameAfterCA = (long*)matMalloc(mNumMacroFrames * sizeof(long))
4753        for (i=0  i<mNumMacroFrames  i++)
4754            pDelayInSamplesPerFrameAfterCA[i] = mpDelayInSamplesPerFrame[i]
4755
4756
4757        mProcessData.Init(1, 1.0)
4758
4759        mFAIntermediate.Init(mNumMacroFrames)
4760        mFAIntermediate.AvgEnergyRef = ERef
4761        mFAIntermediate.AvgEnergyDeg = EDeg
4762        mFAIntermediate.pDelayVec = pDelayInSamplesPerFrameAfterCA
4763        mFAIntermediate.pPitchVec = mCAIntermediate.pPitchVec
4764        mFAIntermediate.pDegEnergy = mCAIntermediate.pDegEnergy
4765        mFAIntermediate.pRefEnergy = mCAIntermediate.pRefEnergy
4766        mFAIntermediate.pActiveFrameFlags = mCAIntermediate.pActiveFrameFlags
4767        mFAIntermediate.pReliabilityPerFrame = ReliabilityPerFrame
4768        mFAIntermediate.pSearchRangeLow = pSearchRangePerMacroFrameLow
4769        mFAIntermediate.pSearchRangeHigh = pSearchRangePerMacroFrameHigh
4770        mFAIntermediate.pConstDelayMarker = mCAIntermediate.pConstDelayMarker
4771
4772        if (Control & 0x1)
4773        {
4774
4775            bool IsSpecialAlignment =
    mProcessData.mDelayFineAlignCorrlen!=mProcessData.mSRDetectFineAlignCorrlen
4776
4777            mpDelaySearch->FineAlign(&mFAIntermediate, mppSignals, mpActiveFrameDetection,
    pDelayInSamplesPerFrameAfterCA, ReliabilityPerFrame, &mNumMacroFrames,
    mProcessData.mStepSize, 2*IterationData.mWindowSize,
    mProcessData.mSRDetectFineAlignCorrlen, IsSpecialAlignment?FA_FOR_SRDETECTION:0)
4778
4779            LagToSamples = 1
4780
4781            OTA_FLOAT RelativeSamplerateDifference_linear =
    GetSampleRateRatioDiff(mFAIntermediate.pActiveFrameFlags,pDelayInSamplesPerFrameAfterCA,
    mNumMacroFrames)
4782
4783            RelativeSamplerateDifference = RelativeSamplerateDifference_linear
4784
4785
4786        }
4787
4788        if (Control & 0x2 && mProcessData.mDelayFineAlignCorrlen>0
4789            &&
    (RelativeSamplerateDifference<1+mProcessData.mMaxToleratedRelativeSamplerateDifference
4790            && RelativeSamplerateDifference>1-mProcessData.mMaxToleratedRelativeSamplerateDifference
```

