CODAR OCEAN SENSORS

Cross Spectra File Format

113.01.1609.FF
Sep 23, 2016

Cross Spectra File Format Version 6

CrossSpectra files are produced by a SeaSonde Radial Site. They contain a snapshot in time of the ocean state in a cross spectra format, which is computed nominally from three antenna measurements. This data represents the reflected energy(self spectra) at each detectable range distance and doppler velocity as well as the cross spectra products of the antennas relative to each other. The cross spectra files are then used to calculate radial velocity vectors and ocean wave states.

The application SeaSondeAcquisition creates raw cross spectra in the "/Codar/SeaSonde/Data/Spectra/SpectraSeries/" folder. SeaSondeAcquisition saves the raw cross spectra file name as “CSQ_XXXX_YY_MM_DD_HHMMSS.cs” where XXXX is the site name; YY is the year, MM is the Month, HHMMSS is the 24hour/minute/second time.

The application SpectraAverager reads the CSQ files and produces 'CSS_XXXX_YY_MM_DD_HHMM.cs' files in the "/Codar/SeaSonde/Data/Spectra/SpectraToProcess/" folder where XXXX is the site name; YY is the year, MM is the Month, HHMM is the 24hour and minute time. 'CSS' stands for CrossSpectra short time, which on a standard SeaSonde covers 15 minutes with an output time every 10minutes.

The files are in a binary format.
They have a variable size header section followed by the self spectra and cross spectra products.
The data uses Big-Endian byte ordering (Most Significant Byte first). This means that on Intel platforms, you will need to swap the byte order for the variable being read.
IEEE floating point values single (4bytes) and double (8byte precision).
Two’s complement, integer values.

Data Type Definitions:

These define the binary fields used for the Header and Data structures below:
UInt8 Unsigned 8bit integer

SInt8	Signed 8bit integer
UInt16	Unsigned 16bit integer
SInt16	Signed 16bit integer
UInt32	Unsigned 32bit integer
SInt32	Signed 32bit integer
UInt64	Unsigned 64bit integer
SInt64	Signed 64bit integer
Float	IEEE single precision floating point number (4 bytes)
Double	IEEE double precision floating point number (8 bytes)
Complex	Two IEEE single precision floating point numbers of real and imaginary pairs (8 bytes, 4 bytes each float)
String	Variable size zero terminated ASCII

Cross Spectra File Contents:

Each file has two major sections, a **Header** section and a **Data** section. The **Header** section has grown with each new version. To read the **Header**, first read the nCsFileVersion and nV1Extent fields. Check these values and then read the rest of the expected header data up to version 5. The version 6 **Header** portion needs to be processed in blocks, details follow below. The **Data** section follows the Header and comes in two different possible contents.

Header Section:

The header size is variable. Each new version contains the information used by the previous versions.

When reading a CrossSpectra file that is a newer version than you expect then use the last known Extent field to skip to the beginning of the data section

The following Header descriptions are a set of data fields to be read in order from the start of the file, where each field description is a value type with implied size, followed by the field name, and followed by the field's description.

All versions start with the following fields as the very first bytes.
You should read this section first, validate it, and then read the rest of the header up to version 5 header if applicable.

SInt16	**nCsFileVersion**	File Version 1 to latest. (If greater than 32, it's probably not a spectra file.)
UInt32	**nDateTime**	TimeStamp. Seconds from Jan 1,1904 local computer time at site. The timestamp for CSQ files represents the start

		time of the data (nCsKind = 1) The timestamp for CSS and CSA files is the center time of the data (nCsKind = 2).
SInt32	**nV1Extent**	Header Bytes extension (Version 4 is +62 Bytes Till Data)

The following is added info for **version 2** to latest.

SInt16	**nCsKind**	Type of CrossSpectra Data. 1 is self spectra for all used channels, followed by cross spectra. Timestamp is start time of data. 2 is self spectra for all used channels, followed by cross spectra, followed by quality data. Timestamp is center time of data.
SInt32	**nV2Extent**	Header Bytes extension (Version 4 is +56 Bytes Till Data)

The following is added info for **version 3** to latest.

Char4	**nSiteCodeName**	Four character site code 'site'
SInt32	**nV3Extent**	Header Bytes extension (Version 4 is +48 Bytes Till Data)

Note. If version is 3 or less, then assume nRangeCells=31, nDopplerCells=512, nFirstRangeCell=1 (version 3 spectra is very not common.)

The following is added info for **version 4** to latest.

