OCEAN OBSERVATORIES INITIATIVE

DATA PRODUCT SPECIFICATION FOR TURBULENT VELOCITY PROFILE AND ECHO INTENSITY

Version 1-00
Document Control Number 1341-00760
2013-05-10

Consortium for Ocean Leadership
1201 New York Ave NW, 4th Floor, Washington DC 20005
www.OceanLeadership.org

in Cooperation with

University of California, San Diego
University of Washington
Woods Hole Oceanographic Institution
Oregon State University
Scripps Institution of Oceanography
Rutgers University

Document Control Sheet

Version	Date	Description	Author
0-01	2012-08-01	Initial draft	S. Denny
0-02	2012-08-10	Revised for ECHOINT as L1	S. Denny
0-03	2012-09-20	Updates from Focused review	S. Denny
0-04	2012-09-26	Formatting revisions	M. Gibney
0-05	2012-11-26	Updates in response to CI comments	S. Denny
0-06	2013-04-02	Incorporation of final edits	M. Gibney
1-00	2013-05-10	Initial Release	E. Chapman

Signature Page

This document has been reviewed and approved for release to Configuration Management.

OOI Senior Systems Engineer: ______________________________

Date: 2013-05-10

This document has been reviewed and meets the needs of the OOI Cyberinfrastructure for the purpose of coding and implementation.

OOI CI Signing Authority: ______________________________

Date: 2013-05-10

Table of Contents

1 Abstract

This document describes the computation used to calculate the OOI Level 1 Turbulent Velocity Profile and OOI Level 1 Echo Intensity core data products, which are calculated using data from Teledyne RDI 5-Beam Workhorse ADCP instruments. This DPS applies to instrument classes of VADCP only. This document is intended to be used by OOI programmers to construct appropriate processes to create the L1 Turbulent Velocity Profile and the L1 Echo Intensity data products. Auxiliary data products are identified to construct higher level data products to compensate for platform motion and orientation.

2 Introduction

2.1 Author Contact Information

Please contact Skip Denny (denny@apl.washington.edu) or the Data Product Specification lead (DPS@lists.oceanobservatories.org) for more information concerning the computation and other items in this document.

2.2 Metadata Information

2.2.1 Data Product Name

The OOI Core Data Product Names for these products are
- VELTURB
- ECHOINT

The OOI Core Data Product Descriptive Names for these products are
- Velocity Profile, vector velocity (u,v,w) of water motion accurate to 3% of full scale, with a resolution of 1 mm/s.
- Echo Intensity, signal strength of the echo returning relative to the ADCP's un-calibrated transmit pulse along the beam.

The auxiliary data products that are required from the VADCP instrument VELTURB data stream are:
- VELCOMP – Instrument Compass Heading, deg. magnetic
- VELTILT – Instrument Tilt measurement, deg., as pitch and roll, from vertical

Auxiliary data products of platform motion (3-axis accelerometer) from platform engineering instruments are required for VELTURB. These products are
- INUACCL-X – X-component of inertial motion, acceleration in mm/s^2
- INUACCL-Y– Y-component of inertial motion, acceleration in mm/s^2
- INUACCL-Z– Z-component of inertial motion, acceleration in mm/s^2

These products are available from the RSN OMS from platform engineering instruments.

2.2.2 Data Product Abstract (for Metadata)

2.2.2.1 For VELTURB:

The OOI Level 1 Turbulent Velocity Profile core data product, VELTURB, from Workhorse 5-beam ADCP provides water velocity (u, v, w) profile measurements in units of mm/s in beam coordinates with additional data (w) from the 5th, vertical beam. The data comes from 2 ports, as the instrument is configured as 2 separate ADCPs that are set to a Master/Slave relationship. Each stream will look the same, except on the 2nd one will only have one valid (5th) beam. ADCPs produce water velocity by measuring relative radial velocity between transducer and sound scatterers drifting in the water currents. The L1 VELTURB data have been corrected for tilt.

Compass heading, tilt and magnetic variation are provided for computation of higher level data products and are sent from only one (primary) unit. Platform motion data is identified for higher level data products to be constructed.

2.2.2.2 For ECHOINT:

The OOI Level 1 Echo Intensity core data product (ECHOINT) from Workhorse ADCPs provides echo intensity measurements in units of decibels (dB re: transmit pulse). ADCPs produce echo intensity by measuring the signal strength of the echo returning relative to the ADCP's un-calibrated transmit pulse along the beam.

2.2.3 Computation Name

N/A

2.2.4 Computation Abstract (for Metadata)

The OOI Level 1 Turbulent Velocity Profile core data product and the OOI Level 0 Echo Intensity core data product are computed by decoding *data ensembles* in either hexadecimal-ASCII or binary format from ADCP family of instruments into turbulent velocity profile. They must be derived from 2 instrument ports, but the auxiliary data is derived from only one of those ports, the primary, or master, 4-beam unit.

2.2.5 Instrument-Specific Metadata

Instrument-specific metadata fields that must be part of the L1 output data product VELTURB are listed in Section 4.4. Echo intensity is stored as a separate L1 core data product (ECHOINT) and it has the same metadata with the associated L1 VELTURB data product.

