```
    typedef double XFLOAT;
    typedef double OTA_FLOAT;

extern BOOL         gBatchMode;

namespace POLQAV2
{

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, 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;

    result = matMax(m_pData[pFrameIndex], aNumberOfBands);

    if(result < lowerBound)
        result = lowerBound;

    return result;
```

```
}

void CSignal::MultiplyWith (int pFrameIndex, XFLOAT pFactor)
{
    matbMpy1(pFactor, m_pData[pFrameIndex], aNumberOfBands);
}

void CPsqmArray::DifferenceOf(const CPsqmArray &pInputCPsqmArray1, const CPsqmArray
&pInputCPsqmArray2)
{
    XFLOAT *pInputCPsqmA1, *pInputCPsqmA2;

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

        pInputCPsqmA1 = pInputCPsqmArray1.m_pData[frameIndex];
        pInputCPsqmA2 = pInputCPsqmArray2.m_pData[frameIndex];

        matbSub3(pInputCPsqmA1, pInputCPsqmA2, m_pData[frameIndex], aNumberOfBands);

        for(int band = 0; band < statics->aDifferenceBarkScalingLen ; band++)
            m_pData[frameIndex][band] *= statics->aDifferenceBarkScaling[band];

    }
}

void CPsqmArray::FractionOf (XFLOAT pConstant, int frame)
{
    if(AlmostEqualUlpsFinal((float)pConstant, 0.0f))
        matbZero(m_pData[frame], aNumberOfBands);
    else
        matbMpy1(pConstant, m_pData[frame], aNumberOfBands);
}

void CPsqmArray::MaskWith (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)
{
    int StartOfFrame = FrameIndex * Windowsize/2;
    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++)
        {
            int utt = GetUtteranceForFrame(pStartSampleUtterance, pStopSampleUtterance,
DelayUtterance, frameIndex, Windowsize);
            if (utt<0)
                matbZero(this->m_pData[frameIndex], aNumberOfBands);
            else
                STFTPowerSpectrumOf (POLQAHandle, pTimeSeries, frameIndex,
DelayUtterance[utt]);
        }
    }

}

void CPsqmArray::STFTPowerSpectrumOf(   CPOLQAData *POLQAHandle,
                                        const CTimeSeries &pTimeSeries,
                                        int pFrameIndex,
                                        int Delay
                                    )
{

    XFLOAT              a, b;

    int                 n = pTimeSeries. GetFrameLength ();
    XFLOAT *x;

    if (POLQAHandle->STFTBufferLength != n)
    {
        if (POLQAHandle->STFTBufferLength > 0)
            matFree(POLQAHandle->STFTBuffer);

        POLQAHandle->STFTBuffer = (XFLOAT*)matMalloc((n + 2)*sizeof(XFLOAT));

        int p = 1;
        POLQAHandle->STFTBufferLengthLog2 = 0;
        while (p < n) {
            p *= 2;
            POLQAHandle->STFTBufferLengthLog2++;
        }

        ASSERT (p == n);

        POLQAHandle->STFTBufferLength = n;
    }
    x = POLQAHandle->STFTBuffer;

    ASSERT (2*aNumberOfBands == n);

    int StartPos = pFrameIndex * n/2 + Delay;
    int EndPos   = StartPos + n - 1;
    int Length;
    const int timeSeriesLen = statics->nrTimesSamples;

    if (StartPos < 0)
        matbZero(&x[0], (((-StartPos) < (n)) ? (-StartPos) : (n)));
    if (EndPos  >= timeSeriesLen)
        matbZero(&x[n-1-(((EndPos - timeSeriesLen) < (n-1)) ? (EndPos - timeSeriesLen)
: (n-1))], (((EndPos - timeSeriesLen +1) < (n)) ? (EndPos - timeSeriesLen +1) :
(n)));

    StartPos = (((StartPos) < (timeSeriesLen -1)) ? (StartPos) : (timeSeriesLen -1));
    EndPos   = (((EndPos) > (0)) ? (EndPos) : (0));
    Length   = (((EndPos) < (timeSeriesLen -1)) ? (EndPos) : (timeSeriesLen -1)) -
(((StartPos) > (-StartPos)) ? (StartPos) : (-StartPos)) + 1;
    if (-StartPos < n && Length <= n-(((-StartPos) > (0)) ? (-StartPos) : (0)))
        matbCopy(pTimeSeries.m_pData + (((StartPos) > (0)) ? (StartPos) : (0)), x +
(((-StartPos) > (0)) ? (-StartPos) : (0)), Length);
    matbMpy2(statics->frameWindow, x, n);

    matRealFft (POLQAHandle->mh, x, POLQAHandle->STFTBufferLengthLog2, MAT_Forw);

    this->m_pData[pFrameIndex][0] = 0;

    for (int bandIndex = 1; bandIndex < aNumberOfBands; bandIndex++)
    {
        a = x [2 * bandIndex];
        b = x [2 * bandIndex + 1];
        this->m_pData[pFrameIndex][bandIndex] = a * a + b * b;
    }

}

void CPsqmArray::STFTPowerAndPhaseSpectrumOf ( CPOLQAData *POLQAHandle,
                                        const CTimeSeries &pTimeSeries,
                                        const CIntArray   &pStartSampleUtterance,
                                        const CIntArray   &pStopSampleUtterance,
                                        const CIntArray   &DelayUtterance,
                                        bool IsRef, bool IgnoreDelay) {
    int frameIndex;
    int Windowsize = pTimeSeries. GetFrameLength ();

    if (IgnoreDelay || IsRef)
    {
        for (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
                STFTPowerAndPhaseSpectrumOf (POLQAHandle, pTimeSeries, frameIndex, 0);
        }
    }
    else
    {
        for (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
                STFTPowerAndPhaseSpectrumOf (POLQAHandle, pTimeSeries, frameIndex,
IsRef?0:DelayUtterance.m_pData[utt]);
        }
    }
}

