```
    typedef double XFLOAT;
    typedef double OTA_FLOAT;

namespace POLQAV2
{

void CPairParameters::Free(POLQA_HANDLE *POLQAHandle)
{
    (*POLQAHandle)->sqStorage->ClearAll();

    if(*POLQAHandle)
    {
        CPOLQAData **POLQAData = (CPOLQAData**)POLQAHandle;
        FreeResultData(&(*POLQAData)->pResults);
        delete (*POLQAData);
        *POLQAHandle = 0;
    }
    aResultsFile.Close();
}

void AddProcessingTime(POLQA_RESULT_DATA *pRes, CNewStdString ProcessingStep, double
TimeMs, long Cycles)
{
    pRes->m_ClockCycles[pRes->m_NumProcessingTimes] = Cycles;
    pRes->m_ProcessingTime[pRes->m_NumProcessingTimes] = TimeMs;
    pRes->m_ProcessingStep[pRes->m_NumProcessingTimes] = ProcessingStep;
    pRes->m_NumProcessingTimes++;
}

POLQA_RESULT_DATA* CPairParameters::GetResults(POLQA_HANDLE POLQAHandle)
{
    if (POLQAHandle)
    {
        CPOLQAData *POLQAData = (CPOLQAData*)POLQAHandle;
        return POLQAData->pResults;
    }
    else return 0;
}

bool CPairParameters::AdjustSampleRates(XFLOAT* pSamplesIn, long NumSamplesIn, XFLOAT*
pSamplesOut, long MaxSigLen, long* pNumSamplesOut)
{
    bool rc=false;
    int Err;
    XFLOAT Rate;
    if (aSampleFrequencyHz>8000.0)
    {
        unsigned long NumSamplesOut = *pNumSamplesOut;
        switch (aListeningCondition)
        {
            case NARROW_H:
            case STANDARD_IRS:
                Rate = 8000.0/aSampleFrequencyHz;
                Err=matConvertSamplerate(pSamplesIn, (unsigned long)NumSamplesIn,
pSamplesOut, (unsigned long)MaxSigLen, Rate, &NumSamplesOut);
                rc = true;
                break;
            case WIDE_H:
                if (aSampleFrequencyHz<48000.0)
                {
                    Rate = 48000.0/aSampleFrequencyHz;
                    Err=matConvertSamplerate(pSamplesIn, (unsigned long)NumSamplesIn,
pSamplesOut, (unsigned long)MaxSigLen, Rate, &NumSamplesOut);
                    rc = true;
                }
                else if (aSampleFrequencyHz>48000.0)
                {
                    Rate = 48000.0/aSampleFrequencyHz;
                    Err=matConvertSamplerate(pSamplesIn, (unsigned long)NumSamplesIn,
pSamplesOut, (unsigned long)MaxSigLen, Rate, &NumSamplesOut);
                    rc = true;
                }
                break;
        }
```

```
        *pNumSamplesOut = NumSamplesOut;
    }
    return rc;
}

bool CPairParameters::PairProcess(XFLOAT* pRefSamples, long NumRefSamples, XFLOAT*
pDegSamples, long NumDegSamples, POLQA_HANDLE POLQAHandle)
{
  CNewLogFile       timeFile;
  CDelayPara          DelayPara;

  CPOLQAData *POLQAData = (CPOLQAData*)POLQAHandle;
  XFLOAT* pRefSigLong = 0;
  XFLOAT* pDegSigLong = 0;
  mpPitchVec = 0;
  mpPitchVecDeg = 0;
  pActiveFrameFlags = 0;
  pAslActiveFrameFlags = 0;
  pMarkSectionFlags = 0;

  OPTTRY
  {
