```
    typedef double XFLOAT;
    typedef double OTA_FLOAT;

namespace SQFUNCS_POLQA_INTERNAL
{
using namespace std;

static int const FFTSIZE = 1024;

double matXMean(const double* in, int len)
{
    double mean= 0.0;
    for (int i=0; i < len; i++)
        mean += in[i];

    mean /= len;
    return mean;
}

SQSignal::SQSignal(XFLOAT const *fInputSignal,
                   long          lInputLen,
                   int           iInputRate,
                   int           iBitResolution,
                   MAT_HANDLE    inMatHandle,
                   FILE*         pLogFile)

: mBitResolution        (iBitResolution),
  mFrameSize            (-1),
  mOverlapFac           (-1.0f),
  mNrOfFrames           (-1),
  mSamplingFreq         (iInputRate),
  mOrigSampFreq         (iInputRate),
  mNrOfSamples          (-1),
  mStart                (-1),
  mEnd                  (-1),
  mNumPause             (-1),
  mPauseCenter          (NULL),
  mDCOffset             (0.0f),
  mCurrentASL           (0.0f),
  mUnalignedASL         (0.0f),
  mASLBeforeFilt        (0.0f),
  mCurNoiseLevel        (0.0f),
  mUnalignedNoiseLev    (0.0f),
  mNoiseLevBeforeFilt   (0.0f),
  mSpeechThreshold      (0.0f),
  mSpeechActivity       (-1.0f),
  mData                 (NULL),
  mEnv                  (NULL),
  mVADprofile           (NULL),
  mAvgSpeechSpec        (NULL),
  mAvgNoiseSpec         (NULL),
  mUsedFiltFreqResp     (NULL),
  mIsInvalidSignal      (false),
  mPreprocessed         (false),
  mFiltered             (false),
  mLevAligned           (false),
  matHandle             (inMatHandle),
  mpLogFile             (pLogFile)
{
    OPTTRY
    {
        if (FFTSIZE != (int)(FRAME_LEN * STD_SAMPLING_RATE))
            OPTTHROW ((string("ATTENTION: Value for FFTSIZE needs to changed to match
current configuration!")));

        if (fInputSignal   == NULL              ||
            lInputLen      <= 0                 ||
            iInputRate     <  MIN_SAMPLING_RATE ||
            iBitResolution <= 2)
            OPTTHROW(( string("Invalid input parameters.")));

        mNrOfSamples      = lInputLen;
        mData             = (XFLOAT*)matMalloc(mNrOfSamples * sizeof(XFLOAT));
        mUsedFiltFreqResp = (XFLOAT*)matMalloc(FILTERS_FREQRESP_LEN * sizeof(XFLOAT));
```

```
        mMaxAmplitude     = (((MAX_AMP_32BIT) < (pow(2.0f, mBitResolution-1))) ?
(MAX_AMP_32BIT) : (pow(2.0f, mBitResolution-1)));

        vmov(fInputSignal, mData, mNrOfSamples);

        matbSet(1.0f, mUsedFiltFreqResp, FILTERS_FREQRESP_LEN);

    }
    OPTCATCH((string errorMsg))
    {
        matFree(mData);
        matFree(mUsedFiltFreqResp);
        OPTTHROW ((string("ERROR in SQSignal::SQSignal(float*, long, int, int): " +
errorMsg + "\n")));
    }
}

SQSignal::SQSignal(SQSignal const   &Signal,
                   int               iInputRate,
                   int               iBitResolution)

: mBitResolution        (iBitResolution),
  mFrameSize            (-1),
  mOverlapFac           (-1.0f),
  mNrOfFrames           (-1),
  mSamplingFreq         (iInputRate),
  mOrigSampFreq         (iInputRate),
  mNrOfSamples          (-1),
  mStart                (-1),
  mEnd                  (-1),
  mNumPause             (-1),
  mPauseCenter          (NULL),
  mDCOffset             (0.0f),
  mCurrentASL           (0.0f),
  mUnalignedASL         (0.0f),
  mASLBeforeFilt        (0.0f),
  mCurNoiseLevel        (0.0f),
  mUnalignedNoiseLev    (0.0f),
  mNoiseLevBeforeFilt   (0.0f),
  mSpeechThreshold      (0.0f),
  mSpeechActivity       (-1.0f),
  mData                 (NULL),
  mEnv                  (NULL),
  mVADprofile           (NULL),
  mAvgSpeechSpec        (NULL),
  mAvgNoiseSpec         (NULL),
  mUsedFiltFreqResp     (NULL),
  mMaxAmplitude         (0.0f),
  mIsInvalidSignal      (false),
  mPreprocessed         (false),
  mFiltered             (false),
  mLevAligned           (false)
{
    OPTTRY
    {
        if (FFTSIZE != (int)(FRAME_LEN * STD_SAMPLING_RATE))
            OPTTHROW(( string("ATTENTION: Value for FFTSIZE needs to changed to match
current configuration!")));

        if (Signal.Data()       == NULL                 ||
            Signal.NrOfSamples()<= 0                    ||
            iInputRate          <  MIN_SAMPLING_RATE    ||
            iBitResolution      <= 2)
            OPTTHROW(( string("Invalid input parameters.")));

        mNrOfSamples      = Signal.NrOfSamples();
        mData             = (XFLOAT*)matMalloc(mNrOfSamples * sizeof(XFLOAT));
        mUsedFiltFreqResp = (XFLOAT*)matMalloc(FILTERS_FREQRESP_LEN * sizeof(XFLOAT));
        mMaxAmplitude     = (((MAX_AMP_32BIT) < (pow(2.0f, mBitResolution-1))) ?