void CPsqmArray::STFTPowerAndPhaseSpectrumOf (  CPOLQAData  *POLQAHandle,
                                                const       CTimeSeries &pTimeSeries,
                                                int         pFrameIndex,
                                                int         Delay)
{

    XFLOAT              a, b;

    int                 n = pTimeSeries. GetFrameLength ();
    XFLOAT              *x;
    CNewStdString          s;
    if (POLQAHandle->STFTBufferLength != n)
    {
        if (POLQAHandle->STFTBufferLength > 0)
            matFree(POLQAHandle->STFTBuffer);

        POLQAHandle->STFTBuffer = (XFLOAT*)matMalloc((n + 2)*sizeof(XFLOAT));

        int p = 1;
        POLQAHandle->STFTBufferLengthLog2 = 0;
        while (p < n) {
            p *= 2;
            POLQAHandle->STFTBufferLengthLog2++;
        }

        ASSERT (p == n);

        POLQAHandle->STFTBufferLength = n;
    }

    x = POLQAHandle->STFTBuffer;

    ASSERT (2*aNumberOfBands == n);

    int StartPos = pFrameIndex * n/2 + Delay;
    int EndPos   = StartPos + n - 1;
    int Length;
    const int timeSeriesLen = statics->nrTimesSamples;

    if (StartPos < 0)
        matbZero(&x[0], (((-StartPos) < (n)) ? (-StartPos) : (n)));
    if (EndPos >= timeSeriesLen)
        matbZero(&x[n-1-(((EndPos - timeSeriesLen) < (n-1)) ? (EndPos - timeSeriesLen)
: (n-1))], (((EndPos - timeSeriesLen +1) < (n)) ? (EndPos - timeSeriesLen +1) :
(n)));

    StartPos = (((StartPos) < (timeSeriesLen -1)) ? (StartPos) : (timeSeriesLen -1));
    EndPos   = (((EndPos) > (0)) ? (EndPos) : (0));
    Length   = (((EndPos) < (timeSeriesLen -1)) ? (EndPos) : (timeSeriesLen -1)) -
(((StartPos) > (-StartPos)) ? (StartPos) : (-StartPos)) + 1;
    if (-StartPos < n && Length <= n-(((-StartPos) > (0)) ? (-StartPos) : (0)))
        matbCopy(pTimeSeries.m_pData + (((StartPos) > (0)) ? (StartPos) : (0)), x +
(((-StartPos) > (0)) ? (-StartPos) : (0)), Length);
    matbMpy2(statics->frameWindow, x, n);

    matRealFft(POLQAHandle->mh, x, POLQAHandle->STFTBufferLengthLog2, MAT_Forw);

    this->m_pData[pFrameIndex][0] = 0;

    for (int bandIndex = 1; bandIndex < aNumberOfBands; bandIndex++) {
        a = x [2 * bandIndex];
        b = x [2 * bandIndex + 1];
        this->m_pData[pFrameIndex][bandIndex] = a * a + b * b;
    }

}

void CHzSpectrum::Initialize (CNewStdString pName, CPOLQAData* POLQAHandle)
{
    CSignal::Initialize(pName, POLQAHandle->statics->aNumberOfHzBands, POLQAHandle);
}

void CBarkSpectrum::Initialize(CNewStdString pName, CPOLQAData *POLQAHandle)
{
    CSignal::Initialize(pName, POLQAHandle->statics->aNumberOfBarkBands, POLQAHandle);

}

void CBarkSpectrum::ExcitationOf(CPOLQAData *POLQAHandle, const CBarkSpectrum
&pInputArray, bool* pUseThisFrame, int pListeningConditionChoice)
 {
    int        mu, nu;
    XFLOAT      t2, q, factor, bandQ, hulpLow=0.0, hulpTotal=0.0, hulp;

    XFLOAT    hulpCorrection, h;
    int       forLim1, forLim2, convLow, convHigh;
    int       frameIndex;
    CNewStdString  s;

    convLow  = 20;
    convHigh = 40;

    for (frameIndex = statics->startFrameIdx; frameIndex <= statics->stopFrameIdx;
frameIndex++)
    {
        hulpLow  = 0.0;
        hulpTotal = 0.0;

        for (nu = 3; nu < statics->aNumberOfBarkBands; nu++)
        {

            bandQ = (XFLOAT) pow ((XFLOAT)pInputArray.m_pData[frameIndex][nu], (XFLOAT)
2.0 / (XFLOAT) 2.0);

            factor = statics->aT1;
            q = bandQ;
            forLim1 = nu - convLow;
            if (forLim1 < 0) {forLim1 = 0;}

            for (mu = nu; ((mu >= forLim1) && (q != 00)); mu--)
            {
                this->m_pData[frameIndex][mu] += q;
                q *= factor;
            }

            t2 = statics->aT20[nu] * (XFLOAT) pow ((XFLOAT)bandQ, (XFLOAT)(0.21 *
statics->aWidthOfBandBark[1]));
            factor = t2;
            q = t2 * bandQ;

            forLim2 = (((aNumberOfBands - 1) < (nu + convHigh)) ? (aNumberOfBands - 1)
: (nu + convHigh));

            for (mu = nu + 1; ((mu <= forLim2) && (q != 00)); mu++) {
                this->m_pData[frameIndex][mu] += q;
                q *= factor;
            }
            if (statics->aCentreOfBandHz[nu]<100.0)
                hulpLow += pInputArray.m_pData[frameIndex][nu];
            if (statics->aCentreOfBandHz[nu]<3600.0)
                hulpTotal += pInputArray.m_pData[frameIndex][nu];
        }

            if (pListeningConditionChoice==3)
                hulpTotal = (2.0*pow((hulpTotal+3.0e4)/(hulpLow+3.0e4),0.2));
            else
                hulpTotal = (6.0*pow((hulpLow+3.0e4)/(hulpTotal+3.0e4),0.05));

        for (mu = 1; mu < aNumberOfBands; mu++) {

            h = this->m_pData[frameIndex][mu];
            this->m_pData[frameIndex][mu] = (XFLOAT) pow (h, (XFLOAT) 2.0/(XFLOAT)
2.0);

            hulp = (statics->aCentreOfBandBark[mu] + 1.0);

            if (pListeningConditionChoice==3) {
                if ( this->m_pData[frameIndex][mu] >(1.0e6/hulp))
                    this->m_pData[frameIndex][mu] = (1.0e6/hulp) - 1.0 +
pow((this->m_pData[frameIndex][mu] - (1.0e6/hulp) +1.0 ),0.99) ;
