```


    typedef double XFLOAT
    typedef double OTA_FLOAT


{

extern XFLOAT gStandardIRSdB [31][2]
extern XFLOAT gModifiedIRSdB [31][2]
extern XFLOAT gWidebandHeadphone [31][2]
extern XFLOAT gNarrowbandHeadphone [31][2]

XFLOAT gIdealGeneral [20][2] = {{0.,0.0},
                                {50., 0.0},
                                {70., 0.0},
                                {100.,0.0},
                                {150., 0.0},
                                {220., 0.0},
                                {350., 0.0},
                                {500., 0.0},
                                {750., 0.0},
                                {1200.,0.0},
                                {1800., 0.0},
                                {2700., 0.0},
                                {4000., 0.0},
                                {6000., 0.0},
                                {8000., 0.0},
                                {10000., 0.0},
                                {12000., 0.0},
                                {16000., 0.0},
                                {20000., 0.0},
                                {24000., 0.0}}

XFLOAT gIdealIrs [20][2] = {{0.,-20.0},
                                {50., 20.0},
                                {70., 52.0},
                                {100., 70.0},
                                {150., 80.0},
                                {220., 83.0},
                                {350., 88.0},
                                {500., 89.0},
                                {750., 90.0},
                                {1200., 91.0},
                                {1800., 92.0},
                                {2700., 93.0},
                                {4000., 75.0},
                                {6000., 20.0},
                                {8000., -100.0},
                                {10000., -100.0},
                                {12000., -100.0},
                                {16000., -200.0},
                                {20000., -200.0},
                                {24000., -200.0}}

XFLOAT gIdealWide [20][2] = {{0.,-20.0},
                                {50., 60.0},
                                {70., 70.0},
                                {100., 80.0},
                                {150., 86.0},
                                {220., 87.0},
                                {350., 88.0},
                                {500., 89.0},
                                {750., 90.0},
                                {1200., 91.0},
                                {1800., 92.0},
                                {2700., 93.0},
                                {4000., 84.0},
```

```
                                    {6000., 72.0},
                                    {8000., 60.0},
                                    {10000., 10.0},
                                    {12000., -10.0},
                                    {16000., -50.0},
                                    {20000., -100.0},
                                    {24000., -200.0}}

void MakeTable (XFLOAT          pFactorFilter [][2],
                CDoubleArray    &pFactorFrame0,
                XFLOAT          pFrequencyResolution,
                int             &pNumberOfPoints)
{
    pNumberOfPoints = 0

    for(int bandIndex = 0  bandIndex < pFactorFrame0.GetSize ()  bandIndex++)
    {
        pFactorFilter [bandIndex][0] = pFrequencyResolution * bandIndex
        pFactorFilter [bandIndex][1] = dB10 (pFactorFrame0.m_pData[bandIndex])
        pNumberOfPoints++
    }
}

void CPairParameters::DetermineCalibrationFactors()
{
    CTimeSeries     calibrationSignal
    CHzSpectrum     hzSpectrum
    CBarkSpectrum   pitchPowerDensity
    CBarkSpectrum   smearedPitchPowerDensity
    CBarkSpectrum   LoudnessDensity
    XFLOAT          peak, totalSone
    XFLOAT          periodInSamples, numberOfPeriodsPerFrame, omega
    XFLOAT          calibrationFactorSp, calibrationFactorSl
    XFLOAT          peak1
    CNewStdString   s

    statics->setACalibrationFactorSl(1)
    statics->setACalibrationFactorSp(1)

    calibrationSignal.Initialize("calibrationSignal", POLQAHandle)
    hzSpectrum.Initialize("hzSpectrum", POLQAHandle)
    pitchPowerDensity.Initialize("pitchPowerDensity", POLQAHandle)
    smearedPitchPowerDensity.Initialize("smearedPitchPowerDensity", POLQAHandle)
    LoudnessDensity.Initialize("LoudnessDensity", POLQAHandle)

    periodInSamples = (XFLOAT)statics->sampleRate / (XFLOAT) 1000.