(MAX_AMP_32BIT) : (pow(2.0f, mBitResolution-1)));
        mIsInvalidSignal  = Signal.IsInvalidSignal();
        matHandle         = Signal.matHandle;

        vmov(Signal.Data(), mData, mNrOfSamples);
        matbSet(1.0f, mUsedFiltFreqResp, FILTERS_FREQRESP_LEN);
    }
```

```
    OPTCATCH ((string errorMsg))
    {
        matFree(mData);
        matFree(mUsedFiltFreqResp);
        OPTTHROW ((string("ERROR in SQSignal::SQSignal(SQSignal, int, int): " +
errorMsg + "\n")));
    }
}
SQSignal::SQSignal(SQSignal const &Signal)

: mBitResolution        (-1),
  mFrameSize            (-1),
  mOverlapFac           (-1.0f),
  mNrOfFrames           (-1),
  mSamplingFreq         (-1),
  mOrigSampFreq         (-1),
  mNrOfSamples          (-1),
  mStart                (-1),
  mEnd                  (-1),
  mNumPause             (-1),
  mPauseCenter          (NULL),
  mDCOffset             (0.0),
  mCurrentASL           (0.0f),
  mUnalignedASL         (0.0f),
  mASLBeforeFilt        (0.0f),
  mCurNoiseLevel        (0.0f),
  mUnalignedNoiseLev    (0.0f),
  mNoiseLevBeforeFilt   (0.0f),
  mSpeechThreshold      (0.0f),
  mSpeechActivity       (-1.0f),
  mData                 (NULL),
  mEnv                  (NULL),
  mVADprofile           (NULL),
  mAvgSpeechSpec        (NULL),
  mAvgNoiseSpec         (NULL),
  mUsedFiltFreqResp     (NULL),
  mMaxAmplitude         (0.0),
  mIsInvalidSignal      (false),
  mPreprocessed         (false),
  mFiltered             (false),
  mLevAligned           (false)
{

    OPTTRY
    {
        if (FFTSIZE != (int)(FRAME_LEN * STD_SAMPLING_RATE))
            OPTTHROW(( string("ATTENTION: Value for FFTSIZE needs to changed to match
current configuration!")));

        if (Signal.Data()       == NULL ||
            Signal.NrOfSamples()<= 0    )
            OPTTHROW ((string("Invalid input parameters.")));

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

```
        mFiltered           = Signal.Filtered();
        mLevAligned         = Signal.LevelAligned();
        matHandle           = Signal.matHandle;
        mpLogFile           = Signal.mpLogFile;

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

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

        if (Signal.UsedFiltFreqResp() != NULL)
        {
            mUsedFiltFreqResp = (XFLOAT*)matMalloc(FILTERS_FREQRESP_LEN *
sizeof(XFLOAT));
            vmov(Signal.UsedFiltFreqResp(), mUsedFiltFreqResp, FILTERS_FREQRESP_LEN);
        }
        if (Signal.Env() != NULL)
        {
            mEnv = (XFLOAT*)matMalloc(mNrOfFrames * sizeof(XFLOAT));
            vmov(Signal.Env(), mEnv, mNrOfFrames);
        }
        if (Signal.VADprofile() != NULL)
        {
            mVADprofile = (short*)matMalloc(mNrOfFrames * sizeof(short));
            sivmov(Signal.VADprofile(), mVADprofile, mNrOfFrames);
        }
        if (Signal.AvgSpeechSpec() != NULL)
        {
            mAvgSpeechSpec = (XFLOAT*)matMalloc(FFTSIZE/2 * sizeof(XFLOAT));
            vmov (Signal.AvgSpeechSpec(), mAvgSpeechSpec, FFTSIZE/2);
        }
        if (Signal.AvgNoiseSpec() != NULL)
        {
            mAvgNoiseSpec = (XFLOAT*)matMalloc(FFTSIZE/2 * sizeof(XFLOAT));
            vmov (Signal.AvgNoiseSpec(), mAvgNoiseSpec, FFTSIZE/2);
        }
        if (Signal.PauseCenter() != NULL)
        {
            mPauseCenter = (long*)matMalloc(mNumPause * sizeof(long));
            ivmov (Signal.PauseCenter(), mPauseCenter, mNumPause);
        }
    }
    OPTCATCH ((string errorMsg))
    {
        if(mData)
            matFree(mData);
        if(mUsedFiltFreqResp)
            matFree(mUsedFiltFreqResp);
        if(mEnv)
            matFree(mEnv);
        if(mVADprofile)
            matFree(mVADprofile);
        if(mAvgSpeechSpec)
            matFree(mAvgSpeechSpec);
        if(mAvgNoiseSpec)
            matFree(mAvgNoiseSpec);
        if(mPauseCenter)
            matFree(mPauseCenter);
        OPTTHROW(( string("ERROR in SQSignal::SQSignal(SQSignal): " + errorMsg +
"\n")));
    }
}

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

```
        matFree(mUsedFiltFreqResp);
    if(mPauseCenter)
        matFree(mPauseCenter);

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

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

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

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

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

    this->mFrameSize            = -1;
    this->mOverlapFac           = -1;
    this->mNrOfFrames           = -1;
    this->mStart                = -1;
    this->mEnd                  = -1;
    this->mNumPause             = -1;
    this->mDCOffset             = -1;
    this->mCurrentASL           =  0;