```
4791            || RelativeSamplerateDifference==-1 ))
4792        {
4793            if (mProcessData.mDelayFineAlignCorrlen==mProcessData.mSRDetectFineAlignCorrlen)
4794            {
4795                for (int i=0  i<mNumMacroFrames  i++)
4796                    mpDelayInSamplesPerFrame[i] = pDelayInSamplesPerFrameAfterCA[i]
4797            }
4798            else
4799            {
4800
4801
4802                mpDelaySearch->FineAlign(&mFAIntermediate, mppSignals, mpActiveFrameDetection,
    mpDelayInSamplesPerFrame, ReliabilityPerFrame, &mNumMacroFrames,
    mProcessData.mStepSize, 2*IterationData.mWindowSize,
    mProcessData.mSRDetectFineAlignCorrlen)
4803
4804            }
4805            LagToSamples = 1
4806        }
4807
4808        TACheckTimeMatEval(mh, 2, &ClockCycles, &TimeDiffFineAlignment)
4809        TACheckTimeMatInit(mh, 2)
4810
4811        OTA_FLOAT avgReliability  = (OTA_FLOAT)0.0
4812        int   numActiveFrames = 0
4813        for (i=0  i<mNumMacroFrames  i++)
4814        {
4815            mpReliabilityPerFrame[i] = ReliabilityPerFrame[i]
4816            if (pActiveFrameFlags[i])
4817            {
4818                avgReliability += ReliabilityPerFrame[i]
4819                numActiveFrames++
4820            }
4821        }
4822        avgReliability /= (((numActiveFrames) > (1)) ? (numActiveFrames) : (1))
4823        if (mpResults)
4824            mpResults->mAvgReliability = avgReliability
4825
4826
4827
4828        mProcessData.Init(1, 1.0)
4829
4830        if (pResult)
4831        {
4832            if(mpResults)
4833            {
4834                mpResults->mNumUtterances = 0
4835                mpResults->mpDelayUtterance = 0
4836                mpResults->mpStartSampleUtterance = 0
4837                mpResults->mpStopSampleUtterance = 0
4838
4839                mpResults->mNumFrames = mNumMacroFrames
4840
4841                DetermineInvalidSections()
4842
4843                if (Control & 0x4)
4844                    CreateUtteranceVectorsRef(&mpResults->mNumUtterances,
    &mpResults->mpStartSampleUtterance, &mpResults->mpStopSampleUtterance,
    &mpResults->mpDelayUtterance, mpReparsePoints, mNumReparsePoints)
4845                else if (Control & 0x8)
4846                    CreateUtteranceVectorsDeg(&mpResults->mNumUtterances,
    &mpResults->mpStartSampleUtterance, &mpResults->mpStopSampleUtterance,
    &mpResults->mpDelayUtterance, mpReparsePoints,
    mNumReparsePoints,pActiveFrameFlags, mNumMacroFrames, mMacroFrameSize)
4847
4848                if (mStartOffset)
4849                {
4850                    if (Control & 0x4)
```

```
4851                    {
4852                        for (i=0  i<mpResults->mNumUtterances  i++)
4853                            mpResults->mpDelayUtterance[i] -= mStartOffset
4854                        if (mStartOffset>0)
4855                        {
4856                            for (i=0  i<mpResults->mNumUtterances  i++)
4857                            {
4858                                mpResults->mpStartSampleUtterance[i] += mStartOffset
4859                                mpResults->mpStopSampleUtterance[i] += mStartOffset
4860                            }
4861                            for (i=0  i<mNumReparsePoints  i++)
4862                            {
4863                                mpReparsePoints[i].Deg.Start -= mStartOffset
4864                                mpReparsePoints[i].Deg.End -= mStartOffset
4865                            }
4866                        }
4867                        else
4868                        {
4869                            for (i=0  i<mNumReparsePoints  i++)
4870                            {
4871                                mpReparsePoints[i].Deg.Start -= mStartOffset
4872                                mpReparsePoints[i].Deg.End -= mStartOffset
4873                            }
4874                        }
4875                    }
4876                    else if (Control & 0x8)
4877                    {
4878                        for (i=0  i<mpResults->mNumUtterances  i++)
4879                            mpResults->mpDelayUtterance[i] += mStartOffset
4880                        if (mStartOffset<0)
4881                        {
4882                            for (i=0  i<mpResults->mNumUtterances  i++)
4883                            {
4884                                mpResults->mpStartSampleUtterance[i] -= mStartOffset
4885                                mpResults->mpStopSampleUtterance[i] -= mStartOffset
4886                            }
4887
4888                            for (i=0  i<mNumReparsePoints  i++)
4889                            {
4890                                mpReparsePoints[i].Deg.Start -= mStartOffset
4891                                mpReparsePoints[i].Deg.End -= mStartOffset
4892                            }
4893                        }
4894                        else
4895                        {
4896                            for (i=0  i<mNumReparsePoints  i++)
4897                            {
4898                                mpReparsePoints[i].Ref.Start += mStartOffset
4899                                mpReparsePoints[i].Ref.End += mStartOffset
4900                            }
4901                        }
4902                    }
4903
4904                    if (mStartOffset<0)
4905                    {
4906                        int StartFrame = (-mStartOffset) / mProcessData.mStepSize
4907                        for (i=mNumMacroFrames-1  i>=0  i--)
4908                        {
4909                            mpDelayInSamplesPerFrame[i+StartFrame] = mpDelayInSamplesPerFrame[i] +
    mStartOffset
4910                            mpReliabilityPerFrame[i+StartFrame] = mpReliabilityPerFrame[i]
4911                            pActiveFrameFlags[i + StartFrame] = pActiveFrameFlags[i]
4912                            mpResults->mpIgnoreFlags[i + StartFrame] = mpResults->mpIgnoreFlags[i]
4913                        }
4914                        for (i=0  i<StartFrame  i++)
4915                        {
4916                            mpDelayInSamplesPerFrame[i] = mpDelayInSamplesPerFrame[StartFrame]
4917                            mpReliabilityPerFrame[i] = mpReliabilityPerFrame[StartFrame]
```