SInt32	**nCoverMinutes**	Coverage Time in minutes for the data. ‘CSQ' is normally 5minutes (4.5 rounded) 'CSS' is normally 15minutes average. 'CSA' is normally 60minutes average.
SInt32	**bDeletedSource**	Was the ‘CSQ' deleted by CSPro after reading.
SInt32	**bOverrideSrcInfo**	If not zero, CSPro used its own preferences to override the source ‘CSQ’ spectra sweep settings.
Float	**fStartFreqMHz**	Transmit Start Freq in MHz
Float	**fRepFreqHz**	Transmit Sweep Rate in Hz
Float	**fBandwidthKHz**	Transmit Sweep bandwidth in kHz
SInt32	**bSweepUp**	Transmit Sweep Freq direction is up if non zero, else down NOTE: CenterFreq is fStartFreqMHz + fBandwidthKHz/2 * -2^(bSweepUp==0)
SInt32	**nDopplerCells**	Number of Doppler Cells (nominally 512)
SInt32	**nRangeCells**	Number of RangeCells (nominally 32 for ‘CSQ', 31

		for 'CSS')
SInt32	**nFirstRangeCell**	Index of First Range Cell in data from zero at the receiver. ‘CSQ' files nominally use zero. 'CSS' files nominally use one because SpectraAverager cuts off the first range cell as meaningless. This value can sometimes be less than zero when bistatic timing causes an artificial negative range start. This value can sometimes be greater than zero when trying to reduce cross spectra to ranges of interest.
Float	**fRangeCellDistKm**	Distance between range cells in kilometers. The distance of a range cell is its index(from1) – 1 + nFirstRangeCell times fRangeCellDistKm.
SInt32	**nV4Extent**	Header Bytes extension (Version 4 is +0 Bytes Till Data) If zero then cross spectra data follows, but if this file were version 5 or greater then the nV4Extent would tell you how many more bytes the version 5 and greater uses until the data.

The following is added info for version 5 to latest.

SInt32	**nOutputInterval**	The Output Interval in Minutes.
Char4	**nCreateTypeCode**	The creator application type code.
Char4	**nCreatorVersion**	The creator application version.
SInt32	**nActiveChannels**	Number of active antennas
SInt32	**nSpectraChannels**	Number antenna used in cross spectra.
UInt32	**nActiveChanBits**	Bit indicator of which antennas are in use msb is ant#1 to lsb #32
SInt32	**nV5Extent**	Header Bytes extension (Version 5 is +0 Bytes Till Data) If zero then cross spectra data follows, but if this file is version 6 or greater then the nV5Extent would tell you how many more bytes the version 6 and greater uses until the data.

The following is added info for version 6 to latest

Version 6 differs than previous versions in that it adds a variable size section that is composed of optional blocks/chunks of data. Each block describes a different piece of meta data. Each of these blocks are typically optional and only exist in the spectra files when applicable. New blocks can be added at any time.

Reading tools should skip over blocks they don't recognize.

Before reading the blocks, first read nCS6ByteSize which comes just after the version 5 header structure

UInt32	**nCS6ByteSize**	Number of bytes of all blocks in the version 6 section. The blocks follow this field.

Then process each block. Be sure to validate the size of each block and their combined sizes (8+ **nBlockDataSize**) should equal **nCS6ByteSize**.

Each block is composed of:

Char4	**nBlockKey**	A four character block identifier.
UInt32	**nBlockDataSize**	Byte size of data to follow. Can be zero.

Followed by **nBlockDataSize** bytes of data. The block's data is specific to the **nBlockKey**. See below for description of block keys and their data format.

These pseudo instructions are for reading the blocks.

While **nCS6ByteSize** > 0
- Read **nBlockKey**
- Read **nBlockDataSize**
- Read **nBlockDataSize** bytes worth of data.
- If you do not recognize nBlockKey, then skip over this data. You should ensure that **nBlockDataSize** is at least as large as the data you're expecting. If it's larger than expected, then skip over the extra data. Once defined, block data cannot ever be changed, but it is possible to append new data to a block.
- Subtract 8 and **nBlockDataSize** from **nCS6ByteSize** to determine when to stop

End While

End of Header Section

Be sure to use the last known extent to skip past possible future data to the start of the Data Section.

If you've processed version 6 block sequentially, make sure you account for the amount read.

Begin Data Section:

The data section is a multi-dimensional array of self and cross spectra data. Below **[nDopplerCells]** stands for an array of values **nDopplerCells** long.

Repeat For 1 to nRangeCells

Read Float[nDopplerCells] Antenna1 voltage squared self spectra.

Read Float[nDopplerCells] Antenna2 voltage squared self spectra.

Read Float[nDopplerCells] Antenna3 voltage squared self spectra.

(Warning: Some Antenna3 amplitude values may be negative to indicate noise or interference at those doppler bins. These negative values should be absolute before use.)

Read Complex[nDopplerCells] Antenna 1 to Antenna 2 cross spectra.

Read Complex[nDopplerCells] Antenna 1 to Antenna 3 cross spectra.