    this->mUnalignedASL         =  0;
    this->mASLBeforeFilt        =  0;
    this->mCurNoiseLevel        =  0;
    this->mUnalignedNoiseLev    =  0;
    this->mNoiseLevBeforeFilt   =  0;
    this->mSpeechThreshold      =  0;
    this->mSpeechActivity       = -1;
    this->mIsInvalidSignal      = this->mIsInvalidSignal && other->IsInvalidSignal();
    this->mPreprocessed         = false;
    this->mFiltered             = false;
    this->mLevAligned           = false;

    if (mEnv           != NULL)
    {
        delete mEnv;
        mEnv = NULL;
    }
    if (mVADprofile    != NULL)
    {
        delete mVADprofile;
        mVADprofile = NULL;
    }
    if (mAvgSpeechSpec != NULL)
    {
        delete mAvgSpeechSpec;
        mAvgSpeechSpec = NULL;
    }
    if (mAvgNoiseSpec  != NULL)
    {
        delete mAvgNoiseSpec;
        mAvgNoiseSpec = NULL;
    }

    return *this;
}
```

```
void SQSignal::assign (SQSignal const *other)
{
    this->mNrOfSamples          = other->NrOfSamples();
    this->mBitResolution        = other->BitResolution();
    this->mMaxAmplitude         = other->MaxAmplitude();
    this->mSamplingFreq         = other->SamplingFreq();
    this->mOrigSampFreq         = other->OrigSampFreq();
    this->mFrameSize            = other->FrameSize();
    this->mOverlapFac           = other->OverlapFac();
    this->mNrOfFrames           = other->NrOfFrames();
    this->mStart                = other->Start();
    this->mEnd                  = other->End();
    this->mNumPause             = other->NumPause();
    this->mDCOffset             = other->DCOffset();
    this->mCurrentASL           = other->CurrentASL();
    this->mUnalignedASL         = other->UnalignedASL();
    this->mASLBeforeFilt        = other->ASLBeforeFilt();
    this->mCurNoiseLevel        = other->CurrentNoiseLevel();
    this->mUnalignedNoiseLev    = other->UnalignedNoiseLevel();
    this->mNoiseLevBeforeFilt   = other->NoiseLevelBeforeFilt();
    this->mSpeechThreshold      = other->SpeechThreshold();
    this->mSpeechActivity       = other->SpeechActivity();
    this->mIsInvalidSignal      = other->IsInvalidSignal();
    this->mPreprocessed         = other->Preprocessed();
    this->mFiltered             = other->Filtered();
    this->mLevAligned           = other->LevelAligned();

    if (this->mData             != NULL) matFree(mData);             mData = NULL;
    if (this->mEnv              != NULL) matFree(mEnv);              mEnv  = NULL;
    if (this->mVADprofile       != NULL) matFree(mVADprofile);       mVADprofile =
NULL;
    if (this->mAvgSpeechSpec    != NULL) matFree(mAvgSpeechSpec);    mAvgSpeechSpec =
NULL;
    if (this->mAvgNoiseSpec     != NULL) matFree(mAvgNoiseSpec);     mAvgNoiseSpec =
NULL;
    if (this->mUsedFiltFreqResp != NULL) matFree(mUsedFiltFreqResp); mUsedFiltFreqResp
= NULL;
    if (this->mPauseCenter      != NULL) matFree(mPauseCenter);      mPauseCenter =
NULL;

    int const FFTSIZE = (int)(FRAME_LEN * this->mSamplingFreq);

    if (other->Data() != NULL)
    {
        mData = (XFLOAT*)matMalloc(mNrOfSamples * sizeof(XFLOAT));
        vmov(other->Data(), mData, mNrOfSamples);
    }
    if (other->Env() != NULL)
    {
        mEnv = (XFLOAT*)matMalloc(mNrOfFrames * sizeof(XFLOAT));
        vmov(other->Env(), mEnv, mNrOfFrames);
    }
    if (other->VADprofile() != NULL)
    {
        mVADprofile = (short*)matMalloc(mNrOfFrames * sizeof(short));
        sivmov(other->VADprofile(), mVADprofile, mNrOfFrames);
    }
    if (other->AvgSpeechSpec() != NULL)
    {
        mAvgSpeechSpec = (XFLOAT*)matMalloc(FFTSIZE/2 * sizeof(XFLOAT));
        vmov (other->AvgSpeechSpec(), mAvgSpeechSpec, FFTSIZE/2);
    }
    if (other->AvgNoiseSpec() != NULL)
    {
        mAvgNoiseSpec = (XFLOAT*)matMalloc(FFTSIZE/2 * sizeof(XFLOAT));
        vmov (other->AvgNoiseSpec(), mAvgNoiseSpec, FFTSIZE/2);
    }
    if (other->UsedFiltFreqResp() != NULL)
    {
        mUsedFiltFreqResp = (XFLOAT*)matMalloc(FILTERS_FREQRESP_LEN * sizeof(XFLOAT));
        vmov (other->UsedFiltFreqResp(), mUsedFiltFreqResp, FILTERS_FREQRESP_LEN);
    }
    if (other->PauseCenter() != NULL)
    {
        mPauseCenter = (long*)matMalloc(mNumPause * sizeof(long));
```

```
        ivmov (other->PauseCenter(), mPauseCenter, mNumPause);
    }
}

void SQSignal::PreprocessingProperties (XFLOAT fEnvFrameLen,
                                        XFLOAT fEnvOverlapRatio,
                                        XFLOAT fEnvLowerLimdB,
                                        XFLOAT * const maxSigLenBuff)
{
    CalcEnvelope(fEnvFrameLen, fEnvOverlapRatio, fEnvLowerLimdB, maxSigLenBuff);
    CalcASLandNoiseLevel(FRAME_LEN, maxSigLenBuff);
    FindActBoundaries((((mCurNoiseLevel+(XFLOAT)6.0) < ((XFLOAT)-50.0)) ?
