```

    typedef double XFLOAT
    typedef double OTA_FLOAT

extern BOOL         gBatchMode


{

extern XFLOAT       gAbsoluteThresholdP []
extern CNewLogFile      gLogFile
extern CNewStdString    gLogFileName

CSignal::CSignal() : m_pData(0), aInitialized(false)
{
    POLQAHandle = 0
}

CSignal::~CSignal()
{
    Free()
}

void CSignal::Initialize(CNewStdString pName, int pNumberOfBands, const CPOLQAData* POLQAHandle)
{
    this->POLQAHandle = POLQAHandle
    statics = POLQAHandle->statics

    aName = pName
    aNumberOfWindows = statics->nrFrames
    aNumberOfBands = pNumberOfBands

    if(aInitialized)
    {
        Free()
        aInitialized = false
    }

    m_pData = (XFLOAT**)matCalloc2D(aNumberOfWindows, pNumberOfBands * sizeof(XFLOAT))

    aName = pName

    ASSERT (aNumberOfWindows > 0)
    ASSERT (aNumberOfBands > 0)

    aInitialized = true
}

void CSignal::Free()
{
    if(m_pData)
        matFree2D((void**)m_pData)
    m_pData = 0
    aNumberOfWindows = 0
    aNumberOfBands = 0
}

int CSignal::GetSize()
{
    return aNumberOfWindows
}

XFLOAT CSignal::Maximum (int pFrameIndex)
{
    XFLOAT   result

    const XFLOAT lowerBound = -1e10
```

```
69
70      result = matMax(m_pData[pFrameIndex], aNumberOfBands)
71
72      if(result < lowerBound)
73          result = lowerBound
74
75      return result
76  }
77
78  void CSignal::MultiplyWith (int pFrameIndex, XFLOAT pFactor)
79  {
80      matbMpy1(pFactor, m_pData[pFrameIndex], aNumberOfBands)
81  }
82
83  void CPsqmArray::DifferenceOf(const CPsqmArray &pInputCPsqmArray1, const CPsqmArray
    &pInputCPsqmArray2)
84  {
85      XFLOAT *pInputCPsqmA1, *pInputCPsqmA2
86
87      for (int frameIndex = statics->startFrameIdx  frameIndex <= statics->stopFrameIdx  frameIndex++)
88      {
89
90          pInputCPsqmA1 = pInputCPsqmArray1.m_pData[frameIndex]
91          pInputCPsqmA2 = pInputCPsqmArray2.m_pData[frameIndex]
92
93          matbSub3(pInputCPsqmA1, pInputCPsqmA2, m_pData[frameIndex], aNumberOfBands)
94
95          for(int band = 0  band < statics->aDifferenceBarkScalingLen   band++)
96              m_pData[frameIndex][band] *= statics->aDifferenceBarkScaling[band]
97
98      }
99  }
100
101 void CPsqmArray::FractionOf (XFLOAT pConstant, int frame)
102 {
103     if(AlmostEqualUlpsFinal((float)pConstant, 0.0f))
104         matbZero(m_pData[frame], aNumberOfBands)
105     else
106         matbMpy1(pConstant, m_pData[frame], aNumberOfBands)
107 }
108
109 void CPsqmArray::MaskWith(CPOLQAData        *POLQAHandle,
110     const CPsqmArray &pMaskSpectrum) {
111     XFLOAT    maskLevel, h
112     int       frameIndex, bandIndex
113
114     for (frameIndex = statics->startFrameIdx  frameIndex <= statics->stopFrameIdx  frameIndex++) {
115
116         for (bandIndex = 0  bandIndex < aNumberOfBands  bandIndex++) {
117             maskLevel = pMaskSpectrum.m_pData[frameIndex][bandIndex]*pow((bandIndex+3.0),0.05)*0.85
118
119             h = this->m_pData[frameIndex][bandIndex]
120             maskLevel = maskLevels[bandIndex]
121             if (h > maskLevel)
122             {
123                 this->m_pData[frameIndex][bandIndex] -= maskLevel
124             }
125             else
126             {
127                 if (h < -maskLevel)
128                 {
129                     this->m_pData[frameIndex][bandIndex] += maskLevel
130                 }
131                 else
132                 {
133                     this->m_pData[frameIndex][bandIndex] = 0
134                 }
135             }
```

```
        }
    }
}

void CPsqmArray::MaskWith48k(CPOLQAData     *POLQAHandle,
    const CPsqmArray &pMaskSpectrum) {
    XFLOAT    maskLevel, h
    int       frameIndex, bandIndex

    for (frameIndex = statics->startFrameIdx  frameIndex <= statics->stopFrameIdx  frameIndex++) {

        for (bandIndex = 0  bandIndex < aNumberOfBands  bandIndex++) {
            maskLevel = pMaskSpectrum.m_pData[frameIndex][bandIndex] * pow((bandIndex + 3.0),
0.05)*0.85

            h = this->m_pData[frameIndex][bandIndex]
            maskLevel = maskLevels[bandIndex]
            if (h > maskLevel)
            {
                this->m_pData[frameIndex][bandIndex] -= maskLevel
            }
            else
            {
                if (h < -maskLevel)
                {
                    this->m_pData[frameIndex][bandIndex] += maskLevel
                }
                else
                {
                    this->m_pData[frameIndex][bandIndex] = 0
                }
            }
        }
    }
}

void CPsqmArray::MinimumOf (const CPsqmArray &pSignal1, const CPsqmArray &pSignal2)
{
    XFLOAT    h1, h2

    for(int frameIndex = statics->startFrameIdx  frameIndex <= statics->stopFrameIdx  frameIndex++)
    {
        for(int bandIndex = 0  bandIndex < aNumberOfBands  bandIndex++)
        {
            h1 = pSignal1.m_pData[frameIndex][bandIndex]
            h2 = pSignal2.m_pData[frameIndex][bandIndex]

            if (h1 < h2) {
                this->m_pData[frameIndex][bandIndex] = h1
            } else {
                this->m_pData[frameIndex][bandIndex] = h2
            }
        }
    }
}

void CPsqmArray::operator*= (XFLOAT pValue)
{
    XFLOAT *ptr

    for(int frameIndex = statics->startFrameIdx  frameIndex <= statics->stopFrameIdx  frameIndex++)
    {
        ptr = this->m_pData[frameIndex]
        matbMpy1(pValue, ptr, aNumberOfBands)
    }
}

void CPsqmArray::operator*= (const CPsqmArray &pInputSignal)
```

```
{
    XFLOAT *ptr1, *ptr2

    for (int frameIndex = statics->startFrameIdx  frameIndex <= statics->stopFrameIdx  frameIndex++)
    {
        ptr2 = this->m_pData[frameIndex]
        ptr1 = pInputSignal.m_pData[frameIndex]
        matbMpy2(ptr1, ptr2, aNumberOfBands)
    }
}

void CPsqmArray::operator+= (XFLOAT pValue)
{
    XFLOAT *ptr

    for(int frameIndex = statics->startFrameIdx  frameIndex <= statics->stopFrameIdx  frameIndex++)
    {
        ptr = this->m_pData[frameIndex]
        matbAdd1(pValue, ptr, aNumberOfBands)
    }
}

int GetUtteranceForSample(const CIntArray &pStartSampleUtterance, const CIntArray
&pStopSampleUtterance, const CIntArray &DelayUtterance, int SampleIndex)
{
    int i=0
    int Length = pStopSampleUtterance.GetSize()-1

    while (i<Length && pStopSampleUtterance.m_pData[i]<SampleIndex) i++

    return i
}

int GetUtteranceForFrame(const CIntArray &pStartSampleUtterance, const CIntArray
&pStopSampleUtterance, const CIntArray &DelayUtterance, int FrameIndex, int Windowsize)
{
    const XFLOAT OverlapCoeff = 1 - 0.75
    int StartOfFrame = FrameIndex * Windowsize * OverlapCoeff
    return GetUtteranceForSample(pStartSampleUtterance, pStopSampleUtterance, DelayUtterance,
StartOfFrame)
}

void CPsqmArray::STFTPowerSpectrumOf (CPOLQAData *POLQAHandle,
                                      const CTimeSeries &pTimeSeries,
                                      const CIntArray   &pStartSampleUtterance,
                                      const CIntArray   &pStopSampleUtterance,
                                      const CIntArray   &DelayUtterance,
                                      const bool IsRef,
                                      const bool IgnoreDelay)
{
    int Windowsize = pTimeSeries.GetFrameLength()
    if (IgnoreDelay || IsRef)
    {
        for (int frameIndex = statics->startFrameIdx  frameIndex <= statics->stopFrameIdx
frameIndex++)
        {
            int utt = GetUtteranceForFrame(pStartSampleUtterance, pStopSampleUtterance,
DelayUtterance, frameIndex, Windowsize)
            if (utt<0)
                matbZero(this->m_pData[frameIndex], aNumberOfBands)