Read Complex[nDopplerCells] Antenna 2 to Antenna 3 cross spectra.

if nCsKind is ≥ 2 then also read or skip over

Read Float[nDopplerCells] Quality array from zero to one in value. When this value is less than one, it means that SpectraAverager skipped over some data from the averaging.

End Repeat

End Data Section

End File

Version 6 Block Keys

Block key **TIME** is a cross spectra time stamp with more resolution than earlier versions.
Block Size is minimum sizeof following data structure.
Block Data

UInt8	**nTimeMark**;	0=start, 1=center time, 2=end time
UInt16	**nYear**	Gregorian Local Time...
UInt8	**nMonth**	
UInt8	**nDay**	
UInt8	**nHour**	
UInt8	**nMinute**	
double	**fSeconds**	
double	**fCoverageSeconds**	Coverage time of data.
double	**fHoursFromUTC**	

Block Key **ZONE** is a time zone label
Block Size is length of a zero terminated Roman ASCII string; includes zero byte. Maximum 256 bytes
Block Data
String **szTimeZone**

Block Key **CITY** Apple OS X City TimeZone which is typically easier to resolve than TimeZone.
Block Size is length of a zero terminated Roman ASCII string; includes zero byte. Maximum 256 bytes
Block Data
String **szTimeZone**

Block Key **LOCA** Location of Receiver.
Block Size is length of zero terminated Roman ASCII string; includes zero byte. Maximum 256 bytes
Block Data is minimum sizeof following data structure.
double **fLatitude**
double **fLongitude**
double **fAltitudeMeters**

Block Key **SITD** is a Site Description
Block Size is length of zero terminated Roman ASCII string; includes zero byte.
Block Data
String **szSiteDescription**

Block Key **RCVI** is a description of the receiver.
Block Size is minimum size of following data structure.
Block Data

UInt32	**nReceiverModel**	0=Unknown, 1=Awg3/Rcvr2 Chassis AC, 2=Awg3/Rcvr2 Chassis DC, 3=AllInOne, 4=Awg4 Chassis AC,5=Awg4 Chassis DC
UInt32	**nRxAntennaModel**	0=Unknown, 1=Box Loops, 4=Dome Loops, 5=TR Dome Loops
double	**fReferenceGainDB**	Receiver Gain (Loss if negative) in dB
char32	**szFirmware**	32 chars zero term string.

Block Key **TOOL** is Name,Version string of application which created or worked on the cross spectra. There should be a comma between the name and the version. There can be multiple blocks with this key; one for each tool that worked on the data.
Block Size is length of zero terminated Roman ASCII string; includes zero byte. Maximum 256 bytes
Block Data

String	**szToolNameVer**

Block Key **GLRM** is Glitch Removal Information
Block Size is minimum size of following data structure.
Block Data

UInt8	**nMethod**	0=Off,1=Point,2=Range,3=Range&Point, 4=SubDCOnly
UInt8	**nVersion**	
UInt32	**nPointsRemoved**	
UInt32	**nTimesRemoved**	
UInt32	**nSegmentsRemoved**	
double	**fPointPowerThreshold**	
double	**fRangePowerThreshold**	
double	**fRangeBinThreshold**	
UInt8	**bRemoveDC**	0 leave DC alone, non zero removed DC.

Block Key **SUPI** is Stripe Suppression Information
Block Size is minimum size of following data structure.
Block Data

UInt8	**nMethod**	0=Off,1=Normal
UInt8	**nVersion**	0
UInt8	**nMode**	0=Light,1=Heavy,2=MaxLight,3=MaxHeavy
UInt8	**nDebugMode**	0=Off,1=On
UInt32	**nDopplerSuppressed**	number of doppler cells suppressed
double	**fPowerThreshold**	

double **fRangeBinThreshold**
SInt16 **nRangeBanding**
SInt16 **nDopplerDetectionSmoothing**

Block Key **SUPM** is Stripe Suppression reduction applied
Block Size is **nSpectraChannels** times **nDopplerCells** times 4
Block Data
    Repeat For 1 to **nSpectraChannels**
        Repeat For 1 to **nDopplerCells**
        float **fSuppressionVoltageSquared**
    End Repeat
End Repeat

_
Block Key **SUPP** is Stripe Suppression phase applied
Block Size is **nSpectraChannels** times **nDopplerCells** times 4
Block Data
    Repeat For 1 to **nSpectraChannels**
        Repeat For 1 to **nDopplerCells**
            float **fPhaseDegrees**
        End Repeat
    End Repeat

Block Key **ANTG** is Receive Antenna Gain corrections. This an indication of known power balance between the receive antennas to be use by SpectraPlotterMap for plotting purposes. A measured antenna pattern or amplitude corrections do not use this as they already have any possible differences taken into account.
Block Size is **nSpectraChannels** times 8
Block Data
    Repeat For 1 to **nSpectraChannels**
        double **fGainDB**
    End Repeat