                if ( this->m_pData[frameIndex][mu] >(1.0e7/hulp))
                    this->m_pData[frameIndex][mu] = (1.0e7/hulp) - 1.0 +
pow((this->m_pData[frameIndex][mu] - (1.0e7/hulp) +1.0 ),0.98);
                if ( this->m_pData[frameIndex][mu] >(1.0e8/hulp))
                    this->m_pData[frameIndex][mu] = (1.0e8/hulp) - 1.0 +
pow((this->m_pData[frameIndex][mu] - (1.0e8/hulp) +1.0 ),0.95) ;
                if ( this->m_pData[frameIndex][mu] >(1.0e9/hulp))
                    this->m_pData[frameIndex][mu] = (1.0e9/hulp) - 1.0 +
pow((this->m_pData[frameIndex][mu] - (1.0e9/hulp) +1.0 ),0.88) ;
                if ( this->m_pData[frameIndex][mu] >(1.0e10/hulp))
                    this->m_pData[frameIndex][mu] = (1.0e10/hulp) - 1.0 +
pow((this->m_pData[frameIndex][mu] - (1.0e10/hulp) +1.0 ),0.80) ;
            } else {
                if ( this->m_pData[frameIndex][mu] >(1.0e6/hulp))
                    this->m_pData[frameIndex][mu] = (1.0e6/hulp) - 1.0 +
pow((this->m_pData[frameIndex][mu] - (1.0e6/hulp) +1.0 ),0.995) ;
                if ( this->m_pData[frameIndex][mu] >(1.0e7/hulp))
                    this->m_pData[frameIndex][mu] = (1.0e7/hulp) - 1.0 +
pow((this->m_pData[frameIndex][mu] - (1.0e7/hulp) +1.0 ),0.99);
                if ( this->m_pData[frameIndex][mu] >(1.0e8/hulp))
                    this->m_pData[frameIndex][mu] = (1.0e8/hulp) - 1.0 +
pow((this->m_pData[frameIndex][mu] - (1.0e8/hulp) +1.0 ),0.98) ;
                if ( this->m_pData[frameIndex][mu] >(1.0e9/hulp))
                    this->m_pData[frameIndex][mu] = (1.0e9/hulp) - 1.0 +
pow((this->m_pData[frameIndex][mu] - (1.0e9/hulp) +1.0 ),0.93) ;
                if ( this->m_pData[frameIndex][mu] >(1.0e10/hulp))
                    this->m_pData[frameIndex][mu] = (1.0e10/hulp) - 1.0 +
pow((this->m_pData[frameIndex][mu] - (1.0e10/hulp) +1.0 ),0.88) ;
            }

        }
    }

    for (frameIndex = statics->startFrameIdx + 1; frameIndex <= statics->stopFrameIdx;
frameIndex++) {
        for (nu = 1; nu < aNumberOfBands; nu++) {
            if (pListeningConditionChoice==3) {
                if ( (pUseThisFrame[frameIndex-1]) && (pUseThisFrame[frameIndex]) )
                    this->m_pData[frameIndex][nu] +=
(0.2/pow((aCentreOfBandBark.m_pData[nu]+1.0),1.0))*
this->m_pData[frameIndex-1][nu];
            } else {
                if ( (pUseThisFrame[frameIndex-1]) && (pUseThisFrame[frameIndex]) )
                    this->m_pData[frameIndex][nu] +=
(0.23/pow((aCentreOfBandBark.m_pData[nu]+1.0),3.0))*
this->m_pData[frameIndex-1][nu];
            }
        }
    }

    for (frameIndex = statics->startFrameIdx; frameIndex <= statics->stopFrameIdx;
frameIndex++)
    {
        for (nu = 0; nu < aNumberOfBands; nu++) {
            if (pListeningConditionChoice==3) {
                hulp = pow((aCentreOfBandBark.m_pData[nu]+0.5),0.2);
                hulpCorrection = aCentreOfBandBark.m_pData[nu]/hulpTotal;
                if (hulpCorrection>1.0) hulpCorrection = 1.0;
                hulpCorrection = pow(hulpCorrection,8.0);
                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 ) ;
                if (this->m_pData[frameIndex][nu] < 0.0) this->m_pData[frameIndex][nu]
= 0.0;
            } else {
                hulp = pow((aCentreOfBandBark.m_pData[nu]+0.5),0.3);
                hulpCorrection = aCentreOfBandBark.m_pData[nu]/hulpTotal;
                if (hulpCorrection>1.0) hulpCorrection = 1.0;
                hulpCorrection = pow(hulpCorrection,8.0);
                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 ) ;
                if (this->m_pData[frameIndex][nu] < 0.0) this->m_pData[frameIndex][nu]
= 0.0;
            }
        }
    }

}

void CBarkSpectrum::TimeLpAudibleOf (CPOLQAData           *POLQAHandle,
                                     const CBarkSpectrum  &pInputSpectrum,
                                     const CIntArray      &pActiveRatioOk,
                                     XFLOAT                pPower)
{
    for (int bandIndex = 0; bandIndex < aNumberOfBands; bandIndex++)
    {
        XFLOAT result = 0;
        int    count = 0;
        for (int frameIndex = statics->startFrameIdx; frameIndex <=
statics->stopFrameIdx; frameIndex++)
        {
            count++;
            if (pActiveRatioOk.m_pData[frameIndex])
            {
                if (pInputSpectrum.m_pData[frameIndex][bandIndex]  >
10.0*aAbsoluteThresholdPower.m_pData[bandIndex])
                {
                    result += pow((pInputSpectrum.m_pData[frameIndex][bandIndex]),
pPower);
                }
            }
        }

        this->m_pData[0][bandIndex] = (XFLOAT) (pow ( (result/(count+1.0)), 1 / pPower)
);
    }

}

void CBarkSpectrum::TimeLpAudibleOfSilent ( const CBarkSpectrum     &pInputSpectrum,
                                            const CIntArray          &pSilent,
                                            XFLOAT                   pPower,
                                            int
pNumberOfSilentFrames)
{
    for (int bandIndex = 0; bandIndex < aNumberOfBands; bandIndex++)
    {
        XFLOAT result = 0;
        XFLOAT hulp = 0.0;

