```


    typedef double XFLOAT
    typedef double OTA_FLOAT


{

BOOL stringEndsWithWav (const CNewStdString &s) {
    int n

    n = s. GetLength ()
    if (n < 3) {
        return FALSE
    }

    if ((s [n-1] == 'V') || (s [n-1] == 'v')) {
        if ((s [n-2] == 'A')|| (s [n-2] == 'a')) {
            if ((s [n-3] == 'W') || (s [n-3] == 'w')) {
               return TRUE
            }
        }
    }
    return FALSE
}

CTimeSeries::CTimeSeries()
{
    aInitialized = FALSE

    aName = ""

    aHeaderDelayInBytes = 0
    aTrailerDelayInBytes = 0
    aStereoInFile = FALSE
}

BOOL CTimeSeries::Initialize(CNewStdString pName, CPOLQAData *polqaHandle)
{
    aName = pName

    if (aInitialized) {
        if (!gBatchMode) {
        }
        return FALSE
    }

    this->POLQAHandle = polqaHandle

    statics = polqaHandle->statics

    ASSERT(polqaHandle->statics->nrTimesSamples > 0)

    SetSize(polqaHandle->statics->nrTimesSamples)
    aInitialized = TRUE

    SetToConstant(0.)

    return true
}

void CTimeSeries::SetToConstant (XFLOAT pValue)
{
    matbSet(pValue, this->m_pData, statics->nrTimesSamples)
}

void CDoubleArray::RatioOf (const CDoubleArray &pNominator, const CDoubleArray &pDenominator, XFLOAT
pFuzz)
```

```
{
    int i, range
    int n = GetSize ()

    for (i = 0  i < n  i++) {
        int j
        XFLOAT totalWeight = 0
        range = 30
        for (j = -range  j <= range  j++) {
            if ((i + j >= 0) && (i + j < n)) {
                XFLOAT weight = 1.0 / (1.0 + 0.2*(XFLOAT)abs(j))
                this->m_pData[i+j] += weight * (pNominator.m_pData[i+j] + pFuzz) /
(pDenominator.m_pData[i+j] + pFuzz)
                totalWeight += weight
            }
        }
        this->m_pData[i] /= totalWeight
    }
}

void CDoubleArray::CompressOf (const CDoubleArray &pThat, XFLOAT pConstant)
{
    int i
    int n = GetSize ()

    for (i = 0  i < n  i++) {
        this->m_pData[i] = pow (pThat.m_pData[i], pConstant)
    }
}

XFLOAT CDoubleArray::Power (int pStartIndex, int pStopIndex) const
{
    int    i
    XFLOAT  power
    CNewStdString s

    power = 0

    if (pStartIndex < 0) {
        if (!gBatchMode) {
        } else {
            s. Format ("TimeSeries. Power : start index negative! " + aName)
            gLogFile. WriteString (s)
        }
        exit (1)
    }

    if (pStartIndex > pStopIndex) {
        if (!gBatchMode) {
        } else {
            s. Format ("TimeSeries. Power : stop index exceeds start index!\n" + aName)
            gLogFile. WriteString (s)
        }
        exit (1)
    }

    int n = GetSize ()

    if (pStopIndex > n) {
        if (!gBatchMode) {
        } else {
            s. Format ("TimeSeries. Power : stop index exceeds length!\n" + aName)
            gLogFile. WriteString (s)
        }

        exit (1)
    }

```

```
135     for (i = pStartIndex  i < pStopIndex  i++) {
136         XFLOAT h = this->m_pData[i]
137         power += h * h
138     }
139 
140     power /= (pStopIndex - pStartIndex)
141     return power
142 }
143 
144 XFLOAT CDoubleArray::PowerInBand(CPOLQAData *POLQAHandle, XFLOAT pLowerFrequency, XFLOAT
    pUpperFrequency) const
145 {
146     XFLOAT result = 0
147 
148     const XFLOAT frequencyResolutionHz = POLQAHandle->statics->aFrequencyResolutionHz
149     for(int bandIndex = 0  bandIndex < GetSize()  bandIndex++)
150     {
151         const XFLOAT frequency = bandIndex * frequencyResolutionHz
152         if((frequency >= pLowerFrequency) && (frequency <= pUpperFrequency))
153         {
154             result += this->m_pData[bandIndex]
155         }
156     }
157 
158     return result
159 }
160 
161 void CDoubleArray::InvDb2 (CPOLQAData *POLQAHandle,