```
4918                            pActiveFrameFlags[i] = pActiveFrameFlags[StartFrame]
4919                            mpResults->mpIgnoreFlags[i] = mpResults->mpIgnoreFlags[StartFrame]
4920                        }
4921                        mNumMacroFrames += StartFrame
4922                        mpResults->mNumFrames = mNumMacroFrames
4923                    }
4924                    else
4925                    {
4926                        for (i=0  i<mNumMacroFrames  i++)
4927                            mpDelayInSamplesPerFrame[i] += mStartOffset
4928                    }
4929
4930                }
4931
4932                if (Control & 0x1)
4933                    mpResults->mRelSamplerateDev = RelativeSamplerateDifference
4934                else
4935                    mpResults->mRelSamplerateDev = 1.0
4936                mpResults->mResolutionInSamples = mProcessData.mStepSize
4937                mpResults->mStepsize = mProcessData.mStepSize
4938
4939                mpResults->mpRefSections = new SECTION[mNumReparsePoints]
4940                mpResults->mpDegSections = new SECTION[mNumReparsePoints]
4941                for (i=0  i<mNumReparsePoints  i++)
4942                {
4943                    mpResults->mpRefSections[i] = mpReparsePoints[i].Ref
4944                    mpResults->mpDegSections[i] = mpReparsePoints[i].Deg
4945                }
4946                mpResults->mNumSections = mNumReparsePoints
4947
4948                OTA_FLOAT tempDegNoiseLevel, tempDegSignalLevel, tempRefNoiseLevel,
    tempRefSignalLevel
4949                OTA_FLOAT tempDegNoiseThreshold,tempRefNoiseThreshold
4950                mpActiveFrameDetection->GetLevels(0, 0, 1, &tempRefNoiseLevel, &tempRefSignalLevel,
    &tempRefNoiseThreshold, 0)
4951
4952                mpActiveFrameDetection->GetLevels(1, 0, 1, &tempDegNoiseLevel, &tempDegSignalLevel,
    &tempDegNoiseThreshold, 0)
4953
4954                mpResults->mAslFrames = pResult->mAslFrames
4955                mpResults->mAslFramelength = pResult->mAslFramelength
4956                mpResults->mpAslActiveFrameFlags =
    (int*)matMalloc(mpResults->mAslFrames*sizeof(int))
4957                mpResults->mAslFrames = mpActiveFrameDetection->GetActiveFrameFlags(0, 0,
    mpResults->mAslFramelength, mpResults->mpAslActiveFrameFlags, mpResults->mAslFrames,
    0)
4958
4959                mpResults->mAslFramesDeg = pResult->mAslFramesDeg
4960                mpResults->mpAslActiveFrameFlagsDeg =
    (int*)matMalloc(mpResults->mAslFramesDeg*sizeof(int))
4961                mpResults->mAslFramesDeg = mpActiveFrameDetection->GetActiveFrameFlags(1, 0,
    mpResults->mAslFramelength, mpResults->mpAslActiveFrameFlagsDeg,
    mpResults->mAslFramesDeg, 0)
4962
4963                mpResults->mSNRRefdB = 10*log10(tempRefSignalLevel/tempRefNoiseLevel)
4964                mpResults->mSNRDegdB = 10*log10(tempDegSignalLevel/tempDegNoiseLevel)
4965                mpResults->mNoiseLevelRef = tempRefNoiseLevel
4966                mpResults->mNoiseLevelDeg = tempDegNoiseLevel
4967                mpResults->mSignalLevelRef = tempRefSignalLevel
4968                mpResults->mSignalLevelDeg = tempDegSignalLevel
4969                mpResults->mNoiseThresholdRef = tempRefNoiseThreshold
4970                mpResults->mNoiseThresholdDeg = tempDegNoiseThreshold
4971
4972                mpResults->mpDelay            = (long*)matMalloc(mNumMacroFrames * sizeof(long))
4973                mpResults->mpReliability      = (OTA_FLOAT*)matMalloc(mNumMacroFrames *
    sizeof(OTA_FLOAT))
4974                mpResults->mpActiveFrameFlags = (int*)matMalloc(mNumMacroFrames * sizeof(int))
4975                for (i=0  i<mNumMacroFrames  i++)
```