        for (int frameIndex = statics->startFrameIdx; frameIndex <=
statics->stopFrameIdx; frameIndex++)
        {
            if ( (pSilent.m_pData[frameIndex]))
            {
                result += pow((pInputSpectrum.m_pData[frameIndex][bandIndex]), pPower);
            }
        }
        this->m_pData[0][bandIndex] = (XFLOAT) (pow (
(result/(pNumberOfSilentFrames+1.0)),1 / pPower) );
    }
}

void CBarkSpectrum::TimeLpOf (CPOLQAData            *POLQAHandle,
                              const CBarkSpectrum  &pInputSpectrum,
                              const CIntArray      &pActive,
                              XFLOAT                pPower)
{
    int length = statics->stopFrameIdx - statics->startFrameIdx + 1;

    SmartBufferPolqa SB1(POLQAHandle, length);
    XFLOAT *temp1 = SB1.Buffer;
    SmartBufferPolqa SB2(POLQAHandle, length);
    XFLOAT *temp2 = SB2.Buffer;

    int count;
    XFLOAT result;

    for (int bandIndex = 0; bandIndex < aNumberOfBands; bandIndex++)
    {
        result = 0;
        count = 0;
        for (int frameIndex = statics->startFrameIdx; frameIndex <=
statics->stopFrameIdx; frameIndex++)
        {
            if (pActive.m_pData[frameIndex])
            {
                temp1[count] = pInputSpectrum.m_pData[frameIndex][bandIndex];
                count++;
            }
        }
        matbPow2(temp1, pPower, temp2, count);

        result = matSum(temp2, count);

        this->m_pData[0][bandIndex] = (XFLOAT)(pow((result/(count+1.0)), 1/pPower));
    }

}

void CPairParameters::PrintFrequencyResponse(CNewLogFile            &pResultsFile,
                                            const CBarkSpectrum
&pOriginalPitchPowerDensitySum,
                                            const CBarkSpectrum
&pDistortedPitchPowerDensitySum)
{
    XFLOAT  x;
    CNewStdString s;

    pResultsFile. WriteString ("PERCEPTUAL FREQUENCY RESPONSE in dB\n");
    pResultsFile. WriteString ("Frequency (Bark)   Frequency (Hz)   Relative Level (dB)
  Band Index Number\n");
    for (int bandIndex = 0; bandIndex < statics->aNumberOfBarkBands; bandIndex++)
    {
        if ((pOriginalPitchPowerDensitySum.m_pData[0][bandIndex] < 0.1)  &&
(pDistortedPitchPowerDensitySum.m_pData[0][bandIndex] < 0.1) )
        {
            x = 98765.43;
        }
        else
        {
            x = 10*log ((XFLOAT) ((pDistortedPitchPowerDensitySum.m_pData[0][bandIndex]
+ 0.01)/(pOriginalPitchPowerDensitySum.m_pData[0][bandIndex] + 0.01))) /
log(10.0);
        }
        s. Format ("      %5.1f      %8.0f               %5.1f                %i \n",
statics->aCentreOfBandBark[bandIndex], statics->aCentreOfBandHz[bandIndex], x,
bandIndex);
        pResultsFile.WriteString (s);
    }
    pResultsFile.WriteString ("\n\n");
}

void CBarkSpectrum::AudibleFreqRespCompensationOf(CPOLQAData            *POLQAHandle,
                                                  const  CBarkSpectrum
&pInputSpectrum1,
                                                  const  CBarkSpectrum
&pInputSpectrum2,
                                                  XFLOAT                 pConstant,
                                                  XFLOAT                 pPower,
                                                   int
pListeningConditionChoice){
    int    frameIndex;
    int    bandIndex;
    XFLOAT ref_pow, deg_pow;

    ref_pow = 0.0;
    deg_pow = 0.0;
    for (bandIndex = 0; bandIndex < aNumberOfBands; bandIndex++) {
        ref_pow +=  pInputSpectrum1.m_pData[0][bandIndex];
        deg_pow +=  pInputSpectrum2.m_pData[0][bandIndex];
    }

    XFLOAT const ss = (ref_pow+0.1) / (deg_pow+0.1);

    XFLOAT x, y, hulp;
    for (bandIndex = 0; bandIndex < aNumberOfBands; bandIndex++) {
        if (bandIndex == 0)
            x = (ss*pInputSpectrum2.m_pData[0][0] + pConstant) /
(pInputSpectrum1.m_pData[0][0] + pConstant);
        if (bandIndex == 1)
            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);
        if (bandIndex == 2)
```

```
            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);
        if (bandIndex == aNumberOfBands-1)
            x = (ss*pInputSpectrum2.m_pData[0][aNumberOfBands-1] + pConstant) /
(pInputSpectrum1.m_pData[0][aNumberOfBands-1] + pConstant);
        if (bandIndex == aNumberOfBands-2)
            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);
        if ( (bandIndex > 10) && (bandIndex < (aNumberOfBands-4)) )
            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);
        if ((bandIndex > 2) && (bandIndex < (aNumberOfBands-2)) )
            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);

        for (frameIndex = statics->startFrameIdx; frameIndex <= statics->stopFrameIdx;
frameIndex++) {
            hulp = pPower;
            y = pow (x,hulp);
            this->m_pData[frameIndex][bandIndex] *= y;
        }
    }

}

void CBarkSpectrum::AudibleFreqRespCompensationExact (const CBarkSpectrum
&pInputSpectrum1,
                                                  const  CBarkSpectrum
&pInputSpectrum2,
                                                  XFLOAT                    pConstant)