The PD0 output is composed of header, fixed leader data, variable leader data, velocity, correlation magnitude, echo intensity, percent good, bottom track data, and a data-validity checksum. Fixed leader and variable leader data need to be decoded to obtain instrument-specific configuration data including serial number, coordinate system configuration, bin size, and blanking distance. Details of the PD0 output data format are explained in the TRDI manual, Workhorse commands and output data format (TRDI 2010b).

Depth of transducer, speed of sound, and water temperature are required to perform sound speed corrections if data product users request.

Pitch, roll and heading representing VADCP rotation are required to correct the VADCP data when beam coordinate velocity data are converted into earth coordinates. Corrections for magnetic variation, by latitude, are also required to convert from beam to earth coordinates.

The scale of turbulence measurements will also require compensation for platform motion that will be provided by engineering instrumentation on the RSN 200m platforms from an Inertial Navigation Unit (INU) that provides 3-axis accelerations. This data is not within the science instrument data stream, but must be pulled from the RSN OMS data.

Data ensembles include four different kinds of profile data: velocity, echo intensity, correlation magnitude, and percent good. "Velocity" data are used to produce the L1 VELTURB data product. "Echo intensity" data are the L1 ECHOINT data product. "Correlation magnitude" and "Percent good" data in each data ensemble need to be stored as metadata with the L1 VELTURB data product and with the L1 ECHOINT data product. Correlation magnitude is between 0 and 255 and it represents quality of data. Percent good is the fraction of data that has passed data quality criteria such as low correlation, large error velocity, and false target threshold. Both correlation and percent good are measures of data quality, and they will be used for QA/QC procedures.

Table 1. Required metadata for the L1 VELTURB data product and the L1 ECHOINT data product. The data stream from both ports is the same, however only the beam #1 data from the secondary, slave, unit is used. Pitch, roll, tilt and compass data is only available from the primary, master 4-beam unit.

Metadata name	**Field**	**PD0 data types**	**Unit**	**Note**
Instrument frequency	System configuration	Fixed leader	Hz	
CPU firmware version	CPU F/W VER.	Fixed leader	N/A	
CPU firmware revision	CPU F/W REV.	Fixed leader	N/A	
Number of beams	Number of beams	Fixed leader	count	The number of beams used to calculate velocity data (not physical beams)
Number of cells	Number of cells	Fixed leader	count	
Pings per ensemble	Pings per ensemble	Fixed leader	count	
Depth cell length	Depth cell length	Fixed leader	cm	The length of one depth cell.
Blank after transmit	Blank after transmit	Fixed leader	cm	
Coordinate transform	Coord Transform	Fixed leader	N/A	Earth or beam coordinates
Distance to the first bin	Bin 1 distance	Fixed leader	cm	
Instrument serial number	Serial #	Fixed leader	N/A	
Beam angle	Beam angle	Fixed leader	degrees	
Ensemble number	Ensemble number	Variable leader	count	
Real-time clock year	RTC year	Variable leader	years	
Real-time clock month	RTC month	Variable leader	months	
Real-time clock day	RTC day	Variable leader	days	
Real-time clock hour	RTC hour	Variable leader	hours	
Real-time clock minute	RTC minute	Variable leader	minutes	
Real-time clock second	RTC second	Variable leader	seconds	
Real-time clock hundredths	RTC hundredths	Variable leader	hundredths	
Speed of sound	Speed of sound	Variable leader	m/s	

Depth of transducer	Depth of transducer	Variable leader	decimeters	
Heading	Heading	Variable leader	0.01 degrees	
Pitch	Pitch (Tilt 1)	Variable leader	0.01 degrees	
Roll	Roll (Tilt 2)	Variable leader	0.01 degrees	
Temperature	Temperature	Variable leader		
Pressure	Pressure	Variable leader	deca-pascal	
Correlation magnitude	Field 1, 2, 3, and 4 for each depth cell	Correlation magnitude	N/A	A linear scale between 0 and 255
Percent good	Field 1, 2, 3, and 4 for each depth cell	Echo intensity	dB	

2.2.6 Data Product Synonyms

Synonyms for the Velocity Profile data product are

- Turbulent Ocean Current
- Turbulent Velocity

2.2.7 Similar Data Products

VELPROF from ADCPx, where x=A,S,T, variations in the class of ADCPs.

2.3 Instruments

The Data Processing Flow document (DCN 1342-00760) for Workhorse 5-beam ADCP instrument VADCP describes instrument make/model, and describes the flow of velocity profile data from VADCP through all of the relevant QC, calibration, and data product computations and procedures.

VADCPs are deployed on the RSN 200m platforms of the moorings at Hydrate (PN1A) and Axial (PN3A) sites.

2.4 Literature and Reference Documents

The electronic files of the reference documents are stored on Alfresco under REFERENCE>Data Product Specification Artifacts (1341-00760_VELTURB).

Teledyne RD Instruments Workhorse Monitor, Sentinel, Mariner Technical Manual, P/N 957-6150-00 (August 2010).

Teledyne RD Instruments Workhorse 5 Beam ADCP Addendum Manual, P/N 957-6153-00 (June 2009).

Teledyne RD Instruments WinSC and PlanADCP User’s Guide, P/N 957-6159-00 (March 2009).