162                            XFLOAT    pBasisDb [][2],
163                            int       pNumberOfPointsBasis,
164                            XFLOAT    pDeltaDb [][2],
165                            int       pNumberOfPointsDelta)
166 {
167     XFLOAT  centreOfBandHz
168     XFLOAT  gainDb
169     XFLOAT  gain
170 
171     const XFLOAT freqRes = POLQAHandle->statics->aFrequencyResolutionHz
172     const int size = GetSize()
173     for(int bandIndex = 0  bandIndex < size  bandIndex++)
174     {
175         centreOfBandHz = freqRes * (XFLOAT) bandIndex
176         gainDb = interpolate (centreOfBandHz, pBasisDb, pNumberOfPointsBasis)
177         gainDb += interpolate (centreOfBandHz, pDeltaDb, pNumberOfPointsDelta)
178         gain = pow (10.0, gainDb / 10.0)
179 
180         this->m_pData[bandIndex] = gain
181     }
182 }
183 
184 void CDoubleArray::TimeAvgOf(const CPOLQAData *POLQAHandle, const CHzSpectrum &pThat)
185 {
186     XFLOAT result
187     int    count
188 
189     const int stopFrameIdx = POLQAHandle->statics->stopFrameIdx
190     const int aNumberOfHzBands = POLQAHandle->statics->aNumberOfHzBands
191     for(int bandIndex = 0  bandIndex < aNumberOfHzBands  bandIndex++)
192     {
193         result = 0
194         count = 0
195         for(int frameIndex = POLQAHandle->statics->startFrameIdx  frameIndex <= stopFrameIdx
    frameIndex++)
196         {
197             result += (pThat.m_pData[frameIndex])[bandIndex]
198             count++
199         }
200         result /= (XFLOAT) count
```

```
201
202         this->m_pData[bandIndex] = result
203     }
204 }
205
206 const char *CTimeSeries::GetName (void) const
207 {
208     return (const char *) aName
209 }
210
211 void CTimeSeries::operator *= (XFLOAT pFactor)
212 {
213     matbMpy1(pFactor, this->m_pData, statics->nrTimesSamples)
214 }
215
216 void CTimeSeries::operator= (const CTimeSeries &pInputTimeSeries)
217 {
218     matbCopy(pInputTimeSeries.m_pData, this->m_pData, statics->nrTimesSamples)
219 }
220
221 XFLOAT CTimeSeries::Envelope (CPOLQAData *POLQAHandle, int pStartIndex, int frameLength) const
222 {
223     XFLOAT  envelope
224
225     SmartBufferPolqa SB(POLQAHandle, frameLength)
226     XFLOAT *temp = SB.Buffer
227
228     int length
229     if(pStartIndex + frameLength - 1 < statics->nrTimesSamples)
230         length = frameLength
231     else
232         length = statics->nrTimesSamples - pStartIndex
233
234     matbSqr2(this->m_pData+pStartIndex, temp, length)
235     envelope  = matSum(temp, length)
236
237     envelope /= frameLength
238     envelope  = sqrt(envelope)
239
240     return envelope
241 }
242
243 XFLOAT CTimeSeries::WindowedSample(int pFrameIndex, int pSampleIndex, int pWindowSize)
244 {
245     XFLOAT   result
246     int      i
247     const XFLOAT OverlapCoeff = 0.75
248     ASSERT ((0 <= pSampleIndex) && (pSampleIndex < pWindowSize))
249
250     i = pFrameIndex * pWindowSize * (1-OverlapCoeff) + pSampleIndex
251
252     if (i < 0) {
253         return 0
254     }
255
256     if (i >= this->GetSize()) {
257         return 0
258     }
259
260     result = statics->frameWindow[pSampleIndex] * this->m_pData[i]
261
262     return result
263 }
264
265 BOOL CTimeSeries::ReadFromBuffer (XFLOAT* pSamples, long NumberOfSamples)
266 {
267     matbCopy(pSamples, m_pData, NumberOfSamples)
268     aNumberOfSamples = NumberOfSamples
```

```
    return TRUE
}

void upperCase (char *outputString, const char *inputString) {
    int i, n

    n = strlen (inputString)
    for (i = 0  i < n  ++i) {
        outputString [i] = (char) toupper (inputString[i])
    }
}

BOOL CTimeSeries::OpenFile (XFLOAT aSampleFrequencyHz,