{
    XFLOAT    x;

    for (int bandIndex = 0; bandIndex < aNumberOfBands; bandIndex++) {
        x = ( (pInputSpectrum2.m_pData[0][bandIndex] + pConstant) /
(pInputSpectrum1.m_pData[0][bandIndex] + pConstant) );
        for (int frameIndex = statics->startFrameIdx; frameIndex <=
statics->stopFrameIdx; frameIndex++)
        {
            this->m_pData[frameIndex][bandIndex] *= x;
        }

    }
}

void CBarkSpectrum::AudibleNoiseRespCompensationOf (const CBarkSpectrum
&pInputSpectrum) {
    int       frameIndex;

    int       bandIndex;

    for (bandIndex = 0; bandIndex < aNumberOfBands; bandIndex++) {
        for (frameIndex = statics->startFrameIdx; frameIndex <= statics->stopFrameIdx;
frameIndex++)
        {
            this->m_pData[frameIndex][bandIndex] -=
pInputSpectrum.m_pData[0][bandIndex];
            if ( this->m_pData[frameIndex][bandIndex] < 0.0)
                this->m_pData[frameIndex][bandIndex] = 0.0;
        }
    }
```

```
}

void CBarkSpectrum::AudibleNoiseRespCompensationOfPartly (CPOLQAData*
POLQAHandle,
                                                          const CBarkSpectrum
&pInputSpectrum,
                                                          XFLOAT
pFactor)
{

    int  bandIndex;

    for (bandIndex = 0; bandIndex < aNumberOfBands; bandIndex++) {
        for (frameIndex = statics->startFrameIdx; frameIndex <= statics->stopFrameIdx;
frameIndex++) {
            this->m_pData[frameIndex][bandIndex] -=
pFactor*pInputSpectrum.m_pData[0][bandIndex];
            if ( this->m_pData[frameIndex][bandIndex] < 0.0)
                this->m_pData[frameIndex][bandIndex] = 0.0;
        }
    }

}

void CBarkSpectrum::AudibleNoiseRespCompensationOfPartly2(  const CBarkSpectrum
&pInputSpectrum,
                                                            XFLOAT pFactor, XFLOAT
pConstant)
{
    XFLOAT      hulp1, hulp2, bandLow, bandHigh, hulp, hulpPow;

    bandLow = 2.0;
    bandHigh = 20.0;
    for(int bandIndex = 0; bandIndex < statics->aNumberOfBarkBands; bandIndex++)
    {
        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 = 3.0;
    bandHigh = 16.0;
    for (int bandIndex = 0; bandIndex < aNumberOfBands; bandIndex++)
    {
        hulp1 = 1.0;
        hulp2 = 1.0;
        if (statics->aCentreOfBandBark[bandIndex] < bandLow)
            hulp1 = (7.0 - 2.0*statics->aCentreOfBandBark[bandIndex]);
        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>1.2)
            hulpPow=1.2;

        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 (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);

        matbMpy1(statics->aCalibrationFactorSp, sum, aNumberOfBands);

        matbCopy(sum, this->m_pData[frameIndex], aNumberOfBands);
    }

}

void CBarkSpectrum::IntensityWarpingOf (CPOLQAData *POLQAHandle, const CBarkSpectrum 
&pInputSpectrum)
{
    XFLOAT    *hulp, *temp;
    XFLOAT    *modifiedZwickerPower;

    const int aNumberOfBarkBands = statics->aNumberOfBarkBands;

    SmartBufferPolqa SB1(POLQAHandle, aNumberOfBarkBands);
    temp = SB1.Buffer;
    SmartBufferPolqa SB2(POLQAHandle, aNumberOfBarkBands);
    hulp = SB2.Buffer;

    int h1Idx, h2Idx;
    for (h1Idx = aNumberOfBarkBands -1; h1Idx >= 0    && 
statics->aCentreOfBandBark[h1Idx] >= (XFLOAT) 2.0; h1Idx--);
    for (h2Idx = h1Idx+1; h2Idx < aNumberOfBarkBands && 
statics->aCentreOfBandBark[h2Idx] <= (XFLOAT)22.0; h2Idx++);

    for (int frameIndex = statics->startFrameIdx; frameIndex <= statics->stopFrameIdx; 
frameIndex++)
    {
        matbCopy((pInputSpectrum.m_pData[frameIndex]), temp, aNumberOfBarkBands);

        matbAdd1(600.0, temp, aNumberOfBarkBands);
        matbPow2(temp, 0.009, hulp, aNumberOfBarkBands);
        matbThresh1(hulp, aNumberOfBarkBands, 1.2, MAT_GT);

        matbSet(1.0, temp, aNumberOfBarkBands);

        for (bandIndex = 0; bandIndex < aNumberOfBarkBands; bandIndex++)
        {
            if(aCentreOfBandBark.m_pData[bandIndex] < (XFLOAT) 2.0)
                temp[bandIndex] =  -0.03*aCentreOfBandBark.m_pData[bandIndex] + 1.06;

            else
            {
                if(aCentreOfBandBark.m_pData[bandIndex] > (XFLOAT) 22.0)
                    temp[bandIndex] =  0.2*(aCentreOfBandBark.m_pData[bandIndex]-22.0)
+ 1.0;
            }
        }

        modifiedZwickerPower = temp;

        matbMpy2(hulp, modifiedZwickerPower, aNumberOfBarkBands);

        matbMpy1(0.22, modifiedZwickerPower, aNumberOfBarkBands);

        matbZero(this->m_pData[frameIndex], aNumberOfBarkBands);

        XFLOAT *pInputSpec = (pInputSpectrum.m_pData[frameIndex]);

        for (int bandIndex = 0; bandIndex < aNumberOfBarkBands; bandIndex++)
        {
            if (pInputSpec[bandIndex]  > statics->aAbsoluteThresholdPower[bandIndex])
            {
                this->m_pData[frameIndex][bandIndex] = (XFLOAT) (pow
(statics->aAbsoluteThresholdPower[bandIndex] / 0.5,
modifiedZwickerPower[bandIndex])
                        * (pow (0.5 + 0.5 * pInputSpec[bandIndex] /
statics->aAbsoluteThresholdPower[bandIndex],
modifiedZwickerPower[bandIndex]) - 1.0));
            }
        }
    }
    (*this) *= statics->aCalibrationFactorSl;