            else
                STFTPowerSpectrumOf (POLQAHandle, pTimeSeries, frameIndex, 0)
        }
    }
    else
    {
        for (int frameIndex = statics->startFrameIdx  frameIndex <= statics->stopFrameIdx
frameIndex++)
```

```
265         {
266             int utt = GetUtteranceForFrame(pStartSampleUtterance, pStopSampleUtterance,
    DelayUtterance, frameIndex, Windowsize)
267             if (utt<0)
268                 matbZero(this->m_pData[frameIndex], aNumberOfBands)
269             else
270                 STFTPowerSpectrumOf (POLQAHandle, pTimeSeries, frameIndex, DelayUtterance[utt])
271         }
272     }
273
274 }
275
276 void CPsqmArray::STFTPowerSpectrumOf(   CPOLQAData *POLQAHandle,
277                                         const CTimeSeries &pTimeSeries,
278                                         int pFrameIndex,
279                                         int Delay
280                                     )
281 {
282
283     XFLOAT              a, b
284
285     int                 n = pTimeSeries. GetFrameLength ()
286     XFLOAT *x
287     const XFLOAT        OverlapCoeff = 0.75
288     if (POLQAHandle->STFTBufferLength != n)
289     {
290         if (POLQAHandle->STFTBufferLength > 0)
291             matFree(POLQAHandle->STFTBuffer)
292
293         POLQAHandle->STFTBuffer = (XFLOAT*)matMalloc((n + 2)*sizeof(XFLOAT))
294
295         int p = 1
296         POLQAHandle->STFTBufferLengthLog2 = 0
297         while (p < n) {
298             p *= 2
299             POLQAHandle->STFTBufferLengthLog2++
300         }
301
302         ASSERT (p == n)
303
304         POLQAHandle->STFTBufferLength = n
305     }
306     x = POLQAHandle->STFTBuffer
307
308     ASSERT (2*aNumberOfBands == n)
309
310     int StartPos = pFrameIndex * (1-OverlapCoeff)*n + Delay
311     int EndPos   = StartPos + n - 1
312     int Length
313     const int timeSeriesLen = statics->nrTimesSamples
314
315     if (StartPos < 0)
316         matbZero(&x[0], (((-StartPos) < (n)) ? (-StartPos) : (n)))
317     if (EndPos  >= timeSeriesLen)
318         matbZero(&x[n-1-(((EndPos - timeSeriesLen) < (n-1)) ? (EndPos - timeSeriesLen) : (n-1))],
    (((EndPos - timeSeriesLen +1) < (n)) ? (EndPos - timeSeriesLen +1) : (n)))
319
320     StartPos = (((StartPos) < (timeSeriesLen -1)) ? (StartPos) : (timeSeriesLen -1))
321     EndPos   = (((EndPos) > (0)) ? (EndPos) : (0))
322     Length   = (((EndPos) < (timeSeriesLen -1)) ? (EndPos) : (timeSeriesLen -1)) - (((StartPos) >
    (-StartPos)) ? (StartPos) : (-StartPos)) + 1
323     if (-StartPos < n && Length <= n-(((-StartPos) > (0)) ? (-StartPos) : (0)))
324         matbCopy(pTimeSeries.m_pData + (((StartPos) > (0)) ? (StartPos) : (0)), x + (((-StartPos) >
    (0)) ? (-StartPos) : (0)), Length)
325     matbMpy2(statics->frameWindow, x, n)
326
327     matRealFft (POLQAHandle->mh, x, POLQAHandle->STFTBufferLengthLog2, MAT_Forw)
328
```

```
329     this->m_pData[pFrameIndex][0] = 0
330
331     for (int bandIndex = 1  bandIndex < aNumberOfBands  bandIndex++)
332     {
333         a = x [2 * bandIndex]
334         b = x [2 * bandIndex + 1]
335         this->m_pData[pFrameIndex][bandIndex] = a * a + b * b
336     }
337
338 }
339
340 void CPsqmArray::STFTPowerAndPhaseSpectrumOf ( CPOLQAData *POLQAHandle,
341                                         const CTimeSeries &pTimeSeries,
342                                         const CIntArray   &pStartSampleUtterance,
343                                         const CIntArray   &pStopSampleUtterance,
344                                         const CIntArray   &DelayUtterance,
345                                         bool IsRef, bool IgnoreDelay) {
346     int frameIndex
347     int Windowsize = pTimeSeries. GetFrameLength ()
348
349     if (IgnoreDelay || IsRef)
350     {
351         for (frameIndex = statics->startFrameIdx  frameIndex <= statics->stopFrameIdx  frameIndex++)
352         {
353             int utt = GetUtteranceForFrame(pStartSampleUtterance, pStopSampleUtterance,
    DelayUtterance, frameIndex, Windowsize)
354             if (utt<0)
355                 matbZero(this->m_pData[frameIndex], aNumberOfBands)
356             else
357                 STFTPowerAndPhaseSpectrumOf (POLQAHandle, pTimeSeries, frameIndex, 0)
358         }
359     }
360     else
361     {
362         for (frameIndex = statics->startFrameIdx  frameIndex <= statics->stopFrameIdx  frameIndex++)
363         {
364             int utt = GetUtteranceForFrame(pStartSampleUtterance, pStopSampleUtterance,
    DelayUtterance, frameIndex, Windowsize)
365             if (utt<0)
366                 matbZero(this->m_pData[frameIndex], aNumberOfBands)
367             else
368                 STFTPowerAndPhaseSpectrumOf (POLQAHandle, pTimeSeries, frameIndex,
    IsRef?0:DelayUtterance.m_pData[utt])
369         }
370     }
371 }
372
373 void CPsqmArray::STFTPowerAndPhaseSpectrumOf (  CPOLQAData  *POLQAHandle,
374                                                 const       CTimeSeries &pTimeSeries,
375                                                 int         pFrameIndex,
376                                                 int         Delay)
377 {
378
379     XFLOAT              a, b
380
381     int                 n = pTimeSeries. GetFrameLength ()
382     XFLOAT              *x
383     CNewStdString          s
384     const XFLOAT        OverlapCoeff = 0.75
385     if (POLQAHandle->STFTBufferLength != n)
386     {
387         if (POLQAHandle->STFTBufferLength > 0)
388             matFree(POLQAHandle->STFTBuffer)
389
390         POLQAHandle->STFTBuffer = (XFLOAT*)matMalloc((n + 2)*sizeof(XFLOAT))
391
392         int p = 1
393         POLQAHandle->STFTBufferLengthLog2 = 0
```

```
394         while (p < n) {
395             p *= 2
396             POLQAHandle->STFTBufferLengthLog2++
397         }
398 
399         ASSERT (p == n)
400 
401         POLQAHandle->STFTBufferLength = n
402     }
403 
404     x = POLQAHandle->STFTBuffer
405 
406     ASSERT (2*aNumberOfBands == n)
407 
408     int StartPos = pFrameIndex * (1-OverlapCoeff)*n + Delay
409     int EndPos   = StartPos + n - 1
410     int Length
411     const int timeSeriesLen = statics->nrTimesSamples
412 
413     if (StartPos < 0)
414         matbZero(&x[0], (((-StartPos) < (n)) ? (-StartPos) : (n)))
415     if (EndPos >= timeSeriesLen)
416         matbZero(&x[n-1-(((EndPos - timeSeriesLen) < (n-1)) ? (EndPos - timeSeriesLen) : (n-1))],
    (((EndPos - timeSeriesLen +1) < (n)) ? (EndPos - timeSeriesLen +1) : (n)))
417 
418     StartPos = (((StartPos) < (timeSeriesLen -1)) ? (StartPos) : (timeSeriesLen -1))
419     EndPos   = (((EndPos) > (0)) ? (EndPos) : (0))
420     Length   = (((EndPos) < (timeSeriesLen -1)) ? (EndPos) : (timeSeriesLen -1)) - (((StartPos) >
    (-StartPos)) ? (StartPos) : (-StartPos)) + 1
421     if (-StartPos < n && Length <= n-(((-StartPos) > (0)) ? (-StartPos) : (0)))
422         matbCopy(pTimeSeries.m_pData + (((StartPos) > (0)) ? (StartPos) : (0)), x + (((-StartPos) >
    (0)) ? (-StartPos) : (0)), Length)
423     matbMpy2(statics->frameWindow, x, n)
424 
425     matRealFft(POLQAHandle->mh, x, POLQAHandle->STFTBufferLengthLog2, MAT_Forw)
426 
427     this->m_pData[pFrameIndex][0] = 0
428 
429     for (int bandIndex = 1  bandIndex < aNumberOfBands  bandIndex++) {
430         a = x [2 * bandIndex]
431         b = x [2 * bandIndex + 1]
432         this->m_pData[pFrameIndex][bandIndex] = a * a + b * b
433     }
434 
435 }
436 
437 void CHzSpectrum::Initialize (CNewStdString pName, const CPOLQAData* POLQAHandle)