    if (0 && aResultsFile.file)
    {
        fprintf(aResultsFile.file, "\CPairParameters::PairProcess() 1\n");
        double EnergyRef = matSum(pRefSamples, NumRefSamples);
        double EnergyDeg = matSum(pDegSamples, NumDegSamples);
        fprintf(aResultsFile.file, "\tSample sum ref:\t%.15e\n", EnergyRef);
        fprintf(aResultsFile.file, "\tSample sum deg:\t%.15e\n", EnergyDeg);
    }

    long ClockCycles = 0;
    double TimeDiff = 0.0;
    CheckTimeMatInit(POLQAData->mh, 4);

    FreeResultData(&POLQAData->pResults);
    POLQAData->pResults = CreateNewResultStruct();
    POLQA_RESULT_DATA* pDisturbanceOverviewHolder = POLQAData->pResults;

    aSampleFrequencyHzSource = aSampleFrequencyHz;

    const int MaxSigLen = (int)(1.5*NumRefSamples+NumDegSamples)*48000.0 /
aSampleFrequencyHz;
    pRefSigLong = (XFLOAT*)matMalloc(MaxSigLen * sizeof(XFLOAT));
    if (!AdjustSampleRates(pRefSamples, NumRefSamples, pRefSigLong, MaxSigLen,
&NumRefSamples))
        matbCopy(pRefSamples, pRefSigLong, NumRefSamples);

    pDegSigLong = (XFLOAT*)matMalloc(MaxSigLen * sizeof(XFLOAT));
    if (!AdjustSampleRates(pDegSamples, NumDegSamples, pDegSigLong, MaxSigLen,
&NumDegSamples ))
        matbCopy(pDegSamples, pDegSigLong, NumDegSamples);

    if (aSampleFrequencyHz>8000.0)
    {
        switch (aListeningCondition)
        {
            case NARROW_H:
            case STANDARD_IRS:
                aSampleFrequencyHz = 8000;
                break;
            case WIDE_H:
                aSampleFrequencyHz = 48000;
                break;
        }
    }
    statics->setSampleRate(aSampleFrequencyHz);
    pDisturbanceOverviewHolder->m_FileSizeInSamples = NumDegSamples;
    pDisturbanceOverviewHolder->m_SampleFrequencyHz = (long)aSampleFrequencyHz;
    pDisturbanceOverviewHolder->m_ListeningCondition = aListeningCondition;

    pRefSamples = pRefSigLong;
    pDegSamples = pDegSigLong;

    const int Framesize = GetTransformLength();
    mMaxModelFrames = (int)(2*max(NumRefSamples, NumDegSamples)/Framesize+3);
```

```
    DelayPara.AllocVectors(mMaxModelFrames);
    DelayPara.mh = POLQAData->mh;
    DelayPara.MaxSigLen = MaxSigLen;
    DelayPara.OriginalNumberOfSamples = NumRefSamples;
    DelayPara.DistortedNumberOfSamples = NumDegSamples;
    DelayPara.LogFile = aResultsFile.file;
    DelayPara.pOriginalSamples = pRefSamples;
    DelayPara.pDistortedSamples = pDegSamples;
    DelayPara.pStartSampleUtterance = &aStartSampleUtterance;
    DelayPara.pStopSampleUtterance = &aStopSampleUtterance;
    DelayPara.pDelayUtterance = &aDelayUtterance;
    DelayPara.MaxModelFrames = mMaxModelFrames;
    DelayPara.Framesize = Framesize / 2;

    pDisturbanceOverviewHolder->m_DelayPerFrame = (long*)matMalloc(mMaxModelFrames *
sizeof(long));
    pDisturbanceOverviewHolder->m_DelayReliabilityPerFrame =
