SeaSonde Radial Site Release 6

Reduced CrossSpectra File Format

Reduced CrossSpectra files are produced by a SeaSonde Radial Site. There are normal CrossSpectra data, which is reformatted using a lossy algorithm to reduce their size. There are in a binary RIFF format similar in style to Time Series and Range Series files.

Reduced CrossSpectra are created optionally by SpectraArchiver while AnalyzingSpectra to produce Radials and/or Waves. They are also created by SpectraShortener and by the RadialWebServer when asked to upload spectra. Utility SpectraShortener will convert to/from the standard cross spectra format and allow you to adjust how lossy the algorithm is.

The reduced files are lossy because some of the original data precision is rounded off. This rounding causes a small loss of information from the original data, which is mostly extraneous noise and tests show no significant change to the output radial and wave results.

In the normal cross spectra format the values are single precision IEEE floating point. This results in 4bytes where the bits of significant data are shifted by varying exponents. This results in a fairly random distribution of one and zeros, which do not compact well by normal lossless compression utilities and sometime even grow larger due the compression overhead. This lossy method normalizes the data to a dB scale and rounds the data to a fixed precision. The data is converted to fixed integer values of varying size bytes (1,2, or 4) depending on how the data changes. This typically results in a 3 to 1 reduction of the file size. A normal compression utility (zip) can then be applied afterward for even more reduction (typically about 20%)

File Name Format

"CSR_XXXX_yyyy_mm_dd_hhmmss.csr"
where XXXX = four char code site name
where yyyy = created year ei 2009
where mm = created month 01 to 12
where dd = created day 01 to 31
where hh = created hour 00 to 23
where mm = created minute 00 to 59
where ss = created second 00 to 59

File Contents

Format is Resource Indexed File Format. The file is composed of keyed blocks of binary data where each block starts with a 4byte character type code followed by a 4byte long data size of how much data follows.

Big-Endian Byte ordering (MSB first)
IEEE floats & doubles
Twos complement integer values

The file is compose of multiple keys where each key consists of:
A 4 byte character key type code
A 4 byte integer of key data size (can be zero)
Followed by the key data, which is the data size length of bytes.

By convention, Keys with all CAPITALS have subkeys, meaning that the key's data is made up of more keys. When you read a subkey you should read the data in the key as more RIFF keys.

A key may have no data (zero size), in which case the key will contain only the type code and the zero value key size.

When Reading

If you do not recognize the key you should usually skip over it by doing a dummy read of the key's data size.
Do not expect the keys to be in order unless implicitly stated.
Keys can be repeated as needed describing new or changed information.

If you read this file on an Intel Platform or other which uses Little Endian byte ordering the first four bytes will be 'YSSC'. In which case, you will need to swap the byte order on each integer & floating point value.

Data Type Definitions

Fourcc	4bytes four character code (example 'xxxx')
Char	1byte char
LString	#bytes, string
Char[64]	64bytes, string, zero terminated
Char[ ]	[ ]bytes from key data size, zero terminated string
SInt8	1byte Signed -128 to +127 (2s Complement)
UInt8	1byte Unsigned 0 to 255
SInt16	2byte Signed -32768 to 32767(2s Complement)
UInt16	2byte Unsigned 0 to 65535
SInt24	3byte Signed (2s Complement)
SInt32	4byte Signed -2Giga to +2Giga (2s Complement)
UInt32	4byte Unsigned 0 to 4 Giga
Float	4byte IEEE single precision floating point
Double	8byte IEEE double precision floating point
Size32	4byte Unsigned 0 to 4 Gigabytes (tells how much data follows key)

File Contents Layout

Each subkey contents is indicated inside of {} brackets
Each key data content is indented in order after key.

'CSSY' Size32 - This is the first key in the file. All data is inside this key.
{
    'HEAD' Size32 - Data Description Section

{

'sign' Size32 - File signature

Fourcc	File version	'1.04'
Fourcc	SiteCode	'XXXX'
Fourcc	FileType	‘CSSY’
UInt32	UserFlags	0
chr64	FileDescription	"Codar Shortened Cross Spectra"
chr64	OwnerName	"CODAR Ocean Sensors Ltd"
chr64	Comment	""

‘scrn’ Size32 – Source cross spectra filename. Array of chars Size32 long.