438 {
439     CSignal::Initialize(pName, POLQAHandle->statics->aNumberOfHzBands, POLQAHandle)
440 }
441 
442 void CBarkSpectrum::Initialize(CNewStdString pName, const CPOLQAData *POLQAHandle)
443 {
444     CSignal::Initialize(pName, POLQAHandle->statics->aNumberOfBarkBands, POLQAHandle)
445 
446 }
447 
448 void CBarkSpectrum::ExcitationOf(const CPOLQAData *POLQAHandle, const CBarkSpectrum &pInputArray,
    bool* pUseThisFrame, int pListeningConditionChoice)
449  {
450     int       mu, nu
451     XFLOAT     t2, q, factor, bandQ, hulpLow=0.0, hulpTotal=0.0, hulp
452 
453     XFLOAT    hulpCorrection, h
454 
455     int       forLim1, forLim2, convLow, convHigh
456     int       frameIndex
457     CNewStdString  s
```

```
458
459     convLow  = 1
460     convHigh = 10
461
462     for (frameIndex = statics->startFrameIdx  frameIndex <= statics->stopFrameIdx  frameIndex++)
463     {
464         hulpLow  = 0.0
465         hulpTotal = 0.0
466
467         for (nu = 3  nu < statics->aNumberOfBarkBands  nu++)
468         {
469
470             bandQ = (XFLOAT) pow ((XFLOAT)pInputArray.m_pData[frameIndex][nu], (XFLOAT) 2.0 /
    (XFLOAT) 2.0)
471
472             factor = statics->aT1
473             q = bandQ
474             forLim1 = nu - convLow
475             if (forLim1 < 0) {forLim1 = 0 }
476
477             for (mu = nu  ((mu >= forLim1) && (q != 00))  mu--)
478             {
479                 this->m_pData[frameIndex][mu] += q
480                 q *= factor
481             }
482
483             t2 = statics->aT20[nu] * (XFLOAT) pow ((XFLOAT)bandQ, (XFLOAT)(0.22 *
    statics->aWidthOfBandBark[1]))
484             factor = t2
485             q = t2 * bandQ
486
487             forLim2 = (((aNumberOfBands - 1) < (nu + convHigh)) ? (aNumberOfBands - 1) : (nu +
    convHigh))
488
489             for (mu = nu + 1  ((mu <= forLim2) && (q != 00))  mu++) {
490                 this->m_pData[frameIndex][mu] += q
491                 q *= factor
492             }
493             if (statics->aCentreOfBandHz[nu]<100.0)
494                 hulpLow += pInputArray.m_pData[frameIndex][nu]
495             if (statics->aCentreOfBandHz[nu]<3600.0)
496                 hulpTotal += pInputArray.m_pData[frameIndex][nu]
497         }
498
499             hulpTotal = (2.0*pow((hulpTotal+3.0e4)/(hulpLow+3.0e4),0.2))
500
501         for (mu = 1  mu < aNumberOfBands  mu++) {
502
503             h = this->m_pData[frameIndex][mu]
504             this->m_pData[frameIndex][mu] = (XFLOAT) pow (h, (XFLOAT) 2.0/(XFLOAT) 2.0)
505
506             hulp = (statics->aCentreOfBandBark[mu] + 1.0)
507
508             if ( this->m_pData[frameIndex][mu] >(1.0e6/hulp))
509                 this->m_pData[frameIndex][mu] = (1.0e6/hulp) - 1.0 +
    pow((this->m_pData[frameIndex][mu] - (1.0e6/hulp) +1.0 ),0.99)
510             if ( this->m_pData[frameIndex][mu] >(1.0e7/hulp))
511                 this->m_pData[frameIndex][mu] = (1.0e7/hulp) - 1.0 +
    pow((this->m_pData[frameIndex][mu] - (1.0e7/hulp) +1.0 ),0.98)
512             if ( this->m_pData[frameIndex][mu] >(1.0e8/hulp))
513                 this->m_pData[frameIndex][mu] = (1.0e8/hulp) - 1.0 +
    pow((this->m_pData[frameIndex][mu] - (1.0e8/hulp) +1.0 ),0.95)
514             if ( this->m_pData[frameIndex][mu] >(1.0e9/hulp))
515                 this->m_pData[frameIndex][mu] = (1.0e9/hulp) - 1.0 +
    pow((this->m_pData[frameIndex][mu] - (1.0e9/hulp) +1.0 ),0.88)
516             if ( this->m_pData[frameIndex][mu] >(1.0e10/hulp))
517                 this->m_pData[frameIndex][mu] = (1.0e10/hulp) - 1.0 +
    pow((this->m_pData[frameIndex][mu] - (1.0e10/hulp) +1.0 ),0.80)
```

```
518
519         }
520     }
521
522     for (frameIndex = statics->startFrameIdx + 1  frameIndex <= statics->stopFrameIdx  frameIndex++)
    {
523         for (nu = 1  nu < aNumberOfBands  nu++) {
524             if ( (pUseThisFrame[frameIndex-1]) && (pUseThisFrame[frameIndex]) )
525                 this->m_pData[frameIndex][nu] += (0.2/pow((aCentreOfBandBark.m_pData[nu]+1.0),1.0))*
    this->m_pData[frameIndex-1][nu]
526         }
527     }
528
529     for (frameIndex = statics->startFrameIdx  frameIndex <= statics->stopFrameIdx  frameIndex++)
530     {
531         for (nu = 0  nu < aNumberOfBands  nu++) {
532             hulp = pow((aCentreOfBandBark.m_pData[nu]+0.5),0.2)
533             hulpCorrection = aCentreOfBandBark.m_pData[nu]/hulpTotal
534             if (hulpCorrection>1.0) hulpCorrection = 1.0
535             hulpCorrection = pow(hulpCorrection,8.0)
536             this->m_pData[frameIndex][nu] = (pInputArray.m_pData[frameIndex][nu] -
    1.0e-6*this->m_pData[frameIndex][nu]) / pow (
    (hulpCorrection*(this->m_pData[frameIndex][nu] + 4.0e4) + 1.0) , 0.11*hulp )
537             if (this->m_pData[frameIndex][nu] < 0.0) this->m_pData[frameIndex][nu] = 0.0
538         }
539     }
540
541 }
542
543 void CBarkSpectrum::TimeLpAudibleOf(const CPOLQAData      *POLQAHandle,
544                                     const CBarkSpectrum  &pInputSpectrum,
545                                     const CIntArray      &pActiveRatioOk,
546                                     XFLOAT                pPower)
547 {
548
549     for (int bandIndex = 0  bandIndex < aNumberOfBands  bandIndex++)
550     {
551         XFLOAT result = 0
552         int    count = 0
553         for (int frameIndex = statics->startFrameIdx  frameIndex <= statics->stopFrameIdx
    frameIndex++)
554         {
555             count++
556             if (pActiveRatioOk.m_pData[frameIndex])
557             {
558                 if (pInputSpectrum.m_pData[frameIndex][bandIndex]  >
    10.0*aAbsoluteThresholdPower.m_pData[bandIndex])
559                 {
560                     result += pow((pInputSpectrum.m_pData[frameIndex][bandIndex]), pPower)
561                 }
562             }
563         }
564
565         this->m_pData[0][bandIndex] = (XFLOAT) (pow ( (result/(count+1.0)), 1 / pPower) )
566     }
567
568 }
569
570 void CBarkSpectrum::TimeLpAudibleOfSilent ( const CBarkSpectrum     &pInputSpectrum,
571                                             const CIntArray         &pSilent,
572                                             XFLOAT                  pPower,
573                                             int                     pNumberOfSilentFrames)
574 {
575     for (int bandIndex = 0  bandIndex < aNumberOfBands  bandIndex++)
576     {
577         XFLOAT result = 0
578         XFLOAT hulp = 0.0
579
```

```
580         for (int frameIndex = statics->startFrameIdx  frameIndex <= statics->stopFrameIdx
    frameIndex++)
581         {
582             if ( (pSilent.m_pData[frameIndex]))
583             {
584                 result += pow((pInputSpectrum.m_pData[frameIndex][bandIndex]), pPower)
585             }
586         }
587         this->m_pData[0][bandIndex] = (XFLOAT) (pow ( (result/(pNumberOfSilentFrames+1.0)),1 /
    pPower) )
588     }
589 }
590
591 void CBarkSpectrum::TimeLpOf(const CPOLQAData      *POLQAHandle,
592                              const CBarkSpectrum  &pInputSpectrum,
593                              const CIntArray      &pActive,
594                              XFLOAT                pPower)
595 {
596     int length = statics->stopFrameIdx - statics->startFrameIdx + 1
597
598     SmartBufferPolqa SB1(POLQAHandle, length)
599     XFLOAT *temp1 = SB1.Buffer
600     SmartBufferPolqa SB2(POLQAHandle, length)
601     XFLOAT *temp2 = SB2.Buffer
602
603     int count
604     XFLOAT result
605
606     for (int bandIndex = 0  bandIndex < aNumberOfBands  bandIndex++)
607     {