(OTA_FLOAT*)matMalloc(mMaxModelFrames * sizeof(double));

    CheckTimeMatEval(POLQAData->mh, 4, &ClockCycles, &TimeDiff);
    AddProcessingTime(pDisturbanceOverviewHolder, "Model Initialization", TimeDiff,
ClockCycles);
    CheckTimeMatInit(POLQAData->mh, 4);

        bool TAOk = DoCalculateDelayDegPlus(&DelayPara, pDisturbanceOverviewHolder);

    if (!TAOk)
        OPTTHROW((OPT_TRYCATCH_ERRORCODE)CALCULATION_FAILED);

    mpPitchVec = (XFLOAT*)matMalloc(mMaxModelFrames * sizeof(XFLOAT));
    mpPitchVecDeg = (XFLOAT*)matMalloc(mMaxModelFrames * sizeof(XFLOAT));
    pActiveFrameFlags = (bool*)matMalloc(mMaxModelFrames * sizeof(bool));
    pAslActiveFrameFlags = (bool*)matMalloc(mMaxModelFrames * sizeof(bool));

    memcpy(pActiveFrameFlags, DelayPara.pActiveFrameFlags,
sizeof(bool)*mMaxModelFrames);
    memcpy(pAslActiveFrameFlags, DelayPara.pAslActiveFrameFlags,
sizeof(bool)*min(mMaxModelFrames, DelayPara.MaxModelFrames));
    matbCopy(DelayPara.pDelayReliability,pDisturbanceOverviewHolder->m_DelayReliabilityP
erFrame, min(mMaxModelFrames, DelayPara.MaxModelFrames));
    matbCopy(DelayPara.pPitchVecOfRef, mpPitchVec, min(mMaxModelFrames,
DelayPara.MaxModelFrames));
    matbCopy(DelayPara.pPitchVecOfDeg, mpPitchVecDeg, min(mMaxModelFrames,
DelayPara.MaxModelFrames));

    pMarkSectionFlags = (int*)matMalloc(mMaxModelFrames * sizeof(int));
    matbCopy(DelayPara.pIgnoreFrameFlags, pMarkSectionFlags, min(mMaxModelFrames,
DelayPara.MaxModelFrames));

    NumRefSamples = DelayPara.OriginalNumberOfSamples;
    NumDegSamples = DelayPara.DistortedNumberOfSamples;

    mpBGNSwitchingLevel[0] = DelayPara.pBGNSwitchingLevel[0];
    mpBGNSwitchingLevel[1] = DelayPara.pBGNSwitchingLevel[1];
    mpNoiseDuringSilencedB[0] = DelayPara.pNoiseDuringSilencedB[0];
    mpNoiseDuringSilencedB[1] = DelayPara.pNoiseDuringSilencedB[1];
    mpNoiseDuringSpeechdB[0] = DelayPara.pNoiseDuringSpeechdB[0];
    mpNoiseDuringSpeechdB[1] = DelayPara.pNoiseDuringSpeechdB[1];
    pDisturbanceOverviewHolder->m_PitchRef = mPitchFreqRef = DelayPara.PitchFreqRef;
    pDisturbanceOverviewHolder->m_PitchDeg = mPitchFreqDeg = DelayPara.PitchFreqDeg;

    CheckTimeMatEval(POLQAData->mh, 4, &ClockCycles, &TimeDiff);
    AddProcessingTime(pDisturbanceOverviewHolder, "Complete TA", TimeDiff, ClockCycles);
    CheckTimeMatInit(POLQAData->mh, 4);
    statics->setFrameLength(Framesize);
    statics->setNrTimeSamples(max(NumRefSamples, NumDegSamples) +
(long)statics->ZeroPaddingLength);
    {
        const int dummyArrayLength = 1<<matFFTOrder(statics->nrTimesSamples);

        SmartBufferPolqa dummySB(POLQAHandle, dummyArrayLength + 2);
        dummySB.Free();
    }

    InitArrays(statics->nrFrames);
```

```
    aOriginalTimeSeries.Initialize("originalTimeSeries", POLQAHandle);
    aDistortedTimeSeries.Initialize("distortedTimeSeries", POLQAHandle);