Teledyne RD Instruments WinADCP User’s Guide, P/N 957-6158-00 (January 2009)

Teledyne RD Instruments (2010a). ADCP Coordinate Transformation. P/N 951-6079-00.

Teledyne RD Instruments Workhorse commands and output data format. P/N 957-6156-00 (August 2010)

Teledyne RD Instruments (2011). Acoustic Doppler current profiler principles of operation: A practical primer.

Teledyne RD Instruments Tools User's Guide, P/N 957-6157-00 (July 2009).

Stringer, S., R. Dewey, "**Flow parameters from different ADCP beam configurations**", in preparation (2009) for *J.Atmos. & Ocean Tech.*

Gostiaux L., H. van Haren, "**Extracting Meaningful Information from Uncalibrated Backscattered Echo Intensity Data**", May 1, 2010, *J. Atmos & Ocean Tech.*

Stacey, M.T., Monismith, S.G. & Burau, J.R., "**Measurements of Reynolds stress profiles in tidal flows**," *Journal of Geophysical Research*, v.104(C5), pp. 10933-10949, 1999.

Terminology

2.4.1 Definitions

Definitions of general OOI terminology are contained in the Level 2 Reference Module in the OOI requirements database (DOORS). The following terms are defined here for use throughout this document.

Beam (Radial) coordinates: "These are the raw velocity measurements measured independently by each transducer, in units of millimeters per second. The sense is positive when the motion is towards the transducer. These axes are not orthogonal." (TRDI, 2010a, p.16).

Earth axes (geographic or geodetic) coordinates: "Velocity is converted into north, east and up components." (TRDI, 2011)

Instrument coordinates: This is a coordinate oriented relative to the transducer head. ADCPs have four transducers and they are labeled clockwise in the order 3-1-4-2 (TRDI, 2010a, Fig. 3, p.17). "The X-axis lies in the direction from transducer 1 towards transducer 2 and the Y-axis lies in the direction from transducer 4 toward transducer 3. The Z-axis lies along the axis of symmetry of the four beams, pointing away from the water towards the pressure case." (TRDI, 2010a, p.16).

Data ensemble: "A data ensemble consists of the data collected and averaged during the ensemble interval. A data ensemble can contain header, leader, velocity, correlation magnitude, echo intensity, percent-good, and status data." (TRDI, 2010b)

Correlation data: "A measure of data quality, and its output is scaled in units such that the expected correlation (given high signal/noise ratio, S/N) is 128." (TRDI, 2011)

Percent-good data: "A fraction of data passed a variety of criteria. Rejection criteria include low correlation, large error velocity and fish detection. Default thresholds differ for each ADCP; each threshold has an associated command." (TRDI, 2011)

2.4.2 Acronyms, Abbreviations and Notations

General OOI acronyms, abbreviations and notations are contained in the Level 2 Reference Module in the OOI requirements database (DOORS).

2.4.3 Variables and Symbols

None

3 Theory

3.1 Description

Velocity profile is measured by Workhorse series ADCP instruments including the VADCP. Velocity Profile data from VADCP instruments requires tilt corrections and correction for platform motion for beam coordinate velocities and additional compass and magnetic variance corrections as described in Step 8 of the processing flow for earth coordinates. A typical ADCP has four acoustic beams and each beam can measure a single velocity component parallel to the beam using the Doppler Effect. Three beams are necessary to obtain three dimensional velocity profiles. The fourth beam is added to compute a second vertical velocity component to estimate error velocity. The fifth (vertical) beam is a redundant beam to provide a better estimate of the vertical velocity component, w, with the assumption that the scale of turbulence is ensonified by all beams. The 5-beam VADCP is operated as 2 ADCPs with synchronization, where the second ADCP has only 1 valid beam (data for all 4 beams is provided). Broadband operation utilizes pulse compression to achieve higher resolution and better signal-to-noise (SNR) signals.

Output data format of Workhorse ADCPs can be either hexadecimal-ASCII or binary. The standard PD0 format is selected to be used by OOI. PD0 is a binary output format, which uses less storage space and has a faster transmission time than the Hex-ASCII format. The binary velocity profile data product output from the Workhorse ADCP driver, which consists of data ensembles, must be converted into decimal format to obtain an L1 velocity profile data product in mm/s and an L1 echo intensity data product in dB.

3.2 Mathematical Theory

Each beam contains components of u, v, and w:

$$b_1 = -u_1 \sin\theta - w_1 \cos\vartheta$$
$$b_2 = 2 \sin\theta - w_2 \cos\vartheta$$
$$b_3 = -v_3 \sin\theta - w_3 \cos\vartheta$$
$$b_4 = v_4 \sin\theta - w_4 \cos\vartheta$$
$$b_5 = -w_5$$

Where b_x is beam number x, x=1…5 in the Janus configuration (Figure 1).