608         result = 0
609         count = 0
610         for (int frameIndex = statics->startFrameIdx  frameIndex <= statics->stopFrameIdx
    frameIndex++)
611         {
612             if (pActive.m_pData[frameIndex])
613             {
614                 temp1[count] = pInputSpectrum.m_pData[frameIndex][bandIndex]
615                 count++
616             }
617         }
618         matbPow2(temp1, pPower, temp2, count)
619
620         result = matSum(temp2, count)
621
622         this->m_pData[0][bandIndex] = (XFLOAT)(pow((result/(count+1.0)), 1/pPower))
623     }
624
625 }
626
627 void CPairParameters::PrintFrequencyResponse(CNewLogFile            &pResultsFile,
628                                              const CBarkSpectrum    &pOriginalPitchPowerDensitySum,
629                                              const CBarkSpectrum    &pDistortedPitchPowerDensitySum)
630 {
631     XFLOAT  x
632     CNewStdString s
633
634     pResultsFile. WriteString ("PERCEPTUAL FREQUENCY RESPONSE in dB\n")
635     pResultsFile. WriteString ("Frequency (Bark)   Frequency (Hz)   Relative Level (dB)   Band Index
    Number\n")
636     for (int bandIndex = 0  bandIndex < statics->aNumberOfBarkBands  bandIndex++)
637     {
638         if ((pOriginalPitchPowerDensitySum.m_pData[0][bandIndex] < 0.1)  &&
    (pDistortedPitchPowerDensitySum.m_pData[0][bandIndex] < 0.1) )
639         {
640             x = 98765.43
641         }
642         else
```

```
643         {
644             x = 10*log ((XFLOAT) ((pDistortedPitchPowerDensitySum.m_pData[0][bandIndex] +
    0.01)/(pOriginalPitchPowerDensitySum.m_pData[0][bandIndex] + 0.01))) / log(10.0)
645         }
646         s. Format ("      %5.1f      %8.0f               %5.1f                %i \n",
    statics->aCentreOfBandBark[bandIndex], statics->aCentreOfBandHz[bandIndex], x, bandIndex)
647         pResultsFile.WriteString (s)
648     }
649     pResultsFile.WriteString ("\n\n")
650 }
651
652 void CBarkSpectrum::AudibleFreqRespCompensationOf(const CPOLQAData      *POLQAHandle,
653                                                   const  CBarkSpectrum  &pInputSpectrum1,
654                                                   const  CBarkSpectrum  &pInputSpectrum2,
655                                                   XFLOAT                 pConstant,
656                                                   XFLOAT                 pPower,
657                                                    int                   pListeningConditionChoice){
658     int    frameIndex
659     int    bandIndex
660     XFLOAT ref_pow, deg_pow
661
662     ref_pow = 0.0
663     deg_pow = 0.0
664     for (bandIndex = 0  bandIndex < aNumberOfBands  bandIndex++) {
665         ref_pow +=  pInputSpectrum1.m_pData[0][bandIndex]
666         deg_pow +=  pInputSpectrum2.m_pData[0][bandIndex]
667     }
668
669     XFLOAT const ss = (ref_pow+0.1) / (deg_pow+0.1)
670
671     XFLOAT x, y, hulp
672     for (bandIndex = 0  bandIndex < aNumberOfBands  bandIndex++) {
673         if (bandIndex == 0)
674             x = (ss*pInputSpectrum2.m_pData[0][0] + pConstant) / (pInputSpectrum1.m_pData[0][0] +
    pConstant)
675         if (bandIndex == 1)
676             x = (ss*pInputSpectrum2.m_pData[0][0] + ss*pInputSpectrum2.m_pData[0][1] + pConstant) /
    (pInputSpectrum1.m_pData[0][0] + pInputSpectrum1.m_pData[0][1] + pConstant)
677         if (bandIndex == 2)
678             x = (ss*pInputSpectrum2.m_pData[0][1] + ss*pInputSpectrum2.m_pData[0][2] + pConstant) /
    (pInputSpectrum1.m_pData[0][1] + pInputSpectrum1.m_pData[0][2] + pConstant)
679         if (bandIndex == aNumberOfBands-1)
680             x = (ss*pInputSpectrum2.m_pData[0][aNumberOfBands-1] + pConstant) /
    (pInputSpectrum1.m_pData[0][aNumberOfBands-1] + pConstant)
681         if (bandIndex == aNumberOfBands-2)
682             x = (ss*pInputSpectrum2.m_pData[0][aNumberOfBands-1] +
    ss*pInputSpectrum2.m_pData[0][aNumberOfBands-2] + pConstant) /
    (pInputSpectrum1.m_pData[0][aNumberOfBands-1] +
    pInputSpectrum1.m_pData[0][aNumberOfBands-2] + pConstant)
683         if ( (bandIndex > 10) && (bandIndex < (aNumberOfBands-4)) )
684             x = (ss*pInputSpectrum2.m_pData[0][bandIndex-2] +
    ss*pInputSpectrum2.m_pData[0][bandIndex-1] + ss*pInputSpectrum2.m_pData[0][bandIndex] +
    ss*pInputSpectrum2.m_pData[0][bandIndex+1] + ss*pInputSpectrum2.m_pData[0][bandIndex+2]
    + pConstant) / (pInputSpectrum1.m_pData[0][bandIndex-2] +
    pInputSpectrum1.m_pData[0][bandIndex-1] + pInputSpectrum1.m_pData[0][bandIndex] +
    pInputSpectrum1.m_pData[0][bandIndex+1] + pInputSpectrum1.m_pData[0][bandIndex+2] +
    pConstant)
685         if ((bandIndex > 2) && (bandIndex < (aNumberOfBands-2)) )
686             x = (ss*pInputSpectrum2.m_pData[0][bandIndex-1] +
    ss*pInputSpectrum2.m_pData[0][bandIndex] + ss*pInputSpectrum2.m_pData[0][bandIndex+1] +
    pConstant) / (pInputSpectrum1.m_pData[0][bandIndex-1] +
    pInputSpectrum1.m_pData[0][bandIndex] + pInputSpectrum1.m_pData[0][bandIndex+1] +
    pConstant)
687
688         for (frameIndex = statics->startFrameIdx  frameIndex <= statics->stopFrameIdx  frameIndex++)
    {
689             hulp = pPower
690             y = pow (x,hulp)
```

```
691             this->m_pData[frameIndex][bandIndex] *= y
692         }
693     }
694
695 }
696
697 void CBarkSpectrum::AudibleFreqRespCompensationExact (const CBarkSpectrum  &pInputSpectrum1,
698                                                       const  CBarkSpectrum  &pInputSpectrum2,
699                                                       XFLOAT                 pConstant)
700 {
701     XFLOAT    x
702
703     for (int bandIndex = 0  bandIndex < aNumberOfBands  bandIndex++) {
704         x = ( (pInputSpectrum2.m_pData[0][bandIndex] + pConstant) /
    (pInputSpectrum1.m_pData[0][bandIndex] + pConstant) )
705         for (int frameIndex = statics->startFrameIdx  frameIndex <= statics->stopFrameIdx
    frameIndex++)
706         {
707             this->m_pData[frameIndex][bandIndex] *= x
708         }
709
710     }
711 }
712
713 void CBarkSpectrum::AudibleNoiseRespCompensationOf (const CBarkSpectrum  &pInputSpectrum) {
714     int       frameIndex
715
716     int       bandIndex
717
718     for (bandIndex = 0  bandIndex < aNumberOfBands  bandIndex++) {
719         for (frameIndex = statics->startFrameIdx  frameIndex <= statics->stopFrameIdx  frameIndex++)
720         {
721             this->m_pData[frameIndex][bandIndex] -= pInputSpectrum.m_pData[0][bandIndex]
722             if ( this->m_pData[frameIndex][bandIndex] < 0.0)
723                 this->m_pData[frameIndex][bandIndex] = 0.0
724         }
725     }
726
727 }
728
729 void CBarkSpectrum::AudibleNoiseRespCompensationOfPartly (const CPOLQAData*     POLQAHandle,
730                                                           const CBarkSpectrum  &pInputSpectrum,
731                                                           XFLOAT                pFactor)
732 {
733
734     int  bandIndex
735
736     for (bandIndex = 0  bandIndex < aNumberOfBands  bandIndex++) {
737         for (frameIndex = statics->startFrameIdx  frameIndex <= statics->stopFrameIdx  frameIndex++)
    {
738             this->m_pData[frameIndex][bandIndex] -= pFactor*pInputSpectrum.m_pData[0][bandIndex]
739             if ( this->m_pData[frameIndex][bandIndex] < 0.0)
740                 this->m_pData[frameIndex][bandIndex] = 0.0
741         }
742     }
743
744 }
745
746 void CBarkSpectrum::AudibleNoiseRespCompensationOfPartly2(  const CBarkSpectrum  &pInputSpectrum,