(mCurNoiseLevel+(XFLOAT)6.0) : ((XFLOAT)-50.0)), 4, (XFLOAT)0.008, 6, false);
    FindPauseCenter ((mSpeechThreshold + mCurrentASL) / 2);

    mPreprocessed = true;
    return;
}

void SQSignal::Preprocess(int   iTargetFs,
                          XFLOAT fTargetASLdBov,
                          XFLOAT fEnvFrameLen,
                          XFLOAT fEnvOverlapRatio,
                          XFLOAT fEnvLowerLimdB,
                          int   appType,
                          XFLOAT * const maxSigLenBuff,
                          int   IRSSendfilter,
                          FILE* pLogFile)
{
    OPTTRY
    {
        Resample(iTargetFs);
        RecalcDCOffset();
        vsadd(mData, -(mDCOffset * mMaxAmplitude / (XFLOAT)100.0), mData,
mNrOfSamples);

        CalcEnvelope(fEnvFrameLen, fEnvOverlapRatio, fEnvLowerLimdB, maxSigLenBuff);

        CalcASLandNoiseLevel(FRAME_LEN, maxSigLenBuff);

        mFiltered = true && !mIsInvalidSignal;

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

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

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

        long Offset = (long)(mDCOffset * (XFLOAT)1.0e9);
        mDCOffset = (XFLOAT)Offset / (XFLOAT)1.0e9;
        return mDCOffset;
    }
    OPTCATCH( (...))
    {
```

```
        OPTTHROW(( string("RecalcDCOffset failed.\n")));
    }
}

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

        mOverlapFac     = fOverlapRatio;
        mFrameSize      = (int)(mSamplingFreq * fFrameLen);
        mNrOfFrames     = (int)( (XFLOAT)mNrOfSamples / (mFrameSize*mOverlapFac) ) - 1;
        if (mNrOfFrames < 1)
        {
            mIsInvalidSignal = true;

            matFree(mEnv);
            mEnv = NULL;
            return;
        }
        if (mEnv == NULL)
            mEnv = (XFLOAT*)matMalloc(mNrOfFrames * sizeof(XFLOAT));
        else
        {
            matFree(mEnv);
            mEnv = (XFLOAT*)matMalloc(mNrOfFrames * sizeof(XFLOAT));
        }

        for (int i = 0; i < mNrOfFrames; i++)
            rmvesq(&mData[(int)(i*mOverlapFac*mFrameSize)], &mEnv[i], mFrameSize);

        XFLOAT rmsFloor = (((dBtoRMS(fLowerdBLim)) > ((XFLOAT)1e-16f)) ?