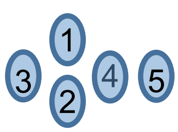

Figure 1. 5-Beam JANUS Configuration

A simplistic inversion with the assumptions:

$u_1 = u_2 = u_{12}$
$v_3 = v_4 = v_{34}$
$w_1 = w_2 = w_{12}$
$w_3 = w_4 = w_{34}$

And the current, u_{12}, is aligned with b_1, as it might be on a shipboard mounting

Yields the following set of equations:

$$u_{12} = ([u_1 + u_2]\sin\theta + [w_1 - w_2]\cos\vartheta)/2$$
$$v_{34} = ([v_3 + v_4]\sin\theta + [w_3 - w_4]\cos\vartheta)/2$$
$$w_{12} = ([w_1 + w_2]\cos\theta + [u_1 - u_2]\sin\vartheta)/2$$
$$w_{34} = ([w_3 + w_4]\cos\theta + [v_3 - v_4]\sin\vartheta)/2$$

The 5th beam can be averaged in with the results of the other 4 beams and may be weighted more heavily. The transformation matrix A (TRDI, 2010a, p.11) from beam coordinates to instrument coordinates is based on this set of assumptions:

$$A = \begin{bmatrix} c*a & -c*a & 0 & 0 & 0 \\ 0 & 0 & -c*a & c*a & 0 \\ b & b & -b & -b & b \\ d & d & -d & -d & d \end{bmatrix} \qquad \text{(Equation 1)}$$

where * denotes multiplication. The constants in Equation 1 include the dot product parameters and are defined as:

a=1/[2*sin(θ)]
b=1/[4*cos(θ)]
c=+1 for a convex transducer head, -1 for concave
d=a/sqrt(2),
where / denotes division. Since the VADCP has convex transducer head, the constant c=+1. The θ is a beam angle, from vertical, which is fixed at 20° for the 4-beam unit, but is 0° for the 5th beam.

The rotation matrix M (TRDI, 2010a, p.19) transforms velocity profiles from instrument coordinates to earth coordinates (Equation 2).

$$M = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos P & -\sin P \\ 0 & \sin P & \cos P \end{bmatrix} * \begin{bmatrix} \cos R & 0 & \sin R \\ 0 & 1 & 0 \\ -\sin R & 0 & \cos R \end{bmatrix}, \quad \text{(Equation 2)}$$

where * denotes matrix multiplication. R, and P are the roll, and pitch angles. Note that R and P in Equation 2 may not be the same as the measured roll and the measured pitch depending on the ADCP orientation configuration. Since the ADCP has upward-looking orientation, 180° must be added to the roll angle measured by internal tilt sensor (Equation 3) before applying the matrix M. And the pitch angle needs to be modified using Equation 4.

R = ***Tilt2*** +180° for upward-looking ADCP (Equation 3)

P = arctan[tan(***Tilt1***)*cos(***Tilt2***)], (Equation 4)

where ***Tilt1*** is the measured pitch and ***Tilt2*** is the measured roll recorded in data ensembles.

The rotation matrix B rotates a vector by φ degrees clockwise (Equation 5).

$$B = \begin{bmatrix} \cos\varphi & \sin\varphi \\ -\sin\varphi & \cos\varphi \end{bmatrix}. \qquad \text{(Equation 5)}$$

The real transform matrix will be more complex as components of [u, v, w] will be present in all beams. Also, this matrix ignores b_5 contributions.

3.3 Known Theoretical Limitations

Too few or too many scattering particles can limit the range of the ADCP. Large scatterers are rejected by the system. The ADCP cannot measure current if scatterers move only perpendicular to each beam, although this is practically not an issue since each beam is 20 degree apart.

3.4 Revision History

No revisions to date.

4 Implementation

4.1 Overview

Decoding ADCP binary data ensembles converts the L0 velocity profile data product into an L1 velocity profile data product in units of mm/s. The example MatLAB code written by Professor Rich Pawlowicz (http://www.eos.ubc.ca/~rich/#RDADCP, rich@eos.ubc.ca) is provided in Appendix A with his permission. Note that this is research code and is provided as an example for processing ADCP data, but does not handle bad data well as noted in the code comments. Not all functions in the example code apply to the creation of the OOI core data product VELTURB.

After each data collection cycle, the WorkHorse ADCP immediately sends a data ensemble that always includes header, fixed leader data, and variable leader data by default, and that may include velocity profile, correlation profile, echo intensity profile, percent good profile, status profile, bottom track data, etc. by user's choice.

Velocity profile data can be either raw single-ping data or multi-ping averaged data in either radial beam coordinates or earth coordinates. Single-ping and multi-ping data are treated identically and are not differentiated in this processing specification. However, single-ping data will not go through the QA/QC steps using percent good data because percent good is not available for single-ping data. The L0 velocity profile data product is converted to decimal in units of mm/s by decoding ADCP data ensembles. If L0 data product is in beam coordinates, it is transformed using matrices in section 3.2.to obtain North-South, East-West and vertical velocities in earth coordinates after decoding raw binary ADCP data files.

RSN VADCP units will stream single-ping data. The L0 data product will be transformed to a L1 data product in mm/s in beam coordinates with only tilt compensation. Higher order data products are the primary desire, but will be left to the researcher and their own methods.

4.2 Inputs

Inputs are:

- L0 VELTURB velocity data product output from the ADCP driver
- VELCOMP – Instrument Compass Heading
- VELTILT – Instrument Tilt measurement
- INUACCL-X – X-component of inertial motion
- INUACCL-Y– Y-component of inertial motion
- INUACCL-Z– Z-component of inertial motion

Input Data Format:

The most common standard data format is PD0, which provides all available information possible. PD0 is the format provided by VADCP. See the ADCP Processing Flow document (DCN 1342-00760) and associated QC lookup tables for details of the QA/QC steps.