'mcda' Size32 - Mac Timestamp of first sweep

UInt32	Seconds from Jan 1,1904

‘dbrf’ Size32

Double – dBm Reference.

‘cs4h’ Size32 – Normal cross spectra header information

(Typically CS ver 4)

SInt16	nCsaFileVersion	File Version 1 to latest.
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 (nCsaKind = 1) The timestamp for CSS and CSA files is the **center time** of the data (nCsaKind = 2).
SInt32	nV1Extent	Header Bytes extension (Version 4 is +62 Bytes Till Data)

-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)

- 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 nRangeCells=31, nDopplerCells=512, nFirstRangeCell=1

-Following is added info for version 4 to latest

SInt32	nCoverageMinutes	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	bOverrideSourceInfo	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' & 'CSA')
SInt32	nFirstRangeCell	Index of First Range Cell in data from zero at the receiver. ‘CSQ' files nominally use zero. 'CSS' or 'CSA' files nominally use one because CSPro cuts off the first range cell as meaningless.
Float	fRangeCellDistKm	Distance between range cells in kilometers.
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.

‘alim’ Size32 – First Order Limits (key Might not exist)

UInt32	<nType>	First Order type (zero)
UInt32	<nRange>	Number of range cell in this first order.
Float	<fRangeKm>	Distance between range cells
Float	<fBearingDeg>	Antenna Bearing
UInt32	<nFirstRange>	First Range cell (normally 1)
UInt32	<nDopplers>	Number of doppler cells in spectra
UInt32	<nReserved1>	zero
UInt32	<nReserved2>	zero
UInt32[4][]		array of first order limits in groups of 4 UInt32s for number of range cells. Each group of 4 is LeftBraggLeftLimit, LeftBraggRightLimit, RightBraggLeftLimit, RightBraggRightLimit. These are doppler cells where LeftBragg is from 1 and Right Bragg is from nDopplers/2 (DC)

‘wlim’ Size32 – Wave First Order Limits (key Might not exist)

same format as ‘alim’

} // End of HEAD

'BODY' Size32 - This key contains the repeated keys for each range cell.

{

It normally contains a list of 'indx', keys for each range cell followed by a ‘scal’ and key data for each cross spectra data type.

'indx' Size32 - This key helps to index the current sweep.

SInt32 range cell Index from zero to number of range cells –1

'scal' Size32 - This key tells how to scale the unshortened integer data to floating point

Sint32	<nType>	Scalar type (one)
Float	<fmin>	smallest value
Float	<fmax>	largest value
Float	<fscale>	scaling values (0xFFFFFFFE)

‘cs1a’ Sint32 – Reduced Encoded Self spectra for antenna 1

'scal' Size32 - This key tells how to scale the shortened integer data to floating point

‘cs2a’ Sint32 – Reduced Encoded Self spectra for antenna 2

'scal' Size32 - This key tells how to scale the shortened integer data to floating point

‘cs3a’ Sint32 – Reduced Encoded Self spectra for antenna 3

'scal' Size32 - This key tells how to scale the shortened integer data to floating point

‘c13r’ Sint32 – Reduced Real part of complex antenna 1 to 3 ratio

'scal' Size32 - This key tells how to scale the shortened integer data to floating point

‘c13i’ Sint32 – Imaginary part of complex antenna 1 to 3 ratio

'scal' Size32 - Reduced This key tells how to scale the shortened integer data to floating point

‘c23r’ Sint32 – Reduced Real part of complex antenna 2 to 3 ratio

'scal' Size32 - This key tells how to scale the shortened integer data to floating point
‘c23i’ Sint32 – Reduced Imaginary part of complex antenna 2 to 3 ratio
'scal' Size32 - This key tells how to scale the shortened integer data to floating point
‘c12r’ Sint32 – Reduced Real part of complex antenna 1 to 2 ratio
'scal' Size32 - This key tells how to scale the shortened integer data to floating point
‘c12i’ Sint32 – Reduced Imaginary part of complex antenna 1 to 2 ratio

