```
1
2
3       typedef double XFLOAT
4       typedef double OTA_FLOAT
5
6
7   {
8
9   void CPairParameters::Free(POLQA_HANDLE *POLQAHandle)
10  {
11      (*POLQAHandle)->sqStorage->ClearAll()
12
13      if(*POLQAHandle)
14      {
15          CPOLQAData **POLQAData = (CPOLQAData**)POLQAHandle
16          FreeResultData(&(*POLQAData)->pResults)
17          delete (*POLQAData)
18          *POLQAHandle = 0
19      }
20      aResultsFile.Close()
21  }
22
23  void AddProcessingTime(POLQA_RESULT_DATA *pRes, CNewStdString ProcessingStep, double TimeMs, long
    Cycles)
24  {
25      pRes->m_ClockCycles[pRes->m_NumProcessingTimes] = Cycles
26      pRes->m_ProcessingTime[pRes->m_NumProcessingTimes] = TimeMs
27      pRes->m_ProcessingStep[pRes->m_NumProcessingTimes] = ProcessingStep
28      pRes->m_NumProcessingTimes++
29  }
30
31  POLQA_RESULT_DATA* CPairParameters::GetResults(POLQA_HANDLE POLQAHandle)
32  {
33      if (POLQAHandle)
34      {
35          CPOLQAData *POLQAData = (CPOLQAData*)POLQAHandle
36          return POLQAData->pResults
37      }
38      else return 0
39  }
40
41  bool CPairParameters::AdjustSampleRates(XFLOAT* pSamplesIn, long NumSamplesIn, XFLOAT* pSamplesOut,
    long MaxSigLen, long* pNumSamplesOut)
42  {
43      bool rc=false
44      int Err
45      XFLOAT Rate
46      if (aSampleFrequencyHz>8000.0)
47      {
48          unsigned long NumSamplesOut = *pNumSamplesOut
49          switch (aListeningCondition)
50          {
51              case NARROW_H:
52              case STANDARD_IRS:
53                  Err=matConvertSamplerate2(pSamplesIn, (unsigned long)NumSamplesIn, pSamplesOut,
    (unsigned long)MaxSigLen, aSampleFrequencyHz, 8000, &NumSamplesOut)
54                  rc = true
55                  break
56
57              case WIDE_H:
58                  if (aSampleFrequencyHz<48000.0)
59                  {
60                      Err=matConvertSamplerate2(pSamplesIn, (unsigned long)NumSamplesIn, pSamplesOut,
    (unsigned long)MaxSigLen, aSampleFrequencyHz, 48000, &NumSamplesOut)
61                      rc = true
62                  }
63                  else if (aSampleFrequencyHz>48000.0)
64                  {
```

```
                    Err=matConvertSamplerate2(pSamplesIn, (unsigned long)NumSamplesIn, pSamplesOut,
(unsigned long)MaxSigLen, aSampleFrequencyHz, 48000, &NumSamplesOut)
                    rc = true
                }
                break
        }
        *pNumSamplesOut = NumSamplesOut
    }
    return rc
}

void CPairParameters::CreateDelayVectorFromUtterances(long* pDelayVec, const int VecLen, const int
FrameSize)
{
    if (aStartSampleUtterance.GetSize()>0)
    {
        {
            int StartFrame = aStartSampleUtterance[0]/FrameSize
            long Delay = aDelayUtterance[0]
            for (int i=0  i<StartFrame && i<VecLen  i++)
                pDelayVec[i] = aDelayUtterance[0]
        }

        for (int u=0  u<aStartSampleUtterance.GetSize()-1  u++)
        {
            int StartFrame = aStartSampleUtterance[u]/FrameSize
            int StopFrame  = aStartSampleUtterance[u+1]/FrameSize
            long Delay = aDelayUtterance[u]

            for (int f=StartFrame  f<StopFrame && f<VecLen  f++)
                pDelayVec[f] = Delay
        }

        {
            int u = aStartSampleUtterance.GetSize()-1
            int StartFrame = aStartSampleUtterance[u]/FrameSize
            long Delay = aDelayUtterance[u]
            for (int f = StartFrame  f<VecLen  f++)
                pDelayVec[f] = Delay
        }
    }
    else
    {
        for (int i=0  i<VecLen  i++)
            pDelayVec[i] = 0
    }
}

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, "\tCPairParameters::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()

    POLQAData->pResults->DimArrayLength = 0

    for (int i=0 i<POLQAData->pResults->DimArrayLength i++)
        POLQAData->pResults->DimArray[i] = 0
    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:
                aSampleFrequencyHz = 8000
                break
            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

    const XFLOAT OverlapCoeff = 0.75
    pRefSamples = pRefSigLong
    pDegSamples = pDegSigLong

    const int Framesize = GetTransformLength()
    mMaxModelFrames = (int)(1/(1-OverlapCoeff)*max(NumRefSamples, NumDegSamples)/Framesize+10)

    const double MAX_ALLOWED_UPSAMPLING = 1.5
    statics->setFrameLength(Framesize)
    statics->setNrTimeSamples(max(NumRefSamples, NumDegSamples) + (long)statics->ZeroPaddingLength)

    const int MaxModelFramesMem = statics->nrFrames + 25

    DelayPara.AllocVectors(MaxModelFramesMem)
    DelayPara.mh = POLQAData->mh
    DelayPara.MaxSigLen = MaxSigLen
    DelayPara.OriginalNumberOfSamples = NumRefSamples
```

```
198     DelayPara.DistortedNumberOfSamples = NumDegSamples
199     DelayPara.LogFile = aResultsFile.file
200     DelayPara.pOriginalSamples = pRefSamples
201     DelayPara.pDistortedSamples = pDegSamples
202     DelayPara.pStartSampleUtterance = &aStartSampleUtterance
203     DelayPara.pStopSampleUtterance = &aStopSampleUtterance
204     DelayPara.pDelayUtterance = &aDelayUtterance
205     DelayPara.MaxModelFrames = mMaxModelFrames
206     DelayPara.Framesize = Framesize * (1 - 0.75)
207 
208     pDisturbanceOverviewHolder->m_DelayPerFrame = (long*)matMalloc(MaxModelFramesMem * sizeof(long))
209     pDisturbanceOverviewHolder->m_DelayPerFrameFromUtt = (long*)matMalloc(MaxModelFramesMem *
    sizeof(long))
210     pDisturbanceOverviewHolder->m_DelayReliabilityPerFrame = (OTA_FLOAT*)matMalloc(MaxModelFramesMem
    * sizeof(double))
211 
212     pDisturbanceOverviewHolder->m_NumberOfFrames = MaxModelFramesMem
213 
214     CheckTimeMatEval(POLQAData->mh, 4, &ClockCycles, &TimeDiff)
215     AddProcessingTime(pDisturbanceOverviewHolder, "Model Initialization", TimeDiff, ClockCycles)
216     CheckTimeMatInit(POLQAData->mh, 4)
217 
218         bool TAOk = DoCalculateDelayDegPlus(&DelayPara, pDisturbanceOverviewHolder)
219 
220     if (!TAOk)
221         OPTTHROW((OPT_TRYCATCH_ERRORCODE)CALCULATION_FAILED)
222 
223     mpPitchVec = (XFLOAT*)matMalloc(MaxModelFramesMem * sizeof(XFLOAT))
224     mpPitchVecDeg = (XFLOAT*)matMalloc(MaxModelFramesMem * sizeof(XFLOAT))
225     pActiveFrameFlags = (bool*)matMalloc(MaxModelFramesMem * sizeof(bool))
226     pAslActiveFrameFlags = (bool*)matMalloc(MaxModelFramesMem * sizeof(bool))
227 
228     memcpy(pActiveFrameFlags, DelayPara.pActiveFrameFlags, sizeof(bool)*min(mMaxModelFrames,
    DelayPara.MaxModelFrames))
229     memcpy(pAslActiveFrameFlags, DelayPara.pAslActiveFrameFlags, sizeof(bool)*min(mMaxModelFrames,
    DelayPara.MaxModelFrames))
230     matbCopy(DelayPara.pDelayReliability,pDisturbanceOverviewHolder->m_DelayReliabilityPerFrame,
    min(mMaxModelFrames, DelayPara.MaxModelFrames))
231     matbCopy(DelayPara.pPitchVecOfRef, mpPitchVec, min(mMaxModelFrames, DelayPara.MaxModelFrames))
232     matbCopy(DelayPara.pPitchVecOfDeg, mpPitchVecDeg, min(mMaxModelFrames, DelayPara.MaxModelFrames))
233 
234     pDisturbanceOverviewHolder->m_pPitchVecOfRef = DelayPara.pPitchVecOfRef
235     pDisturbanceOverviewHolder->m_pPitchVecOfDeg = DelayPara.pPitchVecOfDeg
236 
237     pMarkSectionFlags = (int*)matMalloc(MaxModelFramesMem * sizeof(int))
238     matbCopy(DelayPara.pIgnoreFrameFlags, pMarkSectionFlags, min(mMaxModelFrames,
    DelayPara.MaxModelFrames))
239 
240     NumRefSamples = DelayPara.OriginalNumberOfSamples
241     NumDegSamples = DelayPara.DistortedNumberOfSamples
242 
243     mpBGNSwitchingLevel[0] = DelayPara.pBGNSwitchingLevel[0]
244     mpBGNSwitchingLevel[1] = DelayPara.pBGNSwitchingLevel[1]
245     mpNoiseDuringSilencedB[0] = DelayPara.pNoiseDuringSilencedB[0]
246     mpNoiseDuringSilencedB[1] = DelayPara.pNoiseDuringSilencedB[1]
247     mpNoiseDuringSpeechdB[0] = DelayPara.pNoiseDuringSpeechdB[0]
248     mpNoiseDuringSpeechdB[1] = DelayPara.pNoiseDuringSpeechdB[1]