Block Key **FWIN** is Range and Doppler FFT Windowing
Block Size is minimum size of following data structure.
Block Data
UInt8 **nRangeWindowType** 0=None,1=Blackman, 2=Hamming, 3=Tukey
UInt8 **nDopplerWindowType** 0=None, 1=Blackman, 2=Hamming, 3=Tukey
double **fRangeWindowParam**

double **fDopplerWindowParam**

Block Key **IQAP** is Receiver IQ Balance Measurement/Correction
Block Size is 2 plus **nRanges** times 16
Block Data

UInt8	**nMethod**	0=Off,1=Measured,2=Corrected
UInt8	**nVersion**	1

Repeat For 1 to **nRanges**

double	**fMagnitude**
double	**fPhase**

End Repeat

Block Key **FILL** is Receiver IQ Balance Measurement/Correction
Block Size is minimum size of following data structure.
Block Data

UInt8	**nRangeMethod**	0=None, 1=Linear, 2=FFTPadding
UInt8	**nRangeMult**	1=None, 2=double,...
UInt8	**nDopplerMethod**	0=None, 1=Linear, 2=FFTPadding
UInt8	**nDopplerMult**	1=None, 2=double,...

Block Key **FOLS** is Radial/Elliptical First Order Lines to delineate Bragg
Covers all ranges in spectra. Undetermined ranges should be set to zeros.
Each First Order index is from 0 to nDopplers-1
Block Size is **nRanges** times 8
Block Data

Repeat For 1 to **nRanges**

SInt32	**nNegBraggLeftIndex**
SInt32	**nNegBraggRightIndex**
SInt32	**nPosBraggLeftIndex**
SInt32	**nPosBraggRightIndex**

End Repeat

Block Key **WOLS** is Wave Processing First Order Lines to delineate Bragg
Covers all ranges in spectra. Undetermined ranges should be set to zeros.
Each First Order index is from 0 to nDopplers-1
Block Size is nRanges times 8
Block Data

Repeat For 1 to nRanges

SInt32	**nNegBraggLeftIndex**
SInt32	**nNegBraggRightIndex**
SInt32	**nPosBraggLeftIndex**
SInt32	**nPosBraggRightIndex**

End Repeat

Block Key **BRGR** is Radial/Elliptical Bragg Rejection. Bragg can be rejected typically because ionosphere, noise or interferences is too high. Covers all ranges in spectra. Undetermined ranges should be set to zero.
Block Size is **nRanges** times 1
Block Data
Repeat For 1 to nRanges

UInt8	**nBraggReject**	0=OK, 1=RejectNegBragg, 2=RejectPosBragg, 3=RejectBoth

End Repeat

Notes

Conversion of self spectra to dBm

10 * log10(abs(self spectra)) + receiver gain
If spectra version six with RCVI block then use **fReferenceGainDB** for receiver gain; otherwise use -34.2 dB

File Validation

Because the cross spectra format does not contain any unique identifiers, you must read the header section in order to validate the contents. After reading the header the following statements should all be true.

file size must be > 10 bytes

nCsFileVersion must be ≥ 1 and ≤ 32

If nCsFileVersion is 1 then filesize > 10 and nV1Extent ≥ 0

If nCsFileVersion is 2 then filesize > 16 and nV1Extent ≥ 6 and nV2Extend ≥ 0

If nCsFileVersion is 3 then filesize > 24 and nV1Extent ≥ 14 and nV2Extend ≥ 8 and nV3Extent ≥ 0

If nCsFileVersion is 4 then filesize > 72 and nV1Extent ≥ 62 and nV2Extend ≥ 56 and nV3Extent ≥ 48 and nV4Extent ≥ 0

If nCsFileVersion is ≥ 5 then filesize > 100 and nV1Extent ≥ 90 and nV2Extend ≥ 84 and nV3Extent ≥ 76 and nV4Extent ≥ 28 and nV5Extent ≥ 0

If nCsFileVersion is ≥ 6 then nV5Extent ≥ nCS6ByteSize + 4

if nCsFileVersion is < 4 then assume nRanges is 32 and nDopplers is 512. Note, there’s not many cross spectra in existence using less than version 4. At this time almost all cross spectra are version 4. With this latest Release all new cross spectra created will be version 6.

if nCsFileVersion is < 5 then assume nSpectraChannels is 3

nRanges must be > 0 and ≤ 8192
nDopplers must be > 0 and ≤ 32768

One way to get header size is to use nV1Extent + 10

If nCsKind is ≤ 1 then file size must be ≥ HeaderSize + nRanges * nSpectraChannels * nDopplers * 36

If nCsKind is ≥ 2 then file size must be ≥ HeaderSize + nRanges * nSpectraChannels * nDopplers * 40

Revision History

First Draft Sep 26, 2016

Copyright and Disclaimer