The setup needed to get proper data is:

Table 1. 2-port VADCP Configuration

4-Beam Config, port 1	**5th Beam Config, port 2**
CR1 EX00000 WA255 WE5000 WP1	CR1 EX00000 EZ1000001 WA255 WE5000

WS100 WN40 WV175 WF88 WB1 TE00:00:02.00 TP00:00.00 CF11101 CK	WP1 WS94 WN40 WV175 WF83 WB1 TE00:00:00.00 TP00:00.00 CF11101 CK

PD0 standard output format is provided in Table 2.

Table 2. PDO standard output data buffer format (fig. 8 in TRDI 2010b, p. 129)

ALWAYS OUTPUT	**HEADER** (6 BYTES + [2 x No. OF DATA TYPES])
	FIXED LEADER DATA (59 BYTES)
	VARIABLE LEADER DATA (65 BYTES)
WD-command WP-command	**VELOCITY** (2 BYTES + 8 BYTES PER DEPTH CELL)
	CORRELATION MAGNITUDE (2 BYTES + 4 BYTES PER DEPTH CELL)
	ECHO INTENSITY (2 BYTES + 4 BYTES PER DEPTH CELL)
	PERCENT GOOD (2 BYTES + 4 BYTES PER DEPTH CELL)
BP-command	**BOTTOM TRACK DATA** (85 BYTES)
ALWAYS OUTPUT	**RESERVED** (2 BYTES)
	CHECKSUM (2 BYTES)

Example: An ADCP data output with seven data types (Fixed Leader, Variable Leader, Velocity, Correction magnitude, Echo intensity, percent good data, and Bottom track) and 30 depth cells in PD0 standard output format has 841 bytes of data per ensemble. The Velocity data is the L0 VELTURB data product. The Echo intensity is the L0 ECHOINT data product.

```
 20  BYTES OF HEADER DATA (6 + [2 x 7 Data Types])
 59  BYTES OF FIXED LEADER DATA (FIXED)
 65  BYTES OF VARIABLE LEADER DATA (FIXED)
242  BYTES OF VELOCITY DATA (2 + 8 x 30)
122  BYTES OF CORRELATION MAGNITUDE DATA (2 + 4 x 30)
122  BYTES OF ECHO INTENSITY (2 + 4 x 30)
122  BYTES OF PERCENT-GOOD DATA (2 + 4 x 30)
 85  BYTES OF BOTTOM TRACK DATA (FIXED)
  2  BYTES OF RESERVED FOR TRDI USE (FIXED)
  2  BYTES OF CHECKSUM DATA (FIXED)
841  BYTES OF DATA PER ENSEMBLE
```

Data formats of important data types are given in Table 3 (header) and Table 4 (velocity). Full description of data format for other data types can be found at TRDI (2010b).

Table 3. Output data format header

BIT POSITIONS

BYTE	7	6	5	4	3	2	1	0	
1	HEADER ID (7Fh)								
2	DATA SOURCE ID (7Fh)								
3	NUMBER OF BYTES IN ENSEMBLE								LSB
4									MSB
5	SPARE								
6	NUMBER OF DATA TYPES								
7	OFFSET FOR DATA TYPE #1								LSB
8									MSB
9	OFFSET FOR DATA TYPE #2								LSB
10									MSB
11	OFFSET FOR DATA TYPE #3								LSB
12									MSB
↓	(SEQUENCE CONTINUES FOR UP TO N DATA TYPES)								↓
2N+5	OFFSET FOR DATA TYPE #N								LSB
2N+6									MSB

Table 4. Output data format, Velocity data format (fig. 12 in TRDI, 2010b)

BYTE	BIT POSITIONS: 7/S, 6, 5, 4, 3, 2, 1, 0	
1	VELOCITY ID	LSB 00h
2		MSB 01h
3	DEPTH CELL #1, VELOCITY 1	LSB
4		MSB
5	DEPTH CELL #1, VELOCITY 2	LSB
6		MSB
7	DEPTH CELL #1, VELOCITY 3	LSB
8		MSB
9	DEPTH CELL #1, VELOCITY 4	LSB
10		MSB
11	DEPTH CELL #2, VELOCITY 1	LSB
12		MSB
13	DEPTH CELL #2, VELOCITY 2	LSB
14		MSB
15	DEPTH CELL #2, VELOCITY 3	LSB
16		MSB
17	DEPTH CELL #2, VELOCITY 4	LSB
18		MSB
↓	(SEQUENCE CONTINUES FOR UP TO 128 CELLS)	↓
1019	DEPTH CELL #128, VELOCITY 1	LSB
1020		MSB
1021	DEPTH CELL #128, VELOCITY 2	LSB
1022		MSB
1023	DEPTH CELL #128, VELOCITY 3	LSB
1024		MSB
1025	DEPTH CELL #128, VELOCITY 4	LSB
1026		MSB

4.3 Processing Flow

The decoding process recommended by the manufacturer is summarized here. Details can be found in TRDI (2010b, p.183, section 7. How to decode an ADCP ensemble) cited in Section 2.4. All the necessary algorithms for data product and QC procedures, and the order that the algorithms need to be applied can be found in the ADCP Processing Flow document (DCN 1342-00750).