249     pDisturbanceOverviewHolder->m_PitchRef = mPitchFreqRef = DelayPara.PitchFreqRef
250     pDisturbanceOverviewHolder->m_PitchDeg = mPitchFreqDeg = DelayPara.PitchFreqDeg
251 
252     CreateDelayVectorFromUtterances(pDisturbanceOverviewHolder->m_DelayPerFrameFromUtt,
    mMaxModelFrames, Framesize/2)
253 
254     CheckTimeMatEval(POLQAData->mh, 4, &ClockCycles, &TimeDiff)
255     AddProcessingTime(pDisturbanceOverviewHolder, "Complete TA", TimeDiff, ClockCycles)
256     CheckTimeMatInit(POLQAData->mh, 4)
257     statics->setFrameLength(Framesize)
258     statics->setNrTimeSamples(max(NumRefSamples, NumDegSamples) + (long)statics->ZeroPaddingLength)
```

```
259
260     if (statics->nrFrames>mMaxModelFrames)
261     {
262         matbSet(pDisturbanceOverviewHolder->m_DelayPerFrame[mMaxModelFrames-1],
    pDisturbanceOverviewHolder->m_DelayPerFrame+mMaxModelFrames,
    statics->nrFrames-mMaxModelFrames)
263         matbSet(pDisturbanceOverviewHolder->m_DelayPerFrameFromUtt[mMaxModelFrames-1],
    pDisturbanceOverviewHolder->m_DelayPerFrameFromUtt+mMaxModelFrames,
    statics->nrFrames-mMaxModelFrames)
264         matbSet(pDisturbanceOverviewHolder->m_DelayReliabilityPerFrame[mMaxModelFrames-1],
    pDisturbanceOverviewHolder->m_DelayReliabilityPerFrame+mMaxModelFrames,
    statics->nrFrames-mMaxModelFrames)
265         matbSet(mpPitchVec[mMaxModelFrames-1], mpPitchVec+mMaxModelFrames,
    statics->nrFrames-mMaxModelFrames)
266         matbSet(mpPitchVecDeg[mMaxModelFrames-1], mpPitchVecDeg+mMaxModelFrames,
    statics->nrFrames-mMaxModelFrames)
267         matbSet(pActiveFrameFlags[mMaxModelFrames-1], pActiveFrameFlags+mMaxModelFrames,
    statics->nrFrames-mMaxModelFrames)
268         matbSet(pAslActiveFrameFlags[mMaxModelFrames-1], pAslActiveFrameFlags+mMaxModelFrames,
    statics->nrFrames-mMaxModelFrames)
269         matbSet(pMarkSectionFlags[mMaxModelFrames-1], pMarkSectionFlags+mMaxModelFrames,
    statics->nrFrames-mMaxModelFrames)
270     }
271
272     {
273         const int dummyArrayLength = 1<<matFFTOrder(statics->nrTimesSamples)
274
275         SmartBufferPolqa dummySB(POLQAHandle, dummyArrayLength + 2)
276         dummySB.Free()
277     }
278
279     InitArrays(statics->nrFrames)
280
281     aOriginalTimeSeries.Initialize("originalTimeSeries", POLQAHandle)
282     aDistortedTimeSeries.Initialize("distortedTimeSeries", POLQAHandle)
283     aOriginalTimeSeriesReverb.Initialize("originalTimeSeriesReverb", POLQAHandle)
284
285     aAlignedOriginalTimeSeries.Initialize("alignedDistortedTimeSeries", POLQAHandle)
286
287     aOriginalTimeSeries.ReadFromBuffer(pRefSamples, NumRefSamples)
288     aOriginalNumberOfSamples = NumRefSamples
289
290     aDistortedTimeSeries.ReadFromBuffer(pDegSamples, NumDegSamples)
291
292     aDistortedNumberOfSamples = NumDegSamples-DelayPara.FirstDegSample
293
294
295     NormalizationProcess (aCalibrationDb)
296
297     if (statics->listeningCondition == NARROW_H)
298         statics->listeningCondition = SWB
299
300     CheckTimeMatEval(POLQAData->mh, 4, &ClockCycles, &TimeDiff)
301     AddProcessingTime(pDisturbanceOverviewHolder, "Normalization Process", TimeDiff, ClockCycles)
302
303     CheckTimeMatInit(POLQAData->mh, 4)
304
305     IdealizationProcess(pDisturbanceOverviewHolder)
306
307     ASSERT(mMaxModelFrames>=statics->stopFrameIdx)
308
309     CheckTimeMatEval(POLQAData->mh, 4, &ClockCycles, &TimeDiff)
310     AddProcessingTime(pDisturbanceOverviewHolder, "Idealization Process", TimeDiff, ClockCycles)
311     CheckTimeMatInit(POLQAData->mh, 4)
312
313
314     DisturbanceProcess (pDisturbanceOverviewHolder)
315
```

```
    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)
  }

  OPTCATCH((char* s))
  {
      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

      aResultsFile.
      OPTTHROW(s)
  }

   aResultsFile.

    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
}

}
```