    numberOfPeriodsPerFrame = aTransformLength / periodInSamples
    numberOfPeriodsPerFrame = round(numberOfPeriodsPerFrame)
    periodInSamples = aTransformLength / numberOfPeriodsPerFrame
    omega = 2.0 * PI / periodInSamples

    calibrationSignal.SetToSine((XFLOAT) 29.54, (XFLOAT) omega)
    hzSpectrum.STFTPowerSpectrumOf(POLQAHandle, calibrationSignal, 0, 0)

    peak1 =  hzSpectrum. Maximum (0)
    ASSERT (peak1 > 1e-10)

    pitchPowerDensity. FrequencyWarpingOf (POLQAHandle, hzSpectrum, 1.0)

    peak = pitchPowerDensity. Maximum (0)

    ASSERT (peak > 1e-10)

    calibrationFactorSp = (XFLOAT) (10000./ peak)
    statics->setACalibrationFactorSp(calibrationFactorSp)

    pitchPowerDensity. MultiplyWith (0, calibrationFactorSp)

```

```
137     bool* UseThisFrame = new bool[statics->stopFrameIdx+1]
138     for (int frameIndex = 0  frameIndex <= statics->stopFrameIdx  frameIndex++) {
139         UseThisFrame[frameIndex] = TRUE
140     }
141
142     smearedPitchPowerDensity. ExcitationOf (POLQAHandle, pitchPowerDensity, UseThisFrame,
    statics->listeningCondition)
143     delete[] UseThisFrame
144
145     LoudnessDensity. IntensityWarpingOf (POLQAHandle, smearedPitchPowerDensity)
146
147     totalSone = LoudnessDensity. Integral (POLQAHandle, 0)
148
149     calibrationFactorSl = (XFLOAT) (1.0/ totalSone)
150     statics->setACalibrationFactorSl(calibrationFactorSl)
151
152     LoudnessDensity. MultiplyWith (0, calibrationFactorSl)
153 }
154
155 int CPairParameters::GetTransformLength ()
156 {
157     if (aTransformLength == 0)
158     {
159
160         if (statics->sampleRate / 1000 <= 9.0) { aTransformLength = 256  }
161         else if (statics->sampleRate / 1000 <= 18.0) { aTransformLength = 512  }
162         else if (statics->sampleRate / 1000 <= 36.0) { aTransformLength = 1024  }
163         else { aTransformLength = 2048  }
164     }
165     return aTransformLength
166 }
167
168 void CPairParameters::IdealizationProcess(POLQA_RESULT_DATA* pDisturbanceOverviewHolder)
169 {
170
171     CNewStdString s
172
173     aTransformLength = GetTransformLength()
174     const XFLOAT OverlapCoeff = 0.75
175
176     const int backupNrFrames = statics->nrFrames
177
178     statics->setNrFrames(1)
179     statics->setStartFrameIndex(0)
180     statics->setStopFrameIndex(0)
181
182     DetermineCalibrationFactors()
183
184     int         samplesToSkipAtStartOfOriginalFile, samplesToSkipAtEndOfOriginalFile
185
186     XFLOAT      sumOf5Samples
187     int         CRITERIUM_FOR_SILENCE_OF_5_SAMPLES, i
188
189     CRITERIUM_FOR_SILENCE_OF_5_SAMPLES = (int)2.0E3
190
191     const int CTimeSeriesLength = statics->nrTimesSamples
192     samplesToSkipAtStartOfOriginalFile = 0
193     do
194     {
195         sumOf5Samples = (XFLOAT) 0
196         for (i = 0  i < 5  i++)
197         {
198             sumOf5Samples += (XFLOAT) fabs
    (aOriginalTimeSeries.m_pData[samplesToSkipAtStartOfOriginalFile + i])
199         }
200         if (sumOf5Samples < CRITERIUM_FOR_SILENCE_OF_5_SAMPLES)
201         {
202             samplesToSkipAtStartOfOriginalFile++
```

```
        }
    } while ((sumOf5Samples < CRITERIUM_FOR_SILENCE_OF_5_SAMPLES)
                && (samplesToSkipAtStartOfOriginalFile < CTimeSeriesLength / 2))

    samplesToSkipAtEndOfOriginalFile = 0
    do
    {
        sumOf5Samples = (XFLOAT) 0
        for (i = 0  i < 5  i++)
        {
            sumOf5Samples += (XFLOAT) fabs (aOriginalTimeSeries.m_pData[CTimeSeriesLength - 1 -
samplesToSkipAtEndOfOriginalFile - i])
        }
        if (sumOf5Samples < CRITERIUM_FOR_SILENCE_OF_5_SAMPLES)
        {
            samplesToSkipAtEndOfOriginalFile++
        }
    } while ((sumOf5Samples < CRITERIUM_FOR_SILENCE_OF_5_SAMPLES)