747                                                             XFLOAT pFactor, XFLOAT pConstant)
748 {
749     XFLOAT      hulp1, hulp2, bandLow, bandHigh, hulp, hulpPow
750
751     bandLow = 2.0
752     bandHigh = 20.0
753     for(int bandIndex = 0  bandIndex < statics->aNumberOfBarkBands  bandIndex++)
754     {
755         hulp1 = 1.0
```

```
        hulp2 = 1.0
        if ( statics->aCentreOfBandBark[bandIndex] < bandLow) hulp1 = (3.0 -
statics->aCentreOfBandBark[bandIndex])
        if ( statics->aCentreOfBandBark[bandIndex] > bandHigh) hulp2 = (1.0 + 0.15*(bandHigh -
statics->aCentreOfBandBark[bandIndex]))
        hulp = pInputSpectrum.m_pData[0][bandIndex] - pConstant*hulp2
        if (hulp<0.0) hulp = 0.0
        hulpPow = 0.74*pow((pInputSpectrum.m_pData[0][bandIndex]+1.0),0.05)
        if (hulpPow>0.8) hulpPow=0.8

        for (int frameIndex = statics->startFrameIdx  frameIndex <= statics->stopFrameIdx
frameIndex++) {
            this->m_pData[frameIndex][bandIndex] -= pFactor*hulp*hulpPow*hulp1
            if ( this->m_pData[frameIndex][bandIndex] < 0.0)
                this->m_pData[frameIndex][bandIndex] = 0.0
        }
    }

}

void CBarkSpectrum::AudibleNoiseRespCompensationOfPartlyAdded ( CPOLQAData*             POLQAHandle,
                                                                const CBarkSpectrum
&pInputSpectrum,
                                                                XFLOAT                  pFactor)
{
    XFLOAT    *hulp2, bandLow, bandHigh

    SmartBufferPolqa SB(POLQAHandle, aNumberOfBands)
    XFLOAT *hulp = SB.Buffer

    bandLow = 2.0
    bandHigh = 14.0

    matbSet(1.0, hulp, aNumberOfBands)

    int endHulp1, startHulp2
    endHulp1 = 0
    while(statics->aCentreOfBandBark[endHulp1] < bandLow)
        endHulp1++

    startHulp2 = endHulp1
    while(statics->aCentreOfBandBark[startHulp2] < bandHigh)
        startHulp2++

    matbCopy(statics->aCentreOfBandBark, hulp, endHulp1)
    matbMpy1(-2.0, hulp, endHulp1)
    matbAdd1(5.0, hulp, endHulp1)

    int hulp2Length = aNumberOfBands - endHulp1
    hulp2 = hulp + startHulp2
    matbCopy(statics->aCentreOfBandBark + startHulp2, hulp2, hulp2Length)
    matbMpy1(0.4, hulp2, hulp2Length)
    matbAdd1(1.0-0.4*bandHigh, hulp2, hulp2Length)

    matbMpy1(pFactor, hulp, aNumberOfBands)

    matbMpy2(pInputSpectrum.m_pData[0], hulp, aNumberOfBands)

    for (int frameIndex = statics->startFrameIdx  frameIndex <= statics->stopFrameIdx  frameIndex++)
    {
        matbSub2(hulp, this->m_pData[frameIndex], aNumberOfBands)

        matbThresh1(this->m_pData[frameIndex], aNumberOfBands, 0.0, MAT_LT)
    }
}

void CBarkSpectrum::AudibleNoiseRespCompensationOfPartly2Added(const CBarkSpectrum  &pInputSpectrum,
    XFLOAT pFactor, XFLOAT pConstant)
```

```
{
    XFLOAT      hulp1, hulp2, bandLow, bandHigh, hulp, hulpPow

    bandLow = 4.0
    bandHigh = 16.0
    for (int bandIndex = 0  bandIndex < aNumberOfBands  bandIndex++)
    {
        hulp1 = 1.0
        hulp2 = 1.0
        if (statics->aCentreOfBandBark[bandIndex] < bandLow)
            hulp1 = (9.0 - 2.0*statics->aCentreOfBandBark[bandIndex])
        if (hulp1>3.0) hulp1 = 3.0
        if (statics->aCentreOfBandBark[bandIndex] > bandHigh)

            hulp2 = 1.0 + 0.5*(statics->aCentreOfBandBark[bandIndex] - bandHigh)
        hulp = pInputSpectrum.m_pData[0][bandIndex] - pConstant
        if (hulp<0.0)
            hulp = 0.0
        hulpPow = pow((pInputSpectrum.m_pData[0][bandIndex] + 1.0), 0.08)
        if (hulpPow>1.3)
            hulpPow = 1.3

        for (int frameIndex = statics->startFrameIdx  frameIndex <= statics->stopFrameIdx
frameIndex++) {
            this->m_pData[frameIndex][bandIndex] -= pFactor*hulp*hulp1*hulp2*hulpPow
            if ( this->m_pData[frameIndex][bandIndex] < 0.0)
                this->m_pData[frameIndex][bandIndex] = 0.0
        }
    }

}

void CBarkSpectrum::FrequencyWarpingOf(const CPOLQAData *POLQAHandle, CHzSpectrum const
&pCHzSpectrum, XFLOAT PitchRatio)
{

    PitchRatio = 1.0
    int hzBandIndex

    SmartBufferPolqa SB(POLQAHandle, aNumberOfBands)
    XFLOAT *sum = SB.Buffer

    const int nrOfHzBands = statics->aNumberOfHzBands
    int deltaHzBands

    int NextUpperLineRef
    int NextUpperLineModified

    for(int frameIndex = statics->startFrameIdx  frameIndex <= statics->stopFrameIdx  frameIndex++)
    {
        NextUpperLineRef = 0
        NextUpperLineModified = 0
        hzBandIndex = 0

        for (int barkBandIndex = 0  barkBandIndex < aNumberOfBands  barkBandIndex++)
        {
            NextUpperLineRef += statics->aNumberOfHzBandsInBarkBand[barkBandIndex]
            NextUpperLineModified = (int)((XFLOAT)NextUpperLineRef * PitchRatio + 0.5)

            deltaHzBands = (NextUpperLineModified < nrOfHzBands ? NextUpperLineModified :
nrOfHzBands) - hzBandIndex

            sum[barkBandIndex] = matSum(pCHzSpectrum.m_pData[frameIndex] + hzBandIndex,
deltaHzBands)
            hzBandIndex += deltaHzBands

        }
        matbMpy2(statics->aPowerDensityCorrectionFactor, sum, aNumberOfBands)
```

```
884
885         matbMpy1(statics->aCalibrationFactorSp, sum, aNumberOfBands)
886
887         matbCopy(sum, this->m_pData[frameIndex], aNumberOfBands)
888     }
889
890 }
891
892 void CBarkSpectrum::IntensityWarpingOf(const CPOLQAData *POLQAHandle, const CBarkSpectrum
    &pInputSpectrum)
893 {
894     XFLOAT    *hulp, *temp
895     XFLOAT    *modifiedZwickerPower
896
897     const int aNumberOfBarkBands = statics->aNumberOfBarkBands
898
899     SmartBufferPolqa SB1(POLQAHandle, aNumberOfBarkBands)
900     temp = SB1.Buffer
901     SmartBufferPolqa SB2(POLQAHandle, aNumberOfBarkBands)
902     hulp = SB2.Buffer
903
904     int h1Idx, h2Idx
905     for (h1Idx = aNumberOfBarkBands -1  h1Idx >= 0    && statics->aCentreOfBandBark[h1Idx] >=
    (XFLOAT) 2.0  h1Idx--)
906     for (h2Idx = h1Idx+1  h2Idx < aNumberOfBarkBands && statics->aCentreOfBandBark[h2Idx] <=
    (XFLOAT)22.0  h2Idx++)
907
908     for (int frameIndex = statics->startFrameIdx  frameIndex <= statics->stopFrameIdx  frameIndex++)
909     {
910         matbCopy((pInputSpectrum.m_pData[frameIndex]), temp, aNumberOfBarkBands)
911
912         matbAdd1(600.0, temp, aNumberOfBarkBands)
913         matbPow2(temp, 0.009, hulp, aNumberOfBarkBands)
914         matbThresh1(hulp, aNumberOfBarkBands, 1.1, MAT_GT)
915
916         matbSet(1.0, temp, aNumberOfBarkBands)
917
918         for (bandIndex = 0  bandIndex < aNumberOfBarkBands  bandIndex++)
919         {
920             if(aCentreOfBandBark.m_pData[bandIndex] < (XFLOAT) 2.0)
921                 temp[bandIndex] =  -0.03*aCentreOfBandBark.m_pData[bandIndex] + 1.06

922             else
923             {
924                 if(aCentreOfBandBark.m_pData[bandIndex] > (XFLOAT) 22.0)
925                     temp[bandIndex] =  0.2*(aCentreOfBandBark.m_pData[bandIndex]-22.0) + 1.0

926             }
927         }
928
929         modifiedZwickerPower = temp
930
931         matbMpy2(hulp, modifiedZwickerPower, aNumberOfBarkBands)
932
933         matbMpy1(0.22, modifiedZwickerPower, aNumberOfBarkBands)
934
935         matbZero(this->m_pData[frameIndex], aNumberOfBarkBands)
936
937         XFLOAT *pInputSpec = (pInputSpectrum.m_pData[frameIndex])
938
939         for (int bandIndex = 0  bandIndex < aNumberOfBarkBands  bandIndex++)
940         {
941             if (pInputSpec[bandIndex]  > statics->aAbsoluteThresholdPower[bandIndex])
942             {
943                 this->m_pData[frameIndex][bandIndex] = (XFLOAT) (pow
    (statics->aAbsoluteThresholdPower[bandIndex] / 0.5, modifiedZwickerPower[bandIndex])