```
                {
                    mpResults->mpDelay           [i] = mpDelayInSamplesPerFrame[i]
                }
                matbCopy(pActiveFrameFlags, mpResults->mpActiveFrameFlags, mNumMacroFrames)
                matbCopy(mpReliabilityPerFrame, mpResults->mpReliability, mNumMacroFrames)
            }
            else rc = false
        }
        if(pDelayInSamplesPerFrameAfterCA)
            matFree(pDelayInSamplesPerFrameAfterCA)
    }
    else
    {

    }

    DEBUG_CATCH

    if(ReliabilityPerFrame)
        matFree(ReliabilityPerFrame)
    if(pActiveFrameFlags)
        matFree(pActiveFrameFlags)
    if(pSearchRangePerMacroFrameLow)
        matFree(pSearchRangePerMacroFrameLow)
    if(pSearchRangePerMacroFrameHigh)
        matFree(pSearchRangePerMacroFrameHigh)

    TACheckTimeMatEval(mh, 2, &ClockCycles, &TimeDiffCleanup)

    if (mpResults)
    {
        mpResults->mTimeDiffs[0] = 0
        mpResults->mTimeDiffs[1] = TimeDiffInitalDelay
        mpResults->mTimeDiffs[2] = TimeDiffCoarseAlignment
        mpResults->mTimeDiffs[3] = TimeDiffFineAlignment
        mpResults->mTimeDiffs[4] = TimeDiffCleanup
    }

    if (pResult && mpResults)
        pResult->CopyFrom(mpResults)

    DEBUG_CATCH

    return rc
}

int CTempAlignment::GetPitchFrameSize()
{
    return mpResults->mPitchFrameSize
}

void inline CombineFirstTwoUtterancesRef(int NumUtterancesLeft, int* pStartUtt, int* pStopUtt, int*
pDelayUtt, bool* pIsInsideActiveSection)
{
    pStopUtt[0] = pStopUtt[1]
    for (int i=1  i<NumUtterancesLeft-1  i++)
    {
        pStartUtt[i] = pStartUtt[i+1]
        pStopUtt[i] = pStopUtt[i+1]
        pDelayUtt[i] = pDelayUtt[i+1]
        pIsInsideActiveSection[i] = pIsInsideActiveSection[i+1]
    }
}

void inline CombineFirstTwoUtterancesDeg(int NumUtterancesLeft, int* pStartUtt, int* pStopUtt, int*
pDelayUtt)
{
    pStopUtt[0] = pStopUtt[1]
```