                            CNewStdString  pSoundFilePathName,
                            int  &pNumberOfSamples,
                            int    pStereoIfNotWavFile,
                            int    pRightIfStereo)
{
    unsigned int      flen, lengthInBytes, fileLength, sampleFrequency, bytesPerSecond
    int               n
    short             numChannels, bitsPerSample
    short             tag
    char              riffId[5], formatId [5], dataId[5]
    CNewFile             soundFile
    CNewStdString           s

    aHeaderDelayInBytes = 0
    aTrailerDelayInBytes = 0
    aRightIfStereo = pRightIfStereo

    if (!soundFile. Open (pSoundFilePathName, "rb")) {
        if (gBatchMode) {

            exit (1)
        } else {
        }
        return FALSE
    }

    if (!stringEndsWithWav (pSoundFilePathName)) {
        aStereoInFile = pStereoIfNotWavFile

        soundFile. SeekToEnd ()

        if (pStereoIfNotWavFile) {
            aNumberOfSamples = soundFile. GetLength () / 4
        } else {
            aNumberOfSamples = soundFile. GetLength () / 2
        }
    } else {
        soundFile. Read (riffId, 4)
        riffId [4] = '\0'
        upperCase (riffId, riffId)

        if (0 != strcmp (riffId, "RIFF")) {
            if (!gBatchMode) {
            }

            return FALSE
        }

        soundFile. Read (&fileLength, 4)
        soundFile. Read (riffId, 4)
        riffId[4] = '\0'
        upperCase (riffId, riffId)

        if (0 != strcmp (riffId, "WAVE")) {
            if (!gBatchMode) {
```

```
            }

            return FALSE
        }

        soundFile. Read (formatId, 4)
        formatId [4] = '\0'

        soundFile. Read (&flen, 4)
        soundFile. Read (&tag, 2)

        if (tag != 1) {
            if (!gBatchMode) {
            }
            return FALSE
        }

        soundFile. Read (&numChannels, 2)
        switch (numChannels) {
        case 1:
            aStereoInFile = FALSE
            break
        case 2:
            aStereoInFile = TRUE
            break
        default:
            if (!gBatchMode) {
            }
            return FALSE
        }

        soundFile. Read (&sampleFrequency, 4)

        if (aSampleFrequencyHz != (XFLOAT)sampleFrequency)
        {

            return FALSE
        }

        soundFile. Read (&bytesPerSecond, 4)

        soundFile. Read (&tag, 2)

        soundFile. Read (&bitsPerSample, 2)
        if (bitsPerSample != 16) {

            exit (1)
        }

        soundFile.Seek(flen - 16, SEEK_CUR)

        bool Found = false
        while (!Found)
        {
            if (4 == soundFile.Read(dataId, 4))
            {
                dataId[4] = '\0'
                upperCase(dataId, dataId)
                if (0 != strcmp(dataId, "DATA"))
                {
                    unsigned int cksize
                    if (4 == soundFile.Read(&cksize, 4))
                    {
                        soundFile.Seek(cksize, SEEK_CUR)
                    }
                    else break
                }
                else Found = true
```

```
405             }
406             else break
407         }
408
409         if (!Found)
410         {
411             s.Format("WAV-file %s: WAVE header invalid (%s!=DATA)!\n", pSoundFilePathName.c_str(),
    dataId)
412             if (!gBatchMode)
413             {
414             }
415             exit(1)
416         }
417
418         soundFile. Read (&lengthInBytes, 4)
419
420         aHeaderDelayInBytes = soundFile. GetPosition ()
421
422         n = soundFile. GetLength ()
423         aTrailerDelayInBytes = n - aHeaderDelayInBytes - lengthInBytes
424
425         aNumberOfSamples = lengthInBytes / 2
426