The processing flow for the velocity profile computation is as follows:

Step 1:

Locate the header data by locating the header ID number. All data types have a specific and unique ID number. The most common IDs are listed in Table 2.

Table 2. Common data format IDs (Table 47 in TRDI, 2010b, p.183)

ID	Description
0x7F7F	Header
0x0000	Fixed Leader
0x0080	Variable Leader
0x0100	Velocity Profile Data
0x0200	Correlation Profile Data
0x0300	Echo Intensity Profile Data
0x0400	Percent Good Profile Data
0x0500	Status Profile Data
0x0600	Bottom Track Data
0x0800	MicroCAT Data

Step 2:
Calculate the “checksum” by summing total number of bytes in the ensemble excluding the 2-byte of checksum. Then read the 2-byte checksum at the end of the ensemble and compare it to the value you calculated. If they match, then the data ensemble is valid. If they do not match, search for the next header ID number occurrence. One example of header data format of PD0 output data format is given below (TRDI, 2010b).

Step 3:
Read the number of data types in the header data. Data types can be fixed leader, variable leader, velocity, echo intensity, correlation, percent good, etc.

Step 4:
Read the offset to each data type in the header data.

Step 5:
Locate the data ID type of velocity and metadata described in section 2.2.5. Then confirm the data ID numbers. ID numbers for all data types need to be identified to decode an ADCP ensemble.

Step 6:
Find what each byte represents in the particular data type. Record the binary data from the Velocity data type as the L0 VELTURB data product. Record the binary data from the Echo Intensity data type as the L0 ECHOINT data product. Convert the targeted binary bytes to decimal to obtain the velocity profile data product in mm/s, the L1 ECHOINT echo intensity data product in dB, and associated metadata. Additional metadata can be found inside of the fixed leader and variable leader as described in section 2.2.5. No further processing is required for the L1 ECHOINT data product.

Step 7:
The velocity profile data product is in beam coordinates, it needs to be transformed to earth coordinates by applying the transformation matrix A (Equation 1) and the rotation matrix M (Equation 2).

Velocity data for each depth cell consist of VEL1, VEL2, VEL3, and VEL4 that refer to beam1 (b1), beam2 (b2), beam3 (b3), and beam4 (b4) velocity in beam coordinates.

$$\begin{bmatrix} u \\ v \\ w \\ e \end{bmatrix} = A * \begin{bmatrix} b1 \\ \mathrm{b2} \\ \mathrm{b3} \\ \mathrm{b4} \\ b5 \end{bmatrix},$$

where * denotes matrix multiplication and the matrix A is defined in Equation 1 in Section 3.2.

Now another transformation matrix M (Equation 2) needs to be applied to adjust velocity profiles in instrument coordinates, [u,v,w] only for pitch and roll, [uu,vv,ww].

$$\begin{bmatrix} uu \\ vv \\ ww \end{bmatrix} = M * \begin{bmatrix} u \\ v \\ w \end{bmatrix},$$

where * denotes matrix multiplication and the matrix M is defined in Equation 2 in Section 3.2.

Step 8:
Apply magnetic variation correction to velocity profile components using Equation 5, if appropriate. The φ in Equation 5 is magnetic declination estimate, which is provided by the Implementing Organization for each deployment. It also can be calculated at NOAA national geophysical data center website (http://www.ngdc.noaa.gov/geomagmodels/Declination.jsp). At this writing, the declination is 15° 58' E for PN1A and 16° 25' E for PN3A, changing by 0° 8' W/year. Positive magnetic variation means that magnetic north is east of true north and negative one means that magnetic north is west of true north. Inputs are horizontal velocity profiles in earth coordinates (uu,vv).

$$\begin{bmatrix} U \\ \mathrm{V} \end{bmatrix} = B * \begin{bmatrix} uu \\ \mathrm{vv} \end{bmatrix},$$

where * denotes matrix multiplication and the matrix B is defined in Equation 5 in Section 3.2.

4.4 Outputs

The output of the velocity profile computation is

- L1 VELTURB Velocity profile in mm/s in beam coordinates with tilt correction applied:
    - L1 VELTURB-U
    - L1 VELTURB-V
    - L1 VELTURB-W
- L1 ECHOINT Echo intensity in dB

The metadata that must be included with the L1 VELTURB and with the L1 ECHOINT data product for both units are listed below. See section 2.2.5 for details.

Metadata	Type	Units
Instrument frequency	Fixed leader	Hz
Beam pattern (concave or convex)	Fixed leader	N/A
Instrument orientation (up- or down-facing)	Fixed leader	N/A
Number of beams	Fixed leader	Count
Number of cells	Fixed leader	Count
Pings per ensemble	Fixed leader	Count
Depth cell length	Fixed leader	cm
Blank after transmit	Fixed leader	cm
Coordinate transform	Fixed leader	N/A
Distance to the first bin	Fixed leader	cm
Instrument serial number	Fixed leader	N/A
Beam angle	Fixed leader	degrees
Ensemble number	Variable leader	Count
RTC year, RTC month, RTC day, RTC hour, RTC minute, RTC second, RTC hundredths	Variable leader	Years, months, days, hours, minutes, seconds, Hundredths
Speed of sound	Variable leader	m/s
Depth of transducer	Variable leader	Decimeters
Heading, pitch (Tilt1), roll (Tilt2)	Variable leader	0.01 deg
Salinity	Variable leader	
Temperature	Variable leader	
Pressure	Variable leader	Deca-Pascal
Correlation magnitude	Correlation magnitude	N/A
Percent good data	Echo intensity	dB

Automated QC algorithms are performed using correlation magnitude, percent good data and error velocity, which will generate QC flags for VELTURB (see DCN 1342-00760 Data Processing Flow Diagram for VADCP).