                && (samplesToSkipAtEndOfOriginalFile < CTimeSeriesLength / 2))

    statics->setNrFrames(backupNrFrames)

    const int StartSampleRef = samplesToSkipAtStartOfOriginalFile
    const int DelayOfStartFrame =
aDelayUtterance.m_pData[GetUtteranceForSample(aStartSampleUtterance, aStopSampleUtterance,
aDelayUtterance, StartSampleRef)]

    const int StopSampleRef = CTimeSeriesLength - samplesToSkipAtEndOfOriginalFile
    const int DelayOfStopFrame = aDelayUtterance.m_pData[GetUtteranceForSample(aStartSampleUtterance,
aStopSampleUtterance, aDelayUtterance, StopSampleRef)]

    const int StartSampleDeg = StartSampleRef - DelayOfStartFrame
    const int StopSampleDeg = StopSampleRef - DelayOfStopFrame

    const int StartFrameDeg = (int)max(0.0, StartSampleDeg / ((1.0-OverlapCoeff)*aTransformLength))
    const int StopFrameDeg = (int)min(double(mMaxModelFrames - 1.0), min(double(backupNrFrames -
1.0), (StopSampleDeg / ((1.0 - OverlapCoeff)*aTransformLength)) - 1.0))

    statics->setStartFrameIndex(StartFrameDeg)
    statics->setStopFrameIndex(StopFrameDeg)

    CHzSpectrum      originalHzPowerSpectrum

    originalHzPowerSpectrum.Initialize("Idealization : originalHzPowerSpectrum", POLQAHandle)

    originalHzPowerSpectrum.STFTPowerSpectrumOf (POLQAHandle, aOriginalTimeSeries,
aStartSampleUtterance, aStopSampleUtterance, aDelayUtterance, false, false)

    CDoubleArray idealHzSpectrumAvg
    idealHzSpectrumAvg.Initialize("idealHzSpectrumAvg", statics->aNumberOfHzBands)

    switch (statics->listeningCondition)
    {
    case STANDARD_IRS:
        idealHzSpectrumAvg.InvDb2 (POLQAHandle, gStandardIRSdB, 31, gIdealIrs, 20)
        break
    case MODIFIED_IRS:
        idealHzSpectrumAvg.InvDb2 (POLQAHandle, gModifiedIRSdB, 31, gIdealIrs, 20)
        break
    case WIDE_H:
        idealHzSpectrumAvg.InvDb2 (POLQAHandle, gWidebandHeadphone, 31, gIdealWide, 20)
        break
    case NARROW_H:
        idealHzSpectrumAvg.InvDb2 (POLQAHandle, gStandardIRSdB, 31, gIdealIrs, 20)
        break
    }

    CDoubleArray originalHzPowerSpectrumAvg
```

```
    originalHzPowerSpectrumAvg.Initialize("originalHzPowerSpectrumAvg", statics->aNumberOfHzBands)

    originalHzPowerSpectrumAvg. TimeAvgOf(POLQAHandle, originalHzPowerSpectrum)

    XFLOAT originalHzPowerSpectrumAvgSum = originalHzPowerSpectrumAvg.PowerInBand(POLQAHandle, 200,
3500)
    XFLOAT idealHzSpectrumAvgSum = idealHzSpectrumAvg.PowerInBand(POLQAHandle, 200, 3500)
    XFLOAT gain = originalHzPowerSpectrumAvgSum / idealHzSpectrumAvgSum

    idealHzSpectrumAvg *= gain

    CDoubleArray ratio
    ratio.Initialize ("ratio", statics->aNumberOfHzBands)

    ratio.RatioOf (idealHzSpectrumAvg, originalHzPowerSpectrumAvg, 1.0E-12)

    ratio.m_pData[0] = 1e-12f

    CDoubleArray    factor
    factor.Initialize("factor", statics->aNumberOfHzBands)

    XFLOAT compressionPower = 0.0
    switch (aListeningCondition)
    {

        default:       compressionPower = 0.0  break

    }

    if (compressionPower != 0.0)
    {
        factor.CompressOf (ratio, compressionPower)

        XFLOAT factorFilter [2048][2]
        int    numberOfPoints

        MakeTable (factorFilter, factor, statics->aFrequencyResolutionHz, numberOfPoints)


        aOriginalTimeSeries. FilterWith (POLQAHandle, FALSE,
                                         statics->sampleRate,
                                         factorFilter,
                                         numberOfPoints,
                                         aOriginalTimeSeries,
                                         aOriginalTimeSeriesFFT,
                                         aOriginalTimeSeriesFilteredFFT)

    }
}

}
```