    aOriginalTimeSeriesReverb.Initialize("originalTimeSeriesReverb", POLQAHandle);

    aAlignedOriginalTimeSeries.Initialize("alignedDistortedTimeSeries", POLQAHandle);

    aOriginalTimeSeries.ReadFromBuffer(pRefSamples, NumRefSamples);
    aOriginalNumberOfSamples = NumRefSamples;

    aDistortedTimeSeries.ReadFromBuffer(pDegSamples, NumDegSamples);

    aDistortedNumberOfSamples = NumDegSamples-DelayPara.FirstDegSample;

    gLogFile.WriteString("\nBegin NormalizationProcess\n");
    NormalizationProcess (aCalibrationDb);

    CheckTimeMatEval(POLQAData->mh, 4, &ClockCycles, &TimeDiff);
    AddProcessingTime(pDisturbanceOverviewHolder, "Normalization Process", TimeDiff,
ClockCycles);

    CheckTimeMatInit(POLQAData->mh, 4);
    gLogFile.WriteString("\nBegin IdealizationProcess\n");
    IdealizationProcess(pDisturbanceOverviewHolder);

    ASSERT(mMaxModelFrames>=statics->stopFrameIdx);

    CheckTimeMatEval(POLQAData->mh, 4, &ClockCycles, &TimeDiff);
    AddProcessingTime(pDisturbanceOverviewHolder, "Idealization Process", TimeDiff,
ClockCycles);
    CheckTimeMatInit(POLQAData->mh, 4);

    gLogFile.WriteString("\nBegin DisturbanceProcess\n");
    DisturbanceProcess (pDisturbanceOverviewHolder);
    if (!DisturbanceTimeProcess (pDisturbanceOverviewHolder)) {
        if (mpPitchVec) matFree(mpPitchVec);
        if (mpPitchVecDeg) matFree(mpPitchVecDeg);
        if (pActiveFrameFlags) matFree(pActiveFrameFlags);
        if (pAslActiveFrameFlags) matFree(pAslActiveFrameFlags);
        if (pMarkSectionFlags) matFree(pMarkSectionFlags);
        return FALSE;
    }

    CheckTimeMatEval(POLQAData->mh, 4, &ClockCycles, &TimeDiff);
    AddProcessingTime(pDisturbanceOverviewHolder, "Disturbance Process", TimeDiff,
ClockCycles);

  }

  OPTCATCH((OPT_TRYCATCH_ERRORCODE &e))
  {
    if(mpPitchVec != 0)matFree(mpPitchVec);mpPitchVec=0;
    if(mpPitchVecDeg != 0)matFree(mpPitchVecDeg);mpPitchVecDeg=0;
    if(pRefSigLong != 0)matFree(pRefSigLong);pRefSigLong=0;
    if(pDegSigLong != 0)matFree(pDegSigLong);pDegSigLong=0;
    if(pMarkSectionFlags != 0)matFree(pMarkSectionFlags);pMarkSectionFlags=0;
    if(pActiveFrameFlags != 0)matFree(pActiveFrameFlags);pActiveFrameFlags=0;
    if(pAslActiveFrameFlags != 0)matFree(pAslActiveFrameFlags);pAslActiveFrameFlags=0;

    OPTTHROW(e);
  }

    if(mpPitchVec != 0)matFree(mpPitchVec);mpPitchVec=0;
    if(mpPitchVecDeg != 0)matFree(mpPitchVecDeg);mpPitchVecDeg=0;
    if(pRefSigLong != 0)matFree(pRefSigLong);pRefSigLong=0;
    if(pDegSigLong != 0)matFree(pDegSigLong);pDegSigLong=0;
    if(pMarkSectionFlags != 0)matFree(pMarkSectionFlags);pMarkSectionFlags=0;
    if(pActiveFrameFlags != 0)matFree(pActiveFrameFlags);pActiveFrameFlags=0;
    if(pAslActiveFrameFlags != 0)matFree(pAslActiveFrameFlags);pAslActiveFrameFlags=0;

    return TRUE;
}

}
```