(dBtoRMS(fLowerdBLim)) : ((XFLOAT)1e-16f));
        if (maxSigLenBuff == NULL)
        {
            vsdiv(mEnv, mMaxAmplitude, mEnv, mNrOfFrames);
            matbThresh1(mEnv, mNrOfFrames, rmsFloor, MAT_LT);
            vlog10(mEnv, mEnv, mNrOfFrames);
            vsmul(mEnv, (XFLOAT)20.0, mEnv, mNrOfFrames);
            vclip(mEnv, -fLowerdBLim, mEnv, mNrOfFrames);
        }
        else
        {
            vsdiv (mEnv,            mMaxAmplitude, mEnv,          mNrOfFrames);
            matbThresh1(mEnv, mNrOfFrames, rmsFloor, MAT_LT);
            vlog10(mEnv,            maxSigLenBuff, mNrOfFrames);
            vsmul (maxSigLenBuff,   (XFLOAT)20.0,         mEnv,          mNrOfFrames);
            vclip (mEnv,            -fLowerdBLim,  mEnv,          mNrOfFrames);
        }

        return;
    }
    OPTCATCH( (string errorMsg))
    {
        OPTTHROW(( string("ERROR in SQSignal::CalcEnvelope(): " + errorMsg + "\n")));
    }
}

void SQSignal::FindActBoundaries(XFLOAT fMinActdB,
                                 int   iMinContActLen,
                                 XFLOAT fFrameLenInSec,
                                 int   iSafetyOffset,
                                 bool  doUseVADprofile)
{
    OPTTRY
    {
        if (mData == NULL)
```

```
            OPTTHROW(( string("No data in signal.")));

        if (mIsInvalidSignal)
        {

            return;
        }
        if (mBitResolution < 2 || mSamplingFreq < 1 || mNrOfSamples < 1)
            OPTTHROW(( string("Invalid signal parameters.")));

        if (doUseVADprofile && (mVADprofile == NULL || mNrOfFrames < 10))
            OPTTHROW(( string("No or invalid VAD profile. Call CalcASLandNoiseLevel()
first.")));

        if (!doUseVADprofile &&
            (fMinActdB >= 0.0f || iMinContActLen < 0 ||
            fFrameLenInSec < 0.0f || iSafetyOffset < 0))
            OPTTHROW(( string("Invalid input arguments.")));

        if (doUseVADprofile)
        {
            int i, start, end;

            for (i = 0; i < mNrOfFrames && mVADprofile[i] == SQ_VAD_NO_SPEECH; i++);
            if (i < mNrOfFrames) start = i;
            else                 start = -1;

            for (i = mNrOfFrames-1; i > start && mVADprofile[i] == SQ_VAD_NO_SPEECH;
i--);
            if (i > start)       end = i;
            else                 end = -1;

            if (start > 0 && end > start && end < mNrOfFrames)
            {
                mStart = ((((int) (start * mFrameSize * mOverlapFac)) > (0)) ? ((int)
(start * mFrameSize * mOverlapFac)) : (0));
                mEnd   = ((((int)ceil( end * mFrameSize * mOverlapFac)) <
(mNrOfSamples-1)) ? ((int)ceil( end * mFrameSize * mOverlapFac)) :
(mNrOfSamples-1));
                return;
            }
            else
            {

            }
        }

        XFLOAT const MIN_ACT_RMS     = pow((XFLOAT)10.0, fMinActdB / (XFLOAT)20.0) *
mMaxAmplitude;
        int const   FINE_FRAME_SIZE = (int)(fFrameLenInSec * mSamplingFreq);

        int         iContActCounter = 0;
        long        i;
        XFLOAT      fTemp;

        for (i = 0; i < mNrOfSamples - FINE_FRAME_SIZE; i += FINE_FRAME_SIZE)
        {
            rmvesq(&mData[i], &fTemp, FINE_FRAME_SIZE);
            if (fTemp > MIN_ACT_RMS)    iContActCounter++;
            else                        iContActCounter = 0;

            if (iContActCounter == iMinContActLen)
                break;
        }

        if (iContActCounter != iMinContActLen)
        {
            mStart = mEnd = -1;
            return;
        }

        mStart = i - (iContActCounter-1)*FINE_FRAME_SIZE;
        mStart = (((0) > (mStart - iSafetyOffset*FINE_FRAME_SIZE)) ? (0) : (mStart -
iSafetyOffset*FINE_FRAME_SIZE));

        iContActCounter = 0;
```

```
        for (i = mNrOfSamples-FINE_FRAME_SIZE; i > mStart+FINE_FRAME_SIZE; i -=
FINE_FRAME_SIZE)
        {
            rmvesq(&mData[i], &fTemp, FINE_FRAME_SIZE);
            if (fTemp > MIN_ACT_RMS)    iContActCounter++;
            else                        iContActCounter = 0;

            if (iContActCounter == iMinContActLen)
                break;
        }

        if (iContActCounter != iMinContActLen)
        {
            mStart = mEnd = -1;
            return;
        }

        mEnd = i + iContActCounter*FINE_FRAME_SIZE - 1;
        mEnd = (((mNrOfSamples-1) < (mEnd + iSafetyOffset*FINE_FRAME_SIZE)) ?