}

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::SpectralFlatnessPower (int pFrameIndex) const
{
    XFLOAT    integral = 0;
    XFLOAT integralLog = 0;
    int          count = 0;

    for (int bandIndex = 3; bandIndex < aNumberOfBands; bandIndex++) {
        XFLOAT h = this->m_pData[pFrameIndex][bandIndex];
        if (h > 1.0) {
            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::Integral (CPOLQAData *POLQAHandle, int pFrameIndex)
{
    SmartBufferPolqa SB(POLQAHandle, aNumberOfBands);
    XFLOAT *temp = SB.Buffer;

    const int length = aNumberOfBands - 1;

    matbMpy3(this->m_pData[pFrameIndex] + 1, statics->aWidthOfBandBark + 1, temp,
length);

    return matSum(temp, length);
}

XFLOAT CBarkSpectrum::IntegralLpOverBandRange (CPOLQAData *POLQAHandle, int
pFrameIndex, XFLOAT pPower, int bandIdxLow, int bandIdxHigh) {
    XFLOAT   result;

    int      bandIndex;
    result = (XFLOAT) 0;
    for (bandIndex = 0; bandIndex < aNumberOfBands; bandIndex++) {
        if ( (bandIndex >= bandIdxLow) && (bandIndex <= bandIdxHigh) )
            result += pow ( (XFLOAT)(this->m_pData[pFrameIndex][bandIndex] *
aWidthOfBandBark.m_pData[bandIndex]), pPower);
    }
    result = pow ((XFLOAT)result, (XFLOAT)(1.0/pPower));
    return (XFLOAT) result;

}

XFLOAT CBarkSpectrum::SumLpOverBandRange (int pFrameIndex, XFLOAT pPower, XFLOAT
pBandLow, XFLOAT pBandHigh )
{
    XFLOAT   result;
    result = (XFLOAT) 0;
    for (int bandIndex = 0; bandIndex < aNumberOfBands; bandIndex++)
    {
        if ( (statics->aCentreOfBandBark[bandIndex] > pBandLow) &&
(statics->aCentreOfBandBark[bandIndex] < pBandHigh) )
            result += pow ( (XFLOAT)this->m_pData[pFrameIndex][bandIndex], pPower);

    }
    result = pow ((XFLOAT)result, (XFLOAT)((XFLOAT)1.0/pPower));
    return result;
}

void CBarkSpectrum::MultiplyWithOverBandRange (int pFrameIndex, XFLOAT pFactor, XFLOAT
pBandLow, XFLOAT pBandHigh )
{
    int startBand = 0;
    int stopBand;
    int length;
    while(statics->aCentreOfBandBark[startBand] <= pBandLow)
        startBand++;

    stopBand = startBand;

    while(statics->aCentreOfBandBark[stopBand] <= pBandHigh && stopBand <
aNumberOfBands)
        stopBand++;

    length = stopBand - startBand;

    matbMpy1(pFactor, this->m_pData[pFrameIndex] + startBand, length);
}

XFLOAT CBarkSpectrum::IntegralLowNarrowband (CPOLQAData *POLQAHandle, int pFrameIndex)
{
    XFLOAT   hulp, result;

    result = (XFLOAT) 0.0;
    for (int bandIndex = 0; bandIndex < aNumberOfBands; bandIndex++)
    {
        if ( (statics->aCentreOfBandBark[bandIndex] < 12.0) &&
(statics->aCentreOfBandBark[bandIndex] > 2.0) )
        {
            hulp = this->m_pData[pFrameIndex][bandIndex] *
statics->aWidthOfBandBark[bandIndex];
            hulp *=  ((20.0 - statics->aCentreOfBandBark[bandIndex])/8.0);
            result += hulp;
        }
    }
    return result;
}

XFLOAT CBarkSpectrum::IntegralHighNarrowband (int pFrameIndex) {
    XFLOAT   hulp, result;

    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 (pListeningConditionChoice==3) {
            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;
            }
        } else {
            if (statics->aCentreOfBandBark[bandIndex] < 14.0) {
                hulp = this->m_pData[pFrameIndex][bandIndex] *
statics->aWidthOfBandBark[bandIndex];
                hulp *=  ((21.0 - statics->aCentreOfBandBark[bandIndex])/7.0);
                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 (pListeningConditionChoice==3) {
            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;
            }
        } else {
            if (statics->aCentreOfBandBark[bandIndex] > 7.0) {
                hulp = this->m_pData[pFrameIndex][bandIndex] *
statics->aWidthOfBandBark[bandIndex];
                hulp *=  ((statics->aCentreOfBandBark[bandIndex]-3.0)/5.0);
                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 (pListeningConditionChoice==3) {
            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;
            }
        } else {
            if (statics->aCentreOfBandBark[bandIndex] < 11.0) {
                hulp = this->m_pData[pFrameIndex][bandIndex] *
statics->aWidthOfBandBark[bandIndex];
                hulp *=  ((20.0 - statics->aCentreOfBandBark[bandIndex])/8.0);
                if (hulp>1.0) result += hulp;
            }
        }
    }
    return result;
}

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

    result = (XFLOAT) 0.0;
    for (int bandIndex = 0; bandIndex < aNumberOfBands; bandIndex++) {
        if (pListeningConditionChoice==3) {
            if (statics->aCentreOfBandBark[bandIndex] > 10.0) {
                hulp = this->m_pData[pFrameIndex][bandIndex] *
statics->aWidthOfBandBark[bandIndex];
                hulp *=  ((statics->aCentreOfBandBark[bandIndex]-2.0)/8.0);
                if (hulp>1.0) result += hulp;
            }
        } else {
            if (statics->aCentreOfBandBark[bandIndex] > 7.0) {
                hulp = this->m_pData[pFrameIndex][bandIndex] *
statics->aWidthOfBandBark[bandIndex];