427         if (aStereoInFile)
428             aNumberOfSamples /= 2
429
430     }
431
432     if (!aStereoInFile) {
433         aRightIfStereo = FALSE
434     }
435
436     pNumberOfSamples = aNumberOfSamples
437
438     soundFile. Close ()
439
440     return TRUE
441 }
442
443 short SwapBytes (short a) {
444     short b = (short) ((a & 0xff) << 8)
445     short c = (short) ((a & 0xff00) >> 8)
446     return (short) (b | c)
447 }
448
449 BOOL CTimeSeries::ReadFromDisk (const CNewStdString &pSoundFilePathName,
450                                 long                pNumberOfSamples,
451                                 int                 pSwapBytes,
452                                 XFLOAT*             pChecksum)
453 {
454     short               h
455     int                 i
456     short               *buffer
457     CNewFile            soundFile
458
459     bool couldOpenSoundFile = false
460     int openTrials = 0
461     const int maxOpenTrials = 20
462
463     couldOpenSoundFile = soundFile. Open (pSoundFilePathName, "rb")
464     while(!couldOpenSoundFile && openTrials < maxOpenTrials)
465     {
466         couldOpenSoundFile = soundFile. Open (pSoundFilePathName, "rb")
467         openTrials++
468         Sleep(200)
469     }
470
471     if (!couldOpenSoundFile) {
```

```
472         if (gBatchMode)
473         {
474
475             exit (1)
476         }
477         else
478         {
479         }
480         return FALSE
481     }
482
483     buffer = new short [2 * pNumberOfSamples]
484
485     ASSERT (sizeof(short) == 2)
486
487     if (aStereoInFile)
488     {
489         soundFile.Seek(aHeaderDelayInBytes, SEEK_SET)
490
491         if (sizeof (short) * 2 * pNumberOfSamples != soundFile. Read (buffer, sizeof (short) * 2 *
    pNumberOfSamples))
492         {
493             if (!gBatchMode) {
494             }
495
496             exit (1)
497         }
498
499     }
500     else
501     {
502         soundFile.Seek(aHeaderDelayInBytes, SEEK_SET)
503
504         if (sizeof (short) * pNumberOfSamples != soundFile.Read(buffer, sizeof (short) *
    pNumberOfSamples)) {
505             if (!gBatchMode)
506             {
507             }
508
509             exit (1)
510         }
511
512     }
513
514     for (i = 0  i < pNumberOfSamples  i++) {
515
516         if (aStereoInFile) {
517             if (aRightIfStereo) {
518                 h = buffer [2*i+1]
519             } else {
520                 h = buffer [2*i]
521             }
522         } else {
523             h = buffer [i]
524         }
525
526         if (pSwapBytes) {
527             this->m_pData[i] = (XFLOAT) SwapBytes (h)
528         } else {
529             this->m_pData[i] = (XFLOAT) h
530         }
531     }
532
533     delete [] buffer
534     soundFile.Close()
535
536     if (pChecksum)
537     {
```

```
538         XFLOAT sum = 0  for (int i=0  i< pNumberOfSamples  i++) sum+=m_pData[i]
539         *pChecksum = sum
540     }
541
542     return TRUE
543 }
544
545 BOOL MakeStereoFile (FILE* pOutputFile,
546                      const CTimeSeries  &pOriginalTimeSeries,
547                      const CTimeSeries  &pDistortedTimeSeries,
548                      CPOLQAData *POLQAHandle)
549 {
550     int            i
551     int            h
552     short          *buffer
553     CNewLogFile    outputFile(pOutputFile)
554     int            n
555
556     n = POLQAHandle->statics->nrTimesSamples
557
558     ASSERT (sizeof(short) == 2)
559
560     buffer = new short [2*n]
561
562     for (i = 0  i < n  i++) {
563         h = (int) round (pOriginalTimeSeries.m_pData[i]/2.0)
564         if (h < -32767) h = -32767
565         if (h > 32767)  h = 32767
566         h = (short) round (h)