No automated QC is required for the L1 ECHOINT data product.

See Appendix B for a discussion of the accuracy of the output.

4.5 Computational and Numerical Considerations

4.5.1 Numerical Programming Considerations

There are no numerical programming considerations for this computation. No special numerical methods are used.

4.5.2 Computational Requirements

None.

4.6 Code Verification and Test Data Set

A raw binary ADCP output file (mooredwh-bbdf.000) and MatLAB code (adcpdemo.m) written by Professor Rich Pawlowicz are placed *on Alfresco at REFERENCE > DPS Artifacts >1341-00750_VELTURB >VELTURB_Code_adcp_matlab.zip* for code verification and test. This code is implemented into the processing stream of NEPTUNE Canada ("regional-scale underwater ocean observatory network", http://www.neptunecanada.ca/).

The raw binary ADCP output has been decoded and saved on Alfresco (adcp_demo_allEns.mat), which includes velocity profiles in beam coordinates, echo intensity and metadata. The velocity profiles in beam coordinates are transformed into instrument coordinates first using the Matlab function beam2ins (Equation 1), and then into earth coordinates using the Matlab function ins2earth (Equation 2). The output file is placed on Alfresco (adcp_demo_beam2earth.mat).

The test binary file (mooredwh-bbdf.000) has been read by the example code (adcpdemo.m). It includes 256 ensembles (ensemble number from 300 to 555) and 85 depth cells. The output data (adcp_demo_allEns.mat) have been read using Matlab and every field of the data is listed below.

```
           name: 'vadcp'
         config: [1x1 struct]
          mtime: [1x256 double]
         number: [1x256 double]
          pitch: [1x256 double]
           roll: [1x256 double]
        heading: [1x256 double]
      pitch_std: [1x256 double]
       roll_std: [1x256 double]
    heading_std: [1x256 double]
          depth: [1x256 double]
    temperature: [1x256 double]
       salinity: [1x256 double]
       pressure: [1x256 double]
   pressure_std: [1x256 double]
       east_vel: [85x256 double]
      north_vel: [85x256 double]
       vert_vel: [85x256 double]
      error_vel: [85x256 double]
           corr: [85x4x256 double]
         status: [85x4x256 double]
         intens: [85x4x256 double]
       bt_range: [4x256 double]
         bt_vel: [4x256 double]
        bt_corr: [4x256 double]
        bt_ampl: [4x256 double]
   bt_perc_good: [4x256 double]
      perc_good: [85x4x255 double]
```

The example code stores four components of velocity profiles in the output file, i.e. east_vel, north_vel, vert_vel, and error_vel. They refer to eastward velocity, northward velocity, vertical velocity, and error velocity in earth coordinates and beam 1, beam 2, beam 3, beam 4 velocities in beam coordinates. In both cases, they are in units of m/s. Note that the L1 VELTURB data product is in units of mm/s.

The velocity profiles in beam coordinates of the first ensemble (ensemble number 300) from the first to the tenth depth cells are shown below extracted from adcp_demo_allEns.mat in units of m/s.

Beam1(mm/s)	Beam2(mm/s)	Beam3(mm/s)	Beam4(mm/s)	Beam5(mm/s)
-30	180	-398	-216	
-295	-132	-436	-605	
-514	213	-131	-92	
-234	309	-473	-58	
-188	291	-443	484	
203	49	188	-5	
-325	188	-168	338	
305	373	291	175	
-204	-2	-179	-80	
-294	172	8	-549	

The velocity profiles in beam coordinates are transformed using the Matlab functions, beam2ins and ins2earth (provided below). Output data are stored in adcp_demo_beam2earth.mat in units of m/s.

uu(mm/s)	vv(mm/s)	ww (mm/s)
217.5	-336.66	140.12
-281.44	-181.54	397.73
-100.21	-1052.2	187.04
483.06	-867.64	163.66
1238	-891.92	9.0644
-245.45	258.46	-128.98
621.79	-849.69	33.362
-180.68	-87.314	-301.67
99.189	-307.34	138.41
-906.32	-546.08	196.59

To apply magnetic variation correction described in step 8 (section 4.3) to horizontal velocity vectors, obtain magnetic variation from the Implementing Organization for each platform and for each deployment. The NOAA magnetic field calculators (http://www.ngdc.noaa.gov/geomag-web/#declination) can also provide magnetic variation. Inputs for the website are date, latitude and longitude.