                hulp *=  ((statics->aCentreOfBandBark[bandIndex]-2.0)/5.0);
                if (hulp>1.0) result += hulp;
            }
        }
    }
    return result;
}

XFLOAT CBarkSpectrum::TotalAudible (CPOLQAData* POLQAHandle, int pFrameIndex, XFLOAT
pFactor)
{
    XFLOAT  threshold, result;

    result = 0.;
    for (int bandIndex = 1; bandIndex < aNumberOfBands; bandIndex++)
    {
        threshold = pFactor * statics->aAbsoluteThresholdPower[bandIndex];
        if (this->m_pData[pFrameIndex][bandIndex] > threshold)
        {
            result += this->m_pData[pFrameIndex][bandIndex];
        }
    }
    return (XFLOAT) result;
}

XFLOAT CBarkSpectrum::Total (int pFrameIndex, XFLOAT pFrequencyLow, XFLOAT
pFrequencyHigh)
{

    int     bandIndex;
    XFLOAT  result;

    result = 0.;
    for (bandIndex = 1; bandIndex < aNumberOfBands; bandIndex++) {
        XFLOAT frequency = aCentreOfBandHz [bandIndex];
        if ((frequency >= pFrequencyLow) && (frequency <= pFrequencyHigh))
        {
            result += (*this) [pFrameIndex] [bandIndex];
        }
    }
    return (XFLOAT) result;

}

void CBarkSpectrum::Orthogonalize(const CIntArray &pSilent)
{
    for (int frameIndex = statics->startFrameIdx; frameIndex <= statics->stopFrameIdx;
frameIndex++)
    {
        if (pSilent.m_pData[frameIndex])
        {
            matbZero(this->m_pData[frameIndex] + 1, aNumberOfBands - 1);
        }
    }
}

XFLOAT CBarkSpectrum::FrameCorrelationTime (int pFrameIndex, int pNumberOfWindows)
const
{
    XFLOAT   sumXY, sumX, sumY, sumX2, sumY2, result;
    int      count;
    count = 0;
    sumXY = 0.0;
    sumX = 0.0;
    sumY = 0.0;
    sumX2 = 0.0;
    sumY2 = 0.0;

    XFLOAT X, Y, f;
    for (int bandIndex = 1; bandIndex < aNumberOfBands; bandIndex++) {
        f = statics->aCentreOfBandHz[bandIndex];

        if (f > 300) {
            X = this->m_pData[pFrameIndex-2][bandIndex];
            Y = this->m_pData[pFrameIndex][bandIndex];
            X = X - 0.8;
            Y = Y - 0.8;
            if (X < 0.0) X = 0.0;
            if (Y < 0.0) Y = 0.0;
            sumXY += X*Y;
            sumX += X;
            sumY += Y;
            sumX2 += X*X;
            sumY2 += Y*Y;
            count++;
        }

    }
    if ( count>2 && sumX>0.0 && sumY>0.0 )
        result = (count*sumXY-sumX*sumY)/
sqrt((count*sumX2-sumX*sumX)*(count*sumY2-sumY*sumY));
    else
        result = 0.0;

    return result;
}

void CBarkSpectrum::ComputeLpWeights (CPOLQAData *POLQAHandle, XFLOAT pBasePower,
XFLOAT pIncrementPower, int pNumberOfPowers, CDoubleArray pDisturbance [])
{

    SmartBufferPolqa SB1(POLQAHandle, pNumberOfPowers);
    XFLOAT *sum = SB1.Buffer;

    SmartBufferPolqa SB2(POLQAHandle, aNumberOfBands);
    XFLOAT *prod = SB2.Buffer;

    pIncrementPower *= 0.5;

    XFLOAT base, factor, totalWeight, z;

    totalWeight = matSum(aWidthOfBandBark.m_pData+1, aNumberOfBands-1);

    for (int frameIndex = statics->startFrameIdx; frameIndex <= statics->stopFrameIdx;
frameIndex++)
    {
        matbZero(sum, pNumberOfPowers);

        matbAbs2(this->m_pData[frameIndex] + 1, prod + 1, aNumberOfBands-1);

        matbMpy2(aWidthOfBandBark.m_pData + 1, prod + 1, aNumberOfBands-1);

        for (int bandIndex = 1; bandIndex < aNumberOfBands; bandIndex++)
        {
            base = pow (prod[bandIndex], pBasePower);
            factor = pow (prod[bandIndex], pIncrementPower);

            z = base;
            for (int i = 0; i < pNumberOfPowers; i++)
            {
                sum [i] += z;
                z *= factor;
            }
        }

        matbMpy1(1/totalWeight, sum, pNumberOfPowers);

        for (int i = 0; i < pNumberOfPowers; i++)
        {
            sum [i] = pow (sum [i], 1./(pBasePower + i * pIncrementPower));
        }
        matbMpy1(totalWeight, sum, pNumberOfPowers);

        for (int i = 0; i < pNumberOfPowers; i++)
        {
            pDisturbance[i].m_pData[frameIndex] = sum[i];
        }
    }
}

void CBarkSpectrum::MultiplyWithAsymmetryFactorAddOf (const CBarkSpectrum
&pOriginalPitchPowerDensity,
                                                      const CBarkSpectrum
&pDistortedPitchPowerDensity,
                                                      int
pListeningConditionChoice,
                                                      XFLOAT pNoiseContrastMax1, const
CIntArray      &pSilent , XFLOAT
pDistortedSilencePowerMeanCompens
ation)
{

    long  frameIndex;
    int   i;
    XFLOAT hulpAsymOrg, hulpAsymDis, ratioHulp,ratioHulpSilence, ratio, h,
bandHulpMax;;
    CNewStdString s;

    for (frameIndex = (statics->startFrameIdx+1); frameIndex <= statics->stopFrameIdx;
frameIndex++) {
        hulpAsymOrg = 0.0;
        hulpAsymDis = 0.0;
        for (i = 0; i < aNumberOfBarkBands; i++) {
            if (aCentreOfBandBark.m_pData[i] > 15.0) {
                bandHulpMax = pow((aCentreOfBandBark.m_pData[i]-14.0),0.5);
            } else {
                bandHulpMax = 1.0;
            }
            hulpAsymOrg +=
bandHulpMax*pOriginalPitchPowerDensity.m_pData[frameIndex][i];
            hulpAsymDis +=