944                         * (pow (0.5 + 0.5 * pInputSpec[bandIndex] /
    statics->aAbsoluteThresholdPower[bandIndex],
```

```
    modifiedZwickerPower[bandIndex]) - 1.0))
945             }
946         }
947     }
948     (*this) *= statics->aCalibrationFactorSl
949
950 }
951
952 void CBarkSpectrum::IntensityWarpingOfSpeechLQ(const CPOLQAData *POLQAHandle, const CBarkSpectrum
    &pInputSpectrum, XFLOAT maxFreqBark)
953 {
954     XFLOAT    *hulp, *temp
955     XFLOAT    *modifiedZwickerPower
956
957     const int aNumberOfBarkBands = statics->aNumberOfBarkBands
958
959     SmartBufferPolqa SB1(POLQAHandle, aNumberOfBarkBands)
960     temp = SB1.Buffer
961     SmartBufferPolqa SB2(POLQAHandle, aNumberOfBarkBands)
962     hulp = SB2.Buffer
963
964     int h1Idx, h2Idx
965     for (h1Idx = aNumberOfBarkBands - 1  h1Idx >= 0 && statics->aCentreOfBandBark[h1Idx] >= (XFLOAT)
    2.0  h1Idx--)
966     for (h2Idx = h1Idx + 1  h2Idx < aNumberOfBarkBands && statics->aCentreOfBandBark[h2Idx] <=
    (XFLOAT)22.0  h2Idx++)
967
968     for (int frameIndex = statics->startFrameIdx  frameIndex <= statics->stopFrameIdx  frameIndex++)
969     {
970         matbCopy((pInputSpectrum.m_pData[frameIndex]), temp, aNumberOfBarkBands)
971
972         matbAdd1(600.0, temp, aNumberOfBarkBands)
973         if (maxFreqBark < 22.0) {
974             matbPow2(temp, 0.011, hulp, aNumberOfBarkBands)
975         } else {
976             matbPow2(temp, 0.008, hulp, aNumberOfBarkBands)
977         }
978         matbThresh1(hulp, aNumberOfBarkBands, 1.2, MAT_GT)
979
980         matbSet(1.0, temp, aNumberOfBarkBands)
981
982         for (int bandIndex = 0  bandIndex <= h1Idx  bandIndex++)
983             temp[bandIndex] = -0.03*statics->aCentreOfBandBark[bandIndex] + 1.06
984         for (int bandIndex = aNumberOfBarkBands - 1  bandIndex >= h2Idx  bandIndex--)
985             temp[bandIndex] = 0.2*(statics->aCentreOfBandBark[bandIndex] - 22.0) + 1.0
986
987         modifiedZwickerPower = temp
988
989         matbMpy2(hulp, modifiedZwickerPower, aNumberOfBarkBands)
990
991         matbMpy1(0.22, modifiedZwickerPower, aNumberOfBarkBands)
992
993         matbZero(this->m_pData[frameIndex], aNumberOfBarkBands)
994
995         XFLOAT *pInputSpec = (pInputSpectrum.m_pData[frameIndex])
996
997         for (int bandIndex = 0  bandIndex < aNumberOfBarkBands  bandIndex++)
998         {
999             if (pInputSpec[bandIndex]  > statics->aAbsoluteThresholdPower[bandIndex])
1000            {
1001                this->m_pData[frameIndex][bandIndex] =
    (XFLOAT)(pow(statics->aAbsoluteThresholdPower[bandIndex] / 0.5,
    modifiedZwickerPower[bandIndex])
1002                    * (pow(0.5 + 0.5 * pInputSpec[bandIndex] /
    statics->aAbsoluteThresholdPower[bandIndex], modifiedZwickerPower[bandIndex]) -
    1.0))
1003            }
1004        }
```

```
1005      }
1006      (*this) *= statics->aCalibrationFactorSl
1007
1008  }
1009
1010  void CBarkSpectrum::InhibitionSpeechLQ(const CPOLQAData *POLQAHandle, const CBarkSpectrum
      &pInputSpectrum, XFLOAT maxFreqBark)
1011  {
1012      XFLOAT    temp, hulp
1013
1014      const int aNumberOfBarkBands = statics->aNumberOfBarkBands
1015      int frameIndex, bandIndex
1016
1017      frameIndex = statics->startFrameIdx
1018      XFLOAT *pInputSpec = pInputSpectrum.m_pData[frameIndex]
1019      for (bandIndex = 0  bandIndex < aNumberOfBarkBands  bandIndex++)
      this->m_pData[frameIndex][bandIndex] = pInputSpec[bandIndex]
1020
1021      for (frameIndex = (statics->startFrameIdx + 1)  frameIndex <= statics->stopFrameIdx
      frameIndex++) {
1022          XFLOAT *pInputSpec = pInputSpectrum.m_pData[frameIndex]
1023          XFLOAT *pInputSpecOld = pInputSpectrum.m_pData[frameIndex - 1]
1024
1025          for (bandIndex = 0  bandIndex < aNumberOfBarkBands  bandIndex++) {
1026              if ((bandIndex>15) && (pInputSpecOld[bandIndex]>2.0)) {
1027                  hulp = 0.0015*(1.0*bandIndex - 15.0)
1028                  if (hulp > 0.02) hulp = 0.02
1029                  temp = (pInputSpec[bandIndex] - hulp*pInputSpecOld[bandIndex])
1030                  if (temp < 0.0) temp = 0.0
1031              }
1032              else {
1033                  temp = pInputSpec[bandIndex]
1034              }
1035              this->m_pData[frameIndex][bandIndex] = temp
1036
1037          }
1038      }
1039
1040  }
1041
1042  void CBarkSpectrum::InhibitionIandM(const CPOLQAData *POLQAHandle, const CBarkSpectrum
      &pInputSpectrum, XFLOAT maxFreqBark)
1043  {
1044      XFLOAT    temp, hulp
1045
1046      const int aNumberOfBarkBands = statics->aNumberOfBarkBands
1047      int frameIndex, bandIndex
1048
1049      frameIndex = statics->startFrameIdx
1050      XFLOAT *pInputSpec = pInputSpectrum.m_pData[frameIndex]
1051      for (bandIndex = 0  bandIndex < aNumberOfBarkBands  bandIndex++)
      this->m_pData[frameIndex][bandIndex] = pInputSpec[bandIndex]
1052
1053      for (frameIndex = (statics->startFrameIdx + 1)  frameIndex <= statics->stopFrameIdx
      frameIndex++) {
1054          XFLOAT *pInputSpec = pInputSpectrum.m_pData[frameIndex]
1055          XFLOAT *pInputSpecOld = pInputSpectrum.m_pData[frameIndex - 1]
1056
1057          for (bandIndex = 0  bandIndex < aNumberOfBarkBands  bandIndex++) {
1058              if ((bandIndex>15) && (pInputSpecOld[bandIndex]>2.0)) {
1059                  hulp = 0.0015*(1.0*bandIndex - 15.0)
1060                  if (hulp > 0.02) hulp = 0.02
1061                  temp = (pInputSpec[bandIndex] - hulp*pInputSpecOld[bandIndex])
1062                  if (temp < 0.0) temp = 0.0
1063              }
1064              else {
1065                  temp = pInputSpec[bandIndex]
1066              }
```

```
            this->m_pData[frameIndex][bandIndex] = temp

        }
    }

}

XFLOAT CBarkSpectrum::SpectralFlatness(int pFrameIndex) const
{

    XFLOAT    integral = 0

    XFLOAT integralLog = 0

    int          count = 0

    for (int bandIndex = 1  bandIndex < aNumberOfBands  bandIndex++) {
        XFLOAT h = this->m_pData[pFrameIndex][bandIndex]
        if (h > 0.0) {
            integral += h

            integralLog += log(h)

            count++
        }
    }

    if (count == 0) {
        return 1
    }

    integral /= count

    integralLog /= count

    if (integral == 0)
    {
        return 1
    }

    XFLOAT result = exp(integralLog) / integral

    return (XFLOAT)result
}

XFLOAT CBarkSpectrum::SpectralFlatnessIandM (int pFrameIndex) const
{

    XFLOAT    integral = 0

    XFLOAT integralLog = 0

    int          count = 0

    for (int bandIndex = 1  bandIndex < aNumberOfBands  bandIndex++) {
        XFLOAT h = this->m_pData[pFrameIndex][bandIndex]
        if (h > 0.02) {
            integral += h

            integralLog += log (h)

            count++
        }
    }

    if (count == 0) {
        return 1
    }
```

```
1135
1136      integral /= count
1137
1138      integralLog /= count
1139
1140      if (integral == 0)
1141      {
1142          return 1
1143      }
1144
1145      XFLOAT result = exp (integralLog) / integral
1146
1147      return (XFLOAT) result
1148  }
1149
1150  XFLOAT CBarkSpectrum::SpectralFlatnessPower (int pFrameIndex) const
1151  {
1152      XFLOAT    integral = 0, hulp
1153      XFLOAT integralLog = 0
1154      int          count = 0
1155
1156      for (int bandIndex = 4  bandIndex < aNumberOfBands  bandIndex++) {
1157          XFLOAT h = this->m_pData[pFrameIndex][bandIndex]
1158          if (h > 1.0) {
1159              hulp = bandIndex -54.0