```
5042      for (int i=1  i<NumUtterancesLeft-1  i++)
5043      {
5044          pStartUtt[i] = pStartUtt[i+1]
5045          pStopUtt[i] = pStopUtt[i+1]
5046          pDelayUtt[i] = pDelayUtt[i+1]
5047      }
5048  }
5049
5050  //Create utterance vectors and Set the utterance information.
5051  //This requires "reversing" the delay information since the calculated delay is the delay
5052  //of the reference signal, but we need the delay of the degraded signal.
5053  //The three vectors must be destroyed by the calling routine!
5054  void CTempAlignment::CreateUtteranceVectorsRef(int*  pNumUtterances, int** ppStartSampleUtterance,
      int** ppStopSampleUtterance, int** ppDelayUtterance, REPARSE_POINT* ReparsePoints, int
      NumReparsePoints)
5055  {
5056
5057  }
5058
5059  //Create utterance vectors and Set the utterance information.
5060  //This version does NOT reverse the delay information!.
5061  //The three vectors are allocated here, but must be destroyed by the calling routine!
5062  void CTempAlignment::CreateUtteranceVectorsDeg(int*  pNumUtterances, int** ppStartSampleUtterance,
      int** ppStopSampleUtterance, int** ppDelayUtterance, REPARSE_POINT* ReparsePoints, int
      NumReparsePoints, int* ActiveFrames, int ActiveFramesLen, int FrameSize)
5063  {
5064      int fdeg
5065
5066      int NumberOfUtterances = 1
5067      for (fdeg=1  fdeg<mNumMacroFrames  fdeg++)
5068          if (mpDelayInSamplesPerFrame[fdeg] != mpDelayInSamplesPerFrame[fdeg-1])
5069              NumberOfUtterances++
5070
5071
5072      int* pStartSampleUtterance = (int*)matMalloc((NumberOfUtterances+5) * sizeof(int))
5073      int* pStopSampleUtterance = (int*)matMalloc((NumberOfUtterances+5) * sizeof(int))
5074      int* pDelayUtterance = (int*)matMalloc((NumberOfUtterances+5) * sizeof(int))
5075
5076      int utt=0
5077
5078      NumberOfUtterances = 0
5079      pStartSampleUtterance[0] = 0
5080      pDelayUtterance[0] = mpDelayInSamplesPerFrame[0]
5081      for (fdeg=1  fdeg<mNumMacroFrames  fdeg++)
5082      {
5083          if (mpDelayInSamplesPerFrame[fdeg] != mpDelayInSamplesPerFrame[fdeg-1])
5084          {
5085              pStopSampleUtterance[NumberOfUtterances] = fdeg*mProcessData.mStepSize - 1
5086              NumberOfUtterances++
5087              pStartSampleUtterance[NumberOfUtterances] = fdeg*mProcessData.mStepSize
5088              pDelayUtterance[NumberOfUtterances] = mpDelayInSamplesPerFrame[fdeg]
5089          }
5090      }
5091      pStopSampleUtterance[NumberOfUtterances] = fdeg*mProcessData.mStepSize - 1
5092      NumberOfUtterances++
5093
5094
5095
5096      int LastUsedReparsePoint=0
5097
5098      for (utt=0  utt<NumberOfUtterances-1  utt++)
5099      {
5100          bool IsInsideActiveSection1 = true
5101          bool IsInsideActiveSection2 = true
5102          while(LastUsedReparsePoint<NumReparsePoints-1 &&
      pStartSampleUtterance[utt]>ReparsePoints[LastUsedReparsePoint].Deg.End)
5103              LastUsedReparsePoint++
5104          if (pStartSampleUtterance[utt]<ReparsePoints[LastUsedReparsePoint].Deg.Start &&
```