‘csgn’ Sint32 – Bit array of the sign of the complex spectra values. This size in bytes is 6 times nDoppler Cells divided by 8 bits. 6 is for 3 complex pairs. The reduced dB values are all positive while the sign of the source complex values are stored here.
A bit value of one indicates that the corresponding complex value should be negative.
C13r doppler cell 0 is stored at bit 0 of byte 0
C13r doppler cell 1 is stored at bit 1 of byte 0
C13r doppler cell 8 is stored at bit 0 of byte 1 in this array.
C13r nDopplers – 1 is stored at bit 7 of byte nDopplers/8-1 (given that nDopplers is a multiple of eight)
C13i doppler cell 0 is stored at bit 0 of byte nDopplers/8
C13i doppler cell 1 is stored at bit 1 of byte nDopplers/8
C23r doppler cell 0 is stored at bit 0 of byte nDopplers/8*2
C23i doppler cell 0 is stored at bit 0 of byte nDopplers/8*3
C12r doppler cell 0 is stored at bit 0 of byte nDopplers/8*4
C12i doppler cell 0 is stored at bit 0 of byte nDopplers/8*5

‘asgn’ Sint32 – Bit array of the sign of the self spectra values. This size in bytes is 3 times nDoppler Cells divided by 8 bits. The reduced dB values are all positive while the sign of the source self spectra values are stored here. Typically only A3 should have negative values which is flag from CSPro to indicate removal of ship/ interference.
A bit value of one indicates that the corresponding complex value should be negative.
cs1a doppler cell 0 is stored at bit 0 of byte 0
cs1a doppler cell 1 is stored at bit 1 of byte 0
cs1a doppler cell 8 is stored at bit 0 of byte 1 in this array.
cs1a nDopplers – 1 is stored at bit 7 of byte nDopplers/8-1 (given that nDopplers is a multiple of eight)
cs2a doppler cell 0 is stored at bit 0 of byte nDopplers/8
cs2a doppler cell 1 is stored at bit 1 of byte nDopplers/8
cs3a doppler cell 0 is stored at bit 0 of byte nDopplers/8*2

'scal' Size32 - This key tells how to scale the shortened integer data to floating point
‘csqf’ Sint32 – Reduced Spectra quality array
} // End Of BODY
} // End of CSSY
'END ' Size32 - End of File key
// End Of File

How to Decode

Each block of Reduced data is decoded by:

Set a starting UInt32 tracking value to 0
Have an output array of UInt32 large enough for nDopplers.

Read the first byte of the block this will tell you what to do next.

{

Read a command byte

If command byte is 0x9C, then read next 4bytes as unsigned 32bit integer, set the tracking value to this integer and append to the output array.

If command byte is 0x94, then read the next byte as unsigned 8bit integer. This byte + 1 is the number of unsigned 32bit (4bytes) integers to follow. Append the integers to the output array. The tracking value should also be set to last unsigned integer value.

If command byte is 0xAC, then read the next 3 bytes as a 24bit signed integer, add this value to the tracking value and append the tracking value to the output array.

If command byte is 0xA4, then read the next byte as unsigned 8bit integer. This byte + 1 is the number of SInt24(3bytes) to follow. In sequence, add each one of these to the tracking value and append each new tracking value to the output array.

If command byte is 0x89, then read the next byte as a signed 8bit integer, add this value to the tracking value and append tracking value to output array.

If command byte is 0x84, then read the next 2 bytes as a 16bit signed integer, add this value to the tracking value and append tracking value to output array.

If command byte is 0x82, then read the next byte as unsigned 8bit integer. This byte + 1 is the number of Sint16(2bytes) to follow. In sequence, add each one of these to the tracking value and append each new tracking value to the output array.

If command byte is 0x81, then read the next byte as unsigned 8bit integer. This byte + 1 is the number of Sint8(1byte) to follow. In sequence, add each one of these to the tracking value and append each new tracking value to the output array.

If command byte is some other value, an error has happened.

} Now loop with the next byte in the reduced block until all bytes are processed. You should check to ensure that you don't exceed the output of nDoppler cells or the reduced block size.

Now convert the output array of fixed UInt32 values into floating point values by applying the 'scal' values. For nDopplers convert each UInt32 value by

If (value is 0xFFFFFFFF) then

Output double is NAN

Else

Output double is value * (fmax-fmin)/fscale + fmin

Then convert each double output to voltage by applying

pow(10.,(double + dbRef)/10.)

Then after you have read the sign array you will need to invert each value that has a corresponding one bit in the sign array.