1160              if (hulp < 1.0) hulp = 1.0
1161              hulp = pow(hulp, 2.0)
1162              integral += h/hulp
1163              integralLog += log (h)
1164              count++
1165          }
1166      }
1167
1168      if (count == 0) {
1169          return 1
1170      }
1171
1172      integral /= count
1173      integralLog /= count
1174
1175      if (integral == 0)
1176      {
1177          return 1
1178      }
1179
1180      XFLOAT result = exp (integralLog) / integral
1181
1182      return (XFLOAT) result
1183  }
1184
1185  XFLOAT CBarkSpectrum::Integral (CPOLQAData *POLQAHandle, int pFrameIndex)
1186  {
1187      SmartBufferPolqa SB(POLQAHandle, aNumberOfBands)
1188      XFLOAT *temp = SB.Buffer
1189
1190      const int length = aNumberOfBands - 1
1191
1192      matbMpy3(this->m_pData[pFrameIndex] + 1, statics->aWidthOfBandBark + 1, temp, length)
1193
1194      return matSum(temp, length)
1195  }
1196
1197  XFLOAT CBarkSpectrum::IntegralLpOverBandRange (CPOLQAData *POLQAHandle, int pFrameIndex, XFLOAT
      pPower, int bandIdxLow, int bandIdxHigh) {
1198      XFLOAT   result
1199
1200      int      bandIndex
1201      result = (XFLOAT) 0
```

```
1202      for (bandIndex = 0  bandIndex < aNumberOfBands  bandIndex++) {
1203          if ( (bandIndex >= bandIdxLow) && (bandIndex <= bandIdxHigh) )
1204              result += pow ( (XFLOAT)(this->m_pData[pFrameIndex][bandIndex] *
      aWidthOfBandBark.m_pData[bandIndex]), pPower)
1205      }
1206      result = pow ((XFLOAT)result, (XFLOAT)(1.0/pPower))
1207      return (XFLOAT) result
1208
1209  }
1210
1211  XFLOAT CBarkSpectrum::SumLpOverBandRange (int pFrameIndex, XFLOAT pPower, XFLOAT pBandLow, XFLOAT
      pBandHigh )
1212  {
1213      XFLOAT   result
1214      result = (XFLOAT) 0
1215      for (int bandIndex = 0  bandIndex < aNumberOfBands  bandIndex++)
1216      {
1217          if ( (statics->aCentreOfBandBark[bandIndex] > pBandLow) &&
      (statics->aCentreOfBandBark[bandIndex] < pBandHigh) )
1218              result += pow ( (XFLOAT)this->m_pData[pFrameIndex][bandIndex], pPower)
1219      }
1220      result = pow ((XFLOAT)result, (XFLOAT)((XFLOAT)1.0/pPower))
1221      return result
1222  }
1223
1224  void CBarkSpectrum::MultiplyWithOverBandRange(int pFrameIndex, XFLOAT pFactor, XFLOAT pBandLow,
      XFLOAT pBandHigh)
1225  {
1226      int startBand = 0
1227      int stopBand
1228      int length
1229
1230      while (startBand < aNumberOfBands && statics->aCentreOfBandBark[startBand] <= pBandLow)
1231          startBand++
1232
1233      stopBand = startBand
1234
1235      while (stopBand < aNumberOfBands && statics->aCentreOfBandBark[stopBand] <= pBandHigh)
1236          stopBand++
1237
1238      length = stopBand - startBand
1239
1240      if (length>0)
1241          matbMpy1(pFactor, this->m_pData[pFrameIndex] + startBand, length)
1242  }
1243
1244  XFLOAT CBarkSpectrum::IntegralLowNarrowband (CPOLQAData *POLQAHandle, int pFrameIndex)
1245  {
1246      XFLOAT   hulp, result
1247
1248      result = (XFLOAT) 0.0
1249      for (int bandIndex = 0  bandIndex < aNumberOfBands  bandIndex++)
1250      {
1251          if ( (statics->aCentreOfBandBark[bandIndex] < 11.0) &&
      (statics->aCentreOfBandBark[bandIndex] > 2.0) )
1252          {
1253              hulp = this->m_pData[pFrameIndex][bandIndex] * statics->aWidthOfBandBark[bandIndex]
1254              hulp *=  ((20.0 - statics->aCentreOfBandBark[bandIndex])/9.0)
1255              result += hulp
1256          }
1257      }
1258      return result
1259  }
1260
1261  XFLOAT CBarkSpectrum::IntegralHighNarrowband (int pFrameIndex) {
1262      XFLOAT   hulp, result
1263
1264      result = (XFLOAT) 0.0
```

```
    for (int bandIndex = 0  bandIndex < aNumberOfBands  bandIndex++) {
        if ( (statics->aCentreOfBandBark[bandIndex] > 7.0) && (statics->aCentreOfBandBark[bandIndex]
< 17.0) )
        {
            hulp = this->m_pData[pFrameIndex][bandIndex] * statics->aWidthOfBandBark[bandIndex]
            hulp *=  ((statics->aCentreOfBandBark[bandIndex] - 2.0)/5.0)
            result += hulp
        }
    }
    return result
}

XFLOAT CBarkSpectrum::IntegralLow2 (int pFrameIndex, int pListeningConditionChoice) {
    XFLOAT   hulp, result

    result = (XFLOAT) 0.0
    for (int bandIndex = 0  bandIndex < aNumberOfBands  bandIndex++) {
        if (statics->aCentreOfBandBark[bandIndex] < 11.0) {
            hulp = this->m_pData[pFrameIndex][bandIndex] * statics->aWidthOfBandBark[bandIndex]
            hulp *=  ((19.0 - statics->aCentreOfBandBark[bandIndex])/8.0)
            if (hulp>1.0) result += hulp
        }
    }
    return result
}

XFLOAT CBarkSpectrum::IntegralHigh2 (int pFrameIndex, int pListeningConditionChoice) {
    XFLOAT   hulp, result

    result = (XFLOAT) 0.0
    for (int bandIndex = 0  bandIndex < aNumberOfBands  bandIndex++) {
        if (statics->aCentreOfBandBark[bandIndex] > 6.0) {
            hulp = this->m_pData[pFrameIndex][bandIndex] * statics->aWidthOfBandBark[bandIndex]
            hulp *=  ((statics->aCentreOfBandBark[bandIndex]-2.0)/6.0)
            if (hulp>1.0) result += hulp
        }
    }
    return result
}

XFLOAT CBarkSpectrum::IntegralHigh3 (int pFrameIndex)
{
    XFLOAT   hulp, result

    result = (XFLOAT) 0.0
    for (int bandIndex = 0  bandIndex < aNumberOfBands  bandIndex++)
    {
        if (statics->aCentreOfBandBark[bandIndex] > 17.0)
        {
            hulp = this->m_pData[pFrameIndex][bandIndex] * statics->aWidthOfBandBark[bandIndex]
            hulp *= (statics->aCentreOfBandBark[bandIndex]/5.0)
            result += hulp
        }
    }
    return result
}

XFLOAT CBarkSpectrum::IntegralLowFrameLoud (int pFrameIndex, int pListeningConditionChoice) {
    XFLOAT   hulp, result

    result = (XFLOAT) 0.0
    for (int bandIndex = 0  bandIndex < aNumberOfBands  bandIndex++) {
        if (statics->aCentreOfBandBark[bandIndex] < 10.0) {
            hulp = this->m_pData[pFrameIndex][bandIndex] * statics->aWidthOfBandBark[bandIndex]
            hulp *=  ((18.0 - statics->aCentreOfBandBark[bandIndex])/8.0)
            if (hulp>1.0) result += hulp
        }
    }
```

```
1332      return result
1333  }
1334
1335  XFLOAT CBarkSpectrum::IntegralHighFrameLoud (int pFrameIndex, int pListeningConditionChoice) {
1336      XFLOAT   hulp, result
1337
1338      result = (XFLOAT) 0.0
1339      for (int bandIndex = 0  bandIndex < aNumberOfBands  bandIndex++) {
1340          if ((statics->aCentreOfBandBark[bandIndex] > 10.0) && (statics->aCentreOfBandBark[bandIndex]
      < 19.0) ) {
1341              hulp = this->m_pData[pFrameIndex][bandIndex] * statics->aWidthOfBandBark[bandIndex]
1342              hulp *=  ((statics->aCentreOfBandBark[bandIndex]-2.0)/12.0)
1343              if (hulp>1.0) result += hulp
1344          }
1345      }
1346      return result
1347  }
1348
1349  XFLOAT CBarkSpectrum::TotalAudible(const CPOLQAData* POLQAHandle, int pFrameIndex, XFLOAT pFactor)
1350  {
1351      XFLOAT  threshold, result
1352
1353      result = 0.
1354      for (int bandIndex = 1  bandIndex < aNumberOfBands  bandIndex++)
1355      {
1356          threshold = pFactor * statics->aAbsoluteThresholdPower[bandIndex]
1357          if (this->m_pData[pFrameIndex][bandIndex] > threshold)
1358          {
1359              result += this->m_pData[pFrameIndex][bandIndex]
1360          }
1361      }
1362      return (XFLOAT) result
1363  }
1364
1365  XFLOAT CBarkSpectrum::Total (int pFrameIndex, XFLOAT pFrequencyLow, XFLOAT pFrequencyHigh)
1366  {
1367
1368      int     bandIndex
1369      XFLOAT  result
1370
1371      result = 0.