bandHulpMax*pDistortedPitchPowerDensity.m_pData[frameIndex][i];
        }
        ratioHulp = pow( ((hulpAsymDis+6.0e6)/(hulpAsymOrg+6.0e6)),
5.0/pow(pNoiseContrastMax1,0.4));

        hulpAsymOrg = 0.0;
        hulpAsymDis = 0.0;
        for (i = 0; i < aNumberOfBarkBands; i++) {
            if (aCentreOfBandBark.m_pData[i] > 15.0) {
                bandHulpMax = pow((aCentreOfBandBark.m_pData[i] - 14.0), 0.5);
            } else {
                bandHulpMax = 1.0;
            }
            hulpAsymOrg +=
bandHulpMax*pOriginalPitchPowerDensity.m_pData[frameIndex][i];
            hulpAsymDis +=
bandHulpMax*pDistortedPitchPowerDensity.m_pData[frameIndex][i];
        }
        ratioHulpSilence = pow( ((hulpAsymDis+6.0e6)/(hulpAsymOrg+6.0e6)),
5.0/pow(pNoiseContrastMax1,0.4));

        for (i = 0; i < aNumberOfBarkBands; i++)
        {
            if (pListeningConditionChoice==3||pListeningConditionChoice==5)
            {
                if (aCentreOfBandBark.m_pData[i] > 18.0)
                    bandHulpMax = pow((aCentreOfBandBark.m_pData[i] - 17.0),0.9);
                else
                    bandHulpMax = 1.0;

                if (aCentreOfBandBark.m_pData[i] < 4.0)
                    bandHulpMax = pow((5.0 - aCentreOfBandBark.m_pData[i]),0.7);
            }
            else
            {
                if (aCentreOfBandBark.m_pData[i] > 18.0)
                    bandHulpMax = pow((aCentreOfBandBark.m_pData[i]-17.0),0.7);
                else
                    bandHulpMax = 1.0;

                if (aCentreOfBandBark.m_pData[i] < 4.0)
                    bandHulpMax = pow((5.0 - aCentreOfBandBark.m_pData[i]),0.7);

                bandHulpMax *= 1.2;
            }

            if (pSilent.m_pData[frameIndex])
            {
                ratio = (pDistortedPitchPowerDensity.m_pData[frameIndex][i]  + (XFLOAT)
200.0)
                          / (pOriginalPitchPowerDensity.m_pData[frameIndex][i] +
(XFLOAT) 200.0);
                h =  pow (ratio, 1.2)/(ratioHulpSilence) ;
                if (h < (XFLOAT) 1.0)
                    this->m_pData[frameIndex][i] *= pow(h,1.2);
                else
                {
                    if (pListeningConditionChoice==3||pListeningConditionChoice==5)
                    {
                        if (h > ( (XFLOAT)
6.5*bandHulpMax*pDistortedSilencePowerMeanCompensation) )
                            h = (XFLOAT)
6.5*bandHulpMax*pDistortedSilencePowerMeanCompensation;
                        this->m_pData[frameIndex][i] *= h;
                    }
                    else
                    {
                        if (h > ( (XFLOAT)
6.8*bandHulpMax*pDistortedSilencePowerMeanCompensation) )
                            h = (XFLOAT)
6.8*bandHulpMax*pDistortedSilencePowerMeanCompensation;
                        this->m_pData[frameIndex][i] *= h;
                    }
                }
            }
            else
            {
                ratio = (pDistortedPitchPowerDensity.m_pData[frameIndex][i]  + (XFLOAT)
2000.0)
                          / (pOriginalPitchPowerDensity.m_pData[frameIndex][i] +
(XFLOAT) 2000.0);
                h =  pow (ratio, 1.2)/(ratioHulp) ;
                if (h < (XFLOAT) 1.0)
                    this->m_pData[frameIndex][i] *= pow(h,1.2);
                else
                {
                    if (pListeningConditionChoice==3||pListeningConditionChoice==5)
                    {
                        if (h > ( (XFLOAT)
7.0*bandHulpMax*pDistortedSilencePowerMeanCompensation) )
                            h = (XFLOAT)
7.0*bandHulpMax*pDistortedSilencePowerMeanCompensation;
                        this->m_pData[frameIndex][i] *= h;
                    }
                    else
                    {
                        if (h > ( (XFLOAT)
7.0*bandHulpMax*pDistortedSilencePowerMeanCompensation) )
                            h = (XFLOAT)
7.0*bandHulpMax*pDistortedSilencePowerMeanCompensation;
                        this->m_pData[frameIndex][i] *= h;
                    }
                }
            }
        }
    }

}

void CBarkSpectrum::DifferenceOfBandlimited (const CPsqmArray &pInputCPsqmArray1, const
CPsqmArray &pInputCPsqmArray2)
```

```
{
    int frameIndex, nu;
    XFLOAT *pInputCPsqmA1, *pInputCPsqmA2;

    for (frameIndex = statics->startFrameIdx; frameIndex <= statics->stopFrameIdx;
frameIndex++)
    {
        pInputCPsqmA1 = (pInputCPsqmArray1.m_pData[frameIndex]);
        pInputCPsqmA2 = (pInputCPsqmArray2.m_pData[frameIndex]);

        matbSub3(pInputCPsqmA1, pInputCPsqmA2, this->m_pData[frameIndex],
aNumberOfBands);

        for(int band = 0; band < aDifferenceBarkScaling.GetSize() ; band++)
            this->m_pData[frameIndex][band] *= aDifferenceBarkScaling[band];

        for (nu = 0; nu < aNumberOfBands; nu++)
            if ( aCentreOfBandBark.m_pData[nu] > 19.0) {
                if ( aCentreOfBandBark.m_pData[nu] > 22.0) {
                    this->m_pData[frameIndex][nu] *= 0.0;
                } else {
                    this->m_pData[frameIndex][nu] /=
(aCentreOfBandBark.m_pData[nu]-18.0);
                }
            }

    }
}

}
```