(mNrOfSamples-1) : (mEnd + iSafetyOffset*FINE_FRAME_SIZE));

        return;
    }
    OPTCATCH( (string errorMsg))
    {
        OPTTHROW(( string("ERROR in SQSignal::FindActBoundaries(): " + errorMsg +
"\n")));
    }
}

void SQSignal::FindPauseCenter (XFLOAT fMinActivitydB)
{
    OPTTRY
    {
        if (mIsInvalidSignal)
        {

            return;
        }
        if (mData == NULL || mEnv == NULL)
            OPTTHROW(( string("Invalid input data.")));

        if(mPauseCenter)
            matFree(mPauseCenter);
        mPauseCenter = NULL;

        XFLOAT const MIN_PAUSE_LENGTH_S = 1.0;
        int   const MAX_NUM_PAUSE = MAX_NUM_SENTENCES - 1;

        bool inPause = false;
        int  pauseStart = 0;
        int  pauseLength = 0;
        int  numPause = 0;
        int  centerFrame[MAX_NUM_PAUSE] = {0};
        int  firstFrame, lastFrame;

        mStart > 0 ? firstFrame = (int)((XFLOAT)mStart / (mFrameSize * mOverlapFac)) :
firstFrame = 0;
        mEnd   > 0 ? lastFrame  = (int)((XFLOAT)mEnd   / (mFrameSize * mOverlapFac)) :
lastFrame  = mNrOfFrames;
        firstFrame = limit(firstFrame, 0, (int)mNrOfFrames-1);
        lastFrame  = limit(lastFrame,  0, (int)mNrOfFrames);

        for (int f = firstFrame; f < lastFrame; f++)
        {
            if (inPause == true)
            {
                if (mEnv[f] > fMinActivitydB)
                {
                    inPause = false;
                    pauseLength = f - pauseStart;
                    if (pauseLength > MIN_PAUSE_LENGTH_S * mSamplingFreq / (mFrameSize
* mOverlapFac))
                    {
                        centerFrame[numPause] = pauseStart + (int)(0.5f*pauseLength);
                        numPause ++;
```

```
                        if (numPause >= MAX_NUM_PAUSE)
                            break;
                    }
                }
            }
            else
            {
                if (mEnv[f] <= fMinActivitydB)
                {
                    inPause = true;
                    pauseStart = f;
                }
            }
        }

        mNumPause = numPause;
        if (mNumPause)
        {
            mPauseCenter = (long*)matMalloc(mNumPause * sizeof(long));
            for (int p = 0; p < mNumPause; p++)
            {
                if (centerFrame[p] > 0)
                {
                    mPauseCenter[p] = round (centerFrame[p] * mFrameSize *
mOverlapFac);
                }
                else
                    OPTTHROW(( string ("Pause at position 0")));
            }
        }
    }
    OPTCATCH( (string errorMsg))
    {
        OPTTHROW(( string("ERROR in SQSignal::FindPauseCenter(): " + errorMsg +
"\n")));
    }
    OPTCATCH( (...))
    {
        OPTTHROW(( string("ERROR in SQSignal::FindPauseCenter(): Unknown error.\n")));
    }
}

void SQSignal::CurrentSentence (int iSample, long *lStart, long *lEnd, short *sIndex)
{
    *sIndex = -1;
    for (short s = 0; s < mNumPause; s++)
    {
        if (iSample < mPauseCenter[s])
        {
            *sIndex = s;
            s == 0 ? *lStart = 0 : *lStart = mPauseCenter[s-1];
            *lEnd = mPauseCenter[s];
            break;
        }
    }
    if (*sIndex == -1)
    {
        *sIndex = mNumPause;
        *lStart = mPauseCenter[mNumPause];
        *lEnd   = mNrOfSamples;
    }
}

void SQSignal::CurrentSentence (int iIdx, long *lStart, long *lEnd) const
{
    if (mNumPause < 0)
        OPTTHROW(( string ("Pauses not yet calculated.")));

    if (iIdx == 0)
        *lStart = 0;
    else
        *lStart = mPauseCenter[iIdx-1];

    if (iIdx == mNumPause)
        *lEnd = mNrOfSamples;
    else
```

```
        *lEnd = mPauseCenter[iIdx];

    if (*lStart < 0 || *lEnd < 0 || *lStart > mNrOfSamples || *lEnd > mNrOfSamples)
    {
        OPTTHROW(( string ("Pauses invalid.")));
    }
}

void SQSignal::CalcASLandNoiseLevel(XFLOAT fFrameLen, XFLOAT * const maxSigLenBuff)
{
OPTTRY
    {
        if (mIsInvalidSignal)
        {

            return;
        }
        if (mBitResolution < 2 || mSamplingFreq < 1 || mData == NULL)
            OPTTHROW(( string("Invalid input parameters.\n")));
        if (mNrOfSamples < 10*mSamplingFreq*fFrameLen*FRAME_OVERLAP_RATIO)
        {
            mIsInvalidSignal = true;
            return;
        }

        int newFrameSize = (int)(fFrameLen * mSamplingFreq);
        if (mEnv == NULL || mFrameSize != newFrameSize || mOverlapFac !=
FRAME_OVERLAP_RATIO)
            CalcEnvelope(fFrameLen, FRAME_OVERLAP_RATIO, MIN_LEVEL_DB, maxSigLenBuff);

        if (mVADprofile == NULL)
            mVADprofile = (short*)matMalloc(mNrOfFrames * sizeof(short));
        else
        {
            matFree(mVADprofile);
            mVADprofile = (short*)matMalloc(mNrOfFrames * sizeof(short));
        }

        if (SQcalcASLandNoise(mEnv,
                              mVADprofile,
                              mNrOfFrames,
                              (int)(mFrameSize*mOverlapFac),
                              MIN_LEVEL_DB,
                              mSamplingFreq,
                              &mSpeechActivity,
                              &mCurrentASL,
                              &mCurNoiseLevel,
                              &mSpeechThreshold)
            != SQ_NO_ERRORS)
        {

            rmvesq(mData, &mCurrentASL, mNrOfSamples);
            mCurrentASL = 20.0f * log10((mCurrentASL+1e-16f) / (mMaxAmplitude+1e-16f));
            mCurrentASL = mCurNoiseLevel = (((mCurrentASL) > (MIN_LEVEL_DB)) ?