```
    pStopSampleUtterance[utt]<ReparsePoints[LastUsedReparsePoint].Deg.Start)
5105            IsInsideActiveSection1 = false
5106
5107        int ReparsePoint2 = LastUsedReparsePoint
5108        while(ReparsePoint2<NumReparsePoints-1 &&
    pStartSampleUtterance[utt+1]>ReparsePoints[ReparsePoint2].Deg.End)
5109            ReparsePoint2++
5110        if (pStartSampleUtterance[utt+1]<ReparsePoints[LastUsedReparsePoint].Deg.Start &&
    pStopSampleUtterance[utt+1]<ReparsePoints[ReparsePoint2].Deg.Start)
5111            IsInsideActiveSection2 = false
5112
5113        if (ReparsePoint2==LastUsedReparsePoint && (IsInsideActiveSection1 &&
    IsInsideActiveSection2) || (!IsInsideActiveSection1 && !IsInsideActiveSection2))
5114        {
5115            if ((!IsInsideActiveSection1 &&
5116                    abs(pDelayUtterance[utt]-pDelayUtterance[utt+1])<0.015*mProcessData.mSamplerate)
5117                ||  abs(pDelayUtterance[utt]-pDelayUtterance[utt+1])<0.0003*mProcessData.mSamplerate
5118                )
5119
5120            {
5121                int Delay
5122                int Len1 = pStopSampleUtterance[utt] - pStartSampleUtterance[utt]
5123                int Len2 = pStopSampleUtterance[utt+1] - pStartSampleUtterance[utt+1]
5124                if (Len2>Len1)
5125                    Delay = pDelayUtterance[utt+1]
5126                else
5127                    Delay = pDelayUtterance[utt]
5128
5129                CombineFirstTwoUtterancesDeg(NumberOfUtterances-utt, pStartSampleUtterance+utt,
    pStopSampleUtterance+utt, pDelayUtterance+utt)
5130                pDelayUtterance[utt] = Delay
5131
5132                NumberOfUtterances--
5133                utt--
5134            }
5135        }
5136    }
5137
5138
5139
5140    SECTION SecA, SecB
5141    int MaxLagSamples = (int)(0.001F*(float)mProcessData.mSamplerate)
5142    int NumSamples = mNumMacroFrames*mProcessData.mStepSize
5143    CFeatureVector RefSig
5144    mppSignals[0]->GetAsFeatureVector(&RefSig, 0)
5145    CFeatureVector DegSig
5146    mppSignals[1]->GetAsFeatureVector(&DegSig, 0)
5147    for (utt=0  utt<NumberOfUtterances  utt++)
5148    {
5149        SecA.Start = pStartSampleUtterance[utt]
5150        SecA.End = pStopSampleUtterance[utt]
5151        SecB.Start = pStartSampleUtterance[utt]-pDelayUtterance[utt]-MaxLagSamples
5152        SecB.End = pStopSampleUtterance[utt]-pDelayUtterance[utt]+MaxLagSamples
5153
5154        if (SecA.Len()<0.5*(float)mProcessData.mSamplerate)
5155            continue
5156        if (SecA.Start<0 || SecA.End>RefSig.mSize)
5157            continue
5158        if (SecB.Start<0 || SecB.End>DegSig.mSize)
5159            continue
5160
5161        int FrameNumStart = floor((float) (SecB.Start)/(float)FrameSize)
5162        int FrameNumEnd = floor((float) (SecB.End)/(float)FrameSize)
5163        int FrameNum = FrameNumStart
5164        bool ActiveSeg = false
5165        while ((!ActiveSeg) && (FrameNum<=FrameNumEnd) && (FrameNum>=FrameNumStart) &&
    (FrameNum<ActiveFramesLen))
5166        {
```

```
5167                ActiveSeg = ActiveSeg || ActiveFrames[FrameNum]
5168                FrameNum++
5169            }
5170            if (!ActiveSeg)
5171                continue
5172            int Delay = 0
5173            Delay = FindSectionAInSectionB(&SecA, &SecB, &RefSig, &DegSig, 0, 1, 1, 2*MaxLagSamples)
5174
5175            Delay = -(Delay -SecA.Start +SecB.Start)
5176            if (Delay!=pDelayUtterance[utt])
5177            {
5178
5179                pDelayUtterance[utt] = Delay
5180            }
5181        }
5182
5183        *pNumUtterances = NumberOfUtterances
5184        *ppStartSampleUtterance = pStartSampleUtterance
5185        *ppStopSampleUtterance = pStopSampleUtterance
5186        *ppDelayUtterance = pDelayUtterance
5187    }
5188
5189    }
5190
5191
```