1372      for (bandIndex = 1  bandIndex < aNumberOfBands  bandIndex++) {
1373          XFLOAT frequency = aCentreOfBandHz [bandIndex]
1374          if ((frequency >= pFrequencyLow) && (frequency <= pFrequencyHigh))
1375          {
1376              result += (*this) [pFrameIndex] [bandIndex]
1377          }
1378      }
1379      return (XFLOAT) result
1380
1381  }
1382
1383  void CBarkSpectrum::Orthogonalize(const CIntArray &pSilent)
1384  {
1385      for (int frameIndex = statics->startFrameIdx  frameIndex <= statics->stopFrameIdx  frameIndex++)
1386      {
1387          if (pSilent.m_pData[frameIndex])
1388          {
1389              matbZero(this->m_pData[frameIndex] + 1, aNumberOfBands - 1)
1390          }
1391      }
1392  }
1393
1394  XFLOAT CBarkSpectrum::FrameCorrelationTime (int pFrameIndex, int pNumberOfWindows) const
1395  {
1396      XFLOAT   sumXY, sumX, sumY, sumX2, sumY2, result
1397      int      count
1398      count = 0
```

```
1399      sumXY = 0.0
1400      sumX = 0.0
1401      sumY = 0.0
1402      sumX2 = 0.0
1403      sumY2 = 0.0
1404
1405      XFLOAT X, Y, f
1406      for (int bandIndex = 1  bandIndex < aNumberOfBands  bandIndex++) {
1407          f = statics->aCentreOfBandHz[bandIndex]
1408
1409          if ((f > 100) && (f < 7000)) {
1410              X = this->m_pData[pFrameIndex - 2][bandIndex]
1411              Y = this->m_pData[pFrameIndex][bandIndex]
1412              sumXY += X*Y
1413              sumX += X
1414              sumY += Y
1415              sumX2 += X*X
1416              sumY2 += Y*Y
1417              count++
1418          }
1419
1420      }
1421      if ( count>2 && sumX>0.0 && sumY>0.0 )
1422          result = (count*sumXY-sumX*sumY)/ sqrt((count*sumX2-sumX*sumX)*(count*sumY2-sumY*sumY))
1423      else
1424          result = 0.0
1425
1426      return result
1427  }
1428
1429  void CBarkSpectrum::ComputeLpWeights (CPOLQAData *POLQAHandle, XFLOAT pBasePower, XFLOAT
      pIncrementPower, int pNumberOfPowers, CDoubleArray pDisturbance [])
1430  {
1431
1432      SmartBufferPolqa SB1(POLQAHandle, pNumberOfPowers)
1433      XFLOAT *sum = SB1.Buffer
1434
1435      SmartBufferPolqa SB2(POLQAHandle, aNumberOfBands)
1436      XFLOAT *prod = SB2.Buffer
1437
1438      pIncrementPower *= 0.5
1439
1440      XFLOAT base, factor, totalWeight, z
1441
1442      totalWeight = matSum(aWidthOfBandBark.m_pData+1, aNumberOfBands-1)
1443
1444      for (int frameIndex = statics->startFrameIdx  frameIndex <= statics->stopFrameIdx  frameIndex++)
1445      {
1446          matbZero(sum, pNumberOfPowers)
1447
1448          matbAbs2(this->m_pData[frameIndex] + 1, prod + 1, aNumberOfBands-1)
1449
1450          matbMpy2(aWidthOfBandBark.m_pData + 1, prod + 1, aNumberOfBands-1)
1451
1452          for (int bandIndex = 1  bandIndex < aNumberOfBands  bandIndex++)
1453          {
1454              base = pow (prod[bandIndex], pBasePower)
1455              factor = pow (prod[bandIndex], pIncrementPower)
1456
1457              z = base
1458              for (int i = 0  i < pNumberOfPowers  i++)
1459              {
1460                  sum [i] += z
1461                  z *= factor
1462              }
1463          }
1464
1465          matbMpy1(1/totalWeight, sum, pNumberOfPowers)
```

```
1466
1467        for (int i = 0  i < pNumberOfPowers  i++)
1468        {
1469            sum [i] = pow (sum [i], 1./(pBasePower + i * pIncrementPower))
1470        }
1471
1472        matbMpy1(totalWeight, sum, pNumberOfPowers)
1473
1474        for (int i = 0  i < pNumberOfPowers  i++)
1475        {
1476            pDisturbance[i].m_pData[frameIndex] = sum[i]
1477        }
1478    }
1479 }
1480
1481 void CBarkSpectrum::MultiplyWithAsymmetryFactorAddOf (const CBarkSpectrum
    &pOriginalPitchPowerDensity,
1482                                                       const CBarkSpectrum
    &pDistortedPitchPowerDensity,
1483                                                       int
    pListeningConditionChoice,
1484                                                       XFLOAT pNoiseContrastMax1, const CIntArray
      &pSilent, XFLOAT
    pDistortedSilencePowerMeanCompensation, XFLOAT
    maxFreqBark)
1485 {
1486
1487     long  frameIndex
1488     int   i
1489     XFLOAT hulpAsymOrg, hulpAsymDis, ratioHulp,ratioHulpSilence, ratio, h, bandHulpMax
1490     CNewStdString s
1491
1492     for (frameIndex = (statics->startFrameIdx+1)  frameIndex <= statics->stopFrameIdx  frameIndex++)
    {
1493         hulpAsymOrg = 0.0
1494         hulpAsymDis = 0.0
1495         for (i = 0  i < aNumberOfBarkBands  i++) {
1496             if (aCentreOfBandBark.m_pData[i] > 15.0) {
1497                 bandHulpMax = pow((aCentreOfBandBark.m_pData[i]-14.0),0.5)
1498             } else {
1499                 bandHulpMax = 1.0
1500             }
1501             hulpAsymOrg += bandHulpMax*pOriginalPitchPowerDensity.m_pData[frameIndex][i]
1502             hulpAsymDis += bandHulpMax*pDistortedPitchPowerDensity.m_pData[frameIndex][i]
1503         }
1504         ratioHulp = pow( ((hulpAsymDis+6.0e6)/(hulpAsymOrg+6.0e6)), 5.0/pow(pNoiseContrastMax1,0.4))
1505
1506         hulpAsymOrg = 0.0
1507         hulpAsymDis = 0.0
1508         for (i = 0  i < aNumberOfBarkBands  i++) {
1509             if (aCentreOfBandBark.m_pData[i] > 15.0) {
1510                 bandHulpMax = pow((aCentreOfBandBark.m_pData[i] - 14.0), 0.5)
1511             } else {
1512                 bandHulpMax = 1.0
1513             }
1514             hulpAsymOrg += bandHulpMax*pOriginalPitchPowerDensity.m_pData[frameIndex][i]
1515             hulpAsymDis += bandHulpMax*pDistortedPitchPowerDensity.m_pData[frameIndex][i]
1516         }
1517         ratioHulpSilence = pow( ((hulpAsymDis+6.0e6)/(hulpAsymOrg+6.0e6)),
    5.0/pow(pNoiseContrastMax1,0.4))
1518
1519         for (i = 0  i < aNumberOfBarkBands  i++)
1520         {
1521             if (aCentreOfBandBark.m_pData[i] > 18.0)
1522                 bandHulpMax = pow((aCentreOfBandBark.m_pData[i] - 17.0),0.9)
1523             else
1524                 bandHulpMax = 1.0
1525
```

```
1526            if (aCentreOfBandBark.m_pData[i] < 4.0)
1527                bandHulpMax = pow((5.0 - aCentreOfBandBark.m_pData[i]),0.7)
1528
1529            if (pSilent.m_pData[frameIndex])
1530            {
1531                ratio = (pDistortedPitchPowerDensity.m_pData[frameIndex][i]  + (XFLOAT) 200.0)
1532                          / (pOriginalPitchPowerDensity.m_pData[frameIndex][i] + (XFLOAT) 200.0)
1533                h =  pow (ratio, 1.2)/(ratioHulpSilence)
1534                if (h < (XFLOAT) 1.0)
1535                    this->m_pData[frameIndex][i] *= pow(h,1.2)
1536                else
1537                {
1538                    if (h > ( (XFLOAT) 6.5*bandHulpMax*pDistortedSilencePowerMeanCompensation) )
1539                        h = (XFLOAT) 6.5*bandHulpMax*pDistortedSilencePowerMeanCompensation
1540                    this->m_pData[frameIndex][i] *= h
1541                }
1542            }
1543            else
1544            {
1545                ratio = (pDistortedPitchPowerDensity.m_pData[frameIndex][i]  + (XFLOAT) 2000.0)
1546                          / (pOriginalPitchPowerDensity.m_pData[frameIndex][i] + (XFLOAT) 2000.0)
1547                h =  pow (ratio, 1.2)/(ratioHulp)
1548                if (h < (XFLOAT) 1.0)
1549                    this->m_pData[frameIndex][i] *= pow(h,1.2)
1550                else
1551                {
1552                    if (h > ( (XFLOAT) 7.0*bandHulpMax*pDistortedSilencePowerMeanCompensation) )
1553                        h = (XFLOAT) 7.0*bandHulpMax*pDistortedSilencePowerMeanCompensation
1554                    this->m_pData[frameIndex][i] *= h
1555                }
1556            }
1557        }
1558    }
1559
1560 }
1561
1562 void CBarkSpectrum::DifferenceOfBandlimited (const CPsqmArray &pInputCPsqmArray1, const CPsqmArray
      &pInputCPsqmArray2)
1563 {
1564     int frameIndex, nu
1565     XFLOAT *pInputCPsqmA1, *pInputCPsqmA2
1566
1567     for (frameIndex = statics->startFrameIdx  frameIndex <= statics->stopFrameIdx  frameIndex++)
1568     {
1569         pInputCPsqmA1 = (pInputCPsqmArray1.m_pData[frameIndex])
1570         pInputCPsqmA2 = (pInputCPsqmArray2.m_pData[frameIndex])
1571
1572         matbSub3(pInputCPsqmA1, pInputCPsqmA2, this->m_pData[frameIndex], aNumberOfBands)
1573
1574         for(int band = 0  band < aDifferenceBarkScaling.GetSize()   band++)
1575             this->m_pData[frameIndex][band] *= aDifferenceBarkScaling[band]
1576
1577         for (nu = 0  nu < aNumberOfBands  nu++)
1578             if ( aCentreOfBandBark.m_pData[nu] > 19.0) {
1579                 if ( aCentreOfBandBark.m_pData[nu] > 22.0) {
1580                     this->m_pData[frameIndex][nu] *= 0.0
1581                 } else {
1582                     this->m_pData[frameIndex][nu] /= (aCentreOfBandBark.m_pData[nu]-18.0)
1583                 }
1584             }
1585
1586     }
1587 }
1588
1589 }
1590
1591
```