(mCurrentASL) : (MIN_LEVEL_DB));
            mIsInvalidSignal = true;
        }

        long x = (long)((XFLOAT)1.0e7*((((-200)) > (mCurrentASL)) ? ((-200)) :
(mCurrentASL)));
        mCurrentASL = (XFLOAT)x / (XFLOAT)1.0e7;

        x = (long)((XFLOAT)1.0e7*((((-200)) > (mCurNoiseLevel)) ? ((-200)) :
(mCurNoiseLevel)));
        mCurNoiseLevel = (XFLOAT)x / (XFLOAT)1.0e7;

        x = (long)((XFLOAT)1.0e7*((((-200)) > (mSpeechThreshold)) ? ((-200)) :
(mSpeechThreshold)));
        mSpeechThreshold = (XFLOAT)x / (XFLOAT)1.0e7;

        if (!mFiltered)
        {
            mASLBeforeFilt      = mCurrentASL;
            mNoiseLevBeforeFilt = mCurNoiseLevel;
        }
        if (!mLevAligned)
```

```
        {
            mUnalignedASL      = mCurrentASL;
            mUnalignedNoiseLev = mCurNoiseLevel;
        }
    }
    OPTCATCH( (string errorMsg))
    {
        OPTTHROW(( string("ERROR in SQSignal::CalcASLandNoiseLevel(): " + errorMsg +
"\n")));
    }
    OPTCATCH( (...))
    {
        OPTTHROW(( string("ERROR in SQSignal::CalcASLandNoiseLevel(): Unknown
error.\n")));
    }
}

void SQSignal::LevelAlign(XFLOAT fTargetASLdBov)
{
    OPTTRY
    {
        if (mIsInvalidSignal)
        {

            return;
        }
        if (mCurrentASL > 0.0f || fTargetASLdBov > 0.0f || mData == NULL ||
mNrOfSamples <= 0)
        OPTTHROW(( string("Invalid signal data or speech level. Did you forget to call
CalcASLandNoiseLevel() before?")));

        XFLOAT fLevCorrFac = dBtoRMS(fTargetASLdBov - mCurrentASL);
        vsmul(mData, fLevCorrFac, mData, mNrOfSamples);

        if (mEnv           != NULL)
            vsadd(mEnv,           fTargetASLdBov - mCurrentASL,        mEnv,
mNrOfFrames);
        if (mAvgSpeechSpec != NULL)
            vsmul(mAvgSpeechSpec, dBtoPow(fTargetASLdBov-mCurrentASL), mAvgSpeechSpec,
FFTSIZE/2);
        if (mAvgNoiseSpec  != NULL)
            vsmul(mAvgNoiseSpec,  dBtoPow(fTargetASLdBov-mCurrentASL), mAvgNoiseSpec,
FFTSIZE/2);

        mSpeechThreshold += fTargetASLdBov - mCurrentASL;
        mCurNoiseLevel   += fTargetASLdBov - mCurrentASL;
        mCurrentASL       = fTargetASLdBov;
        mLevAligned       = true && !mIsInvalidSignal;
    }
    OPTCATCH( (string errorMsg))
    {
        OPTTHROW(( string("ERROR in SQSignal::LevelAlign(): " + errorMsg + "\n")));
    }
}

void SQSignal::SetSamplingFrequency(int iFrequency)
{
    mSamplingFreq = iFrequency;
}

int SQSignal::Resample(int iTargetFs)
{
    if (mData == NULL || mNrOfSamples <= 1 || mSamplingFreq < 1 || iTargetFs < 1)
    {

        return SQERR_RESAMPLING;
    }

    int iRetVal = SQ_NO_ERRORS;

    if (iTargetFs == mSamplingFreq)
        return iRetVal;

    OPTTRY
    {
        unsigned long newNrOfSamples;
```

```
        XFLOAT ResamplingFactor = (XFLOAT)iTargetFs/(XFLOAT)mSamplingFreq;
        int newBufferLength = (int)(ceil(ResamplingFactor*1000)*mNrOfSamples/1000);
        XFLOAT *resampledSignal = (XFLOAT*)matMalloc(newBufferLength * sizeof(XFLOAT));

        iRetVal = matConvertSamplerate(mData, mNrOfSamples, resampledSignal,
newBufferLength, ResamplingFactor, &newNrOfSamples);

        mNrOfSamples = newNrOfSamples;
        matFree(mData);
        mData = (XFLOAT*)matMalloc(mNrOfSamples * sizeof(XFLOAT));
        matbCopy(resampledSignal, mData, newNrOfSamples);
        matFree(resampledSignal);
        resampledSignal = NULL;
        mSamplingFreq = iTargetFs;
    }
    OPTCATCH((string errorMsg))
    {
        errorMsg += "ERROR in SQSignal::Resample():\n";

        iRetVal = SQERR_RESAMPLING;
    }

    return iRetVal;
}

void SQSignal::SlidingWinMeanRemoval(int WINLEN)
{
    if (mData == NULL || mNrOfSamples < 1 || WINLEN < 3 || mNrOfSamples < WINLEN+3)
        OPTTHROW(( string("ERROR in SlidingWinMeanRemoval: Invalid input
arguments.\n")));
    if (mIsInvalidSignal)
    {

        return;
    }

    if ((WINLEN | 0x1) != WINLEN)
        WINLEN--;

    XFLOAT *buff = NULL;
    OPTTRY
    {

        buff = (XFLOAT*)matMalloc(WINLEN/2 * sizeof(XFLOAT));
        matbZero(buff, WINLEN/2);

        XFLOAT windowSum = (XFLOAT)0.0;
        windowSum = matSum(mData, WINLEN/2);

        int   rightWinPos = WINLEN/2;
        int   bufferPos   = 0;
        int   curPos      = 0;
        XFLOAT newVal;
        for (; curPos < mNrOfSamples - WINLEN/2; curPos++, rightWinPos++)
        {
            newVal = mData[curPos];
            mData[curPos] -= windowSum/WINLEN;

            windowSum += mData[rightWinPos] - buff[bufferPos];
            buff[bufferPos] = newVal;
            bufferPos = (bufferPos+1) % (WINLEN/2);
        }

        for (; curPos < mNrOfSamples; curPos++)
        {
            mData[curPos] -= windowSum/WINLEN;

            windowSum -= buff[bufferPos];
            bufferPos = (bufferPos+1) % (WINLEN/2);
        }

        matFree(buff);
        buff = NULL;
    }
    OPTCATCH( (...))