567         buffer [2*i] = (short) h
568         h = (int) round (pDistortedTimeSeries.m_pData[i]/2.0)
569         if (h < -32767) h = -32767
570         if (h > 32767)  h = 32767
571         h = (short) round (h)
572         buffer [2*i + 1] = (short) h
573     }
574
575         outputFile.Write (buffer, sizeof (short) * 2 * n)
576
577     outputFile. Close ()
578     delete [] buffer
579
580     return TRUE
581 }
582
583 void CTimeSeries::SetToSine(XFLOAT pAmplitude, const XFLOAT pOmega)
584 {
585     for(int i = 0  i < statics->nrTimesSamples  i++)
586     {
587        this->m_pData[i] = sin(pOmega * i)
588     }
589     matbMpy1(pAmplitude, this->m_pData, statics->nrTimesSamples)
590 }
591
592 void CTimeSeries::FilterWith (CPOLQAData            *POLQAHandle,
593                               const CTimeSeries     &pInputTimeSeries,
594                               XFLOAT*               pTaps,
595                               int                   TapsLength)
596 {
597     int rc
598     rc = matRunFIRFilter(POLQAHandle->mh, pInputTimeSeries.m_pData, this->m_pData,
    statics->nrTimesSamples, pTaps, TapsLength, MAT_FIRDelayComp)
599     ASSERT(rc == 0)
600 }
601
602 void CTimeSeries::FilterWith (CPOLQAData         *POLQAHandle,
603                               BOOL               pInputFFTAvailable,
604                               XFLOAT             pSampleFrequencyHz,
```

```
                                XFLOAT            pFilterCurve [][2],
                                int               pNumberOfPoints,
                                const CTimeSeries &pInputTimeSeries,
                                CDoubleArray      &pInputFFT,
                                CDoubleArray      &pOutputFFT)
{
    XFLOAT                factorDb, factor
    XFLOAT                overallGainFilter = interpolate ((XFLOAT) 1000, pFilterCurve,
pNumberOfPoints)
    long                  i, powerOf2 = 1, order = 0
    XFLOAT                *x
    XFLOAT                frequencyResolution

    const int aTimeSeriesLength = statics->nrTimesSamples

    while (powerOf2 < aTimeSeriesLength)
    {
        powerOf2 *= 2
        order++
    }

    SmartBufferPolqa SB_x(POLQAHandle, powerOf2 + 2)
    x = SB_x.Buffer

    if (!pInputFFTAvailable)
    {
        matbZero(x, powerOf2 + 2)

        matbCopy(pInputTimeSeries.m_pData, x, aTimeSeriesLength)

        for (i = 0  i < aTimeSeriesLength  i++)
        {
            if (i < 100) {
                x [i] *= (XFLOAT) i / (XFLOAT) 100
            }
            if (aTimeSeriesLength - 1 - i < 100) {
                x [i] *= (XFLOAT) (aTimeSeriesLength - 1 - i) / (XFLOAT) 100
            }
        }

        matRealFft (POLQAHandle->mh, x, order, MAT_Forw)

        pInputFFT.Initialize("pInputFFT", powerOf2 + 2)
        matbCopy(x, pInputFFT.m_pData, powerOf2 + 2)
    }
    else
    {
        matbCopy(pInputFFT.m_pData, x, powerOf2 + 2)
    }

    frequencyResolution = pSampleFrequencyHz / powerOf2

    for (i = 0  i <= powerOf2/2  i++) {
        factorDb = interpolate (i * frequencyResolution, pFilterCurve, pNumberOfPoints) -
overallGainFilter
        factor = pow (10.0, factorDb / 20.0)

        x [2 * i] *= factor
        x [2 * i + 1] *= factor
    }

    pOutputFFT.Initialize("pOutputFFT", powerOf2 + 2)
    matbCopy(x, pOutputFFT.m_pData, powerOf2 + 2)

    matCcsFft (POLQAHandle->mh, x, order, MAT_Inv)

    matbCopy(x, this->m_pData, aTimeSeriesLength)
}
```

```
671
672 int CTimeSeries::GetLength() const
673 {
674     return statics->nrTimesSamples - SkipAtStart
675 }
676
677 int CTimeSeries::GetFrameLength() const
678 {
679     return statics->frameLength
680 }
681
682 }
683
```