```

```
    {
        matFree(buff);
        buff = NULL;
        OPTTHROW(( string("ERROR in SlidingWinMeanRemoval: Unknown error.\n")));
    }
}

XFLOAT const* SQSignal::Data() const
{
    return mData;
}

XFLOAT* SQSignal::Env() const
{
    return mEnv;
}

XFLOAT SQSignal::EnvLevel(int samplePos) const
{
    if (mEnv == NULL || mNrOfFrames <= 0 || samplePos < 0)
        OPTTHROW(-1);

    int safeFramePos =
        round(( samplePos - mFrameSize*(1-mOverlapFac) ) / ( mFrameSize*mOverlapFac ));
    safeFramePos = limit(safeFramePos, 0, (int)mNrOfFrames-1);

    return mEnv[safeFramePos];
}

int SQSignal::BitResolution() const
{
    return mBitResolution;
}

short const* SQSignal::VADprofile() const
{
    return mVADprofile;
}

long SQSignal::SamplePosToFrameNum(int samplePos) const
{
    if (mNrOfFrames <= 0 || samplePos < 0)
        OPTTHROW(-1);

    int safeFramePos =
        round(( samplePos - mFrameSize*(1-mOverlapFac) ) / ( mFrameSize*mOverlapFac ));
    safeFramePos = limit(safeFramePos, 0, (int)mNrOfFrames-1);

    return safeFramePos;
}

XFLOAT const * SQSignal::AvgSpeechSpec() const
{
    return mAvgSpeechSpec;
}

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

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

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

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

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

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

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

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

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

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

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

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

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

    return mPauseCenter[idx];
}

XFLOAT SQSignal::DCOffset() const
{
    return mDCOffset;
}

XFLOAT SQSignal::CurrentASL() const
{
    return mCurrentASL;
}

XFLOAT SQSignal::UnalignedASL() const
{
    return mUnalignedASL;
}

XFLOAT SQSignal::ASLBeforeFilt() const
{
    return mASLBeforeFilt;
}

XFLOAT SQSignal::CurrentNoiseLevel() const
{
    return mCurNoiseLevel;
}

XFLOAT SQSignal::UnalignedNoiseLevel() const
{
    return mUnalignedNoiseLev;
```

```
}

XFLOAT SQSignal::NoiseLevelBeforeFilt() const
{
    return mNoiseLevBeforeFilt;
}

XFLOAT SQSignal::SpeechActivity () const
{
    return mSpeechActivity;
}

XFLOAT SQSignal::SpeechThreshold () const
{
    return mSpeechThreshold;
}

XFLOAT SQSignal::MaxAmplitude () const
{
    return mMaxAmplitude;
}

bool SQSignal::IsInvalidSignal() const
{
    return mIsInvalidSignal;
}

bool SQSignal::Preprocessed() const
{
    return mPreprocessed;
}

bool SQSignal::Filtered() const
{
    return mFiltered;
}

bool SQSignal::LevelAligned() const
{
    return mLevAligned;
}

void SQSignal::SetNumPause(int num)
{
    mNumPause = num;
}

void SQSignal::SetPauseCenter(long *origPauseCenter, int originalFreq)
{
    if(mPauseCenter)
        matFree(mPauseCenter);
    mPauseCenter = NULL;

    if (mNumPause > 0)
    {
        mPauseCenter = (long*)matMalloc(mNumPause * sizeof(long));
        lCopyVector (origPauseCenter, (long)mNumPause, mPauseCenter);
        lScaleVector (mPauseCenter, (long)mNumPause, (XFLOAT)mSamplingFreq /
originalFreq, mPauseCenter);
    }
}

}
```