For instance, horizontal velocity vector in earth coordinate in ensemble number 303 of the example data file (adcp_demo_beam2earth.mat) is (uu,vv) = (441, 172) in mm/sec. If we apply magnetic variation at Station Papa (50°N, 145°W), which is 16.9604° on May 1st, 2012, horizontal velocity vector is transformed to (U,V) = (471.9936, 35.8747) in mm/sec (Figure 4) using the Matlab function mag_var.

Ens	"Eas"	"Nor"	"Mag"	"Dir"
	"mm/s"	"mm/s"	"mm/s"	"deg"
	1	1	1	1
303	441	172	473	68.70

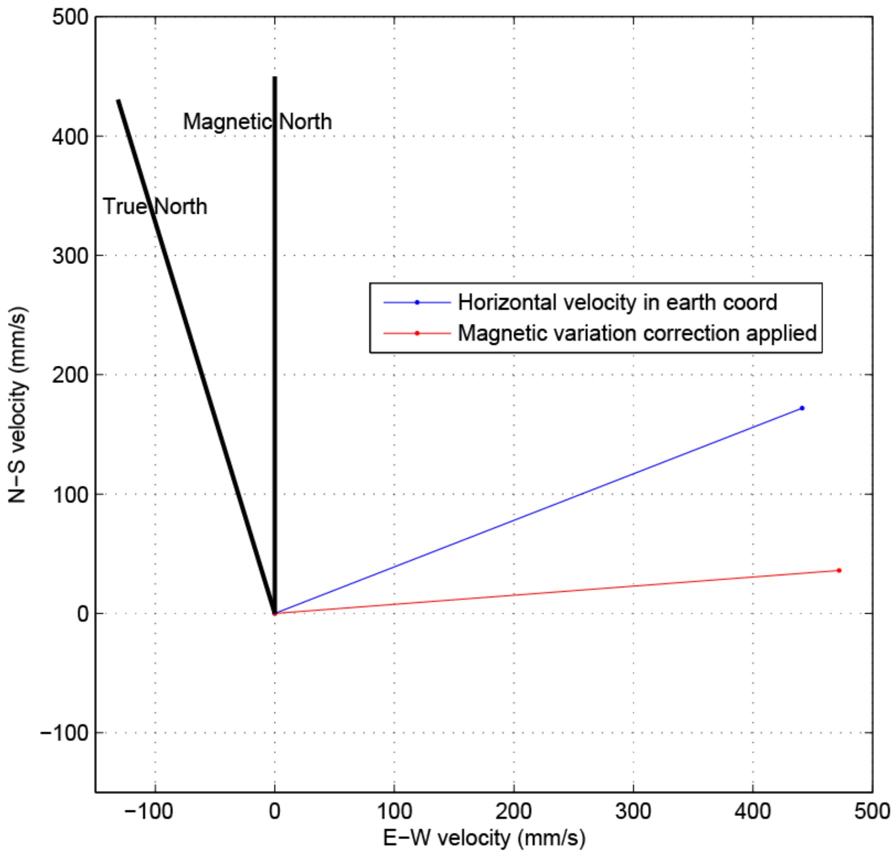

Figure 4. An example of applying the magnetic variation correction.

```
function [u,v,w,e] = beam2ins(b1,b2,b3,b4)
%
% input:
%   b1, b2, b3, b4: velocity profiles in beam coordinates
%
% output:
%   velocity profiles (u,v,w,e) in instrument coordinates
%
%

theta = 20/180*pi;% 20 deg
a = 1./(2.*sin(theta));
b = 1./(4.*cos(theta));
c = 1;% convex (Long Ranger & Quartermaster)
d = a./sqrt(2);

u = c.*a.*(b1-b2);
v = c.*a.*(b4-b3);
w = b.*(b1+b2+b3+b4);
e = d.*(b1+b2-b3-b4);

function [uu,vv,ww] = ins2earth(u,v,w,H,tilt1,tilt2,adcpmode)
%
% USAGE
```

```
% [uu,vv,ww] = ins2earth(u,v,w,H,tilt1,tilt2,adcpmode)
%
%
% input
%   u,v,w: velocity profile in instrument coordinates
%   tilt1: measured pitch, degree
%   tilt2: measured roll, degree
%   adcpmode: 0 for downward looking ADCP, 1 for upward looking ADCP
%
% output
%   uu,vv,ww: velocity profile earth coordinates
%

if adcpmode,
    R = tilt2+180;
else
    R = tilt2;
end

Rrad    = deg2rad(R);
Hrad    = deg2rad(H);
t1rad   = deg2rad(tilt1);
t2rad   = deg2rad(tilt2);

P = atan(tan(t1rad).*cos(t2rad));% rad

M1 = [cos(Hrad) sin(Hrad) 0; -sin(Hrad) cos(Hrad) 0;0 0 1];
M2 = [1 0 0;0 cos(P) -sin(P); 0 sin(P) cos(P)];
M3 = [cos(Rrad) 0 sin(Rrad); 0 1 0; -sin(Rrad) 0 cos(Rrad)];

for i=1:length(u),
    vel(i,:) = M1*M2*M3*[u(i);v(i);w(i)];
end

uu = vel(:,1);
vv = vel(:,2);
ww = vel(:,3);

function radvalue = deg2rad(degvalue)
radvalue = degvalue./180*pi;
```

Appendix A Example Code

This Appendix contains MatLAB code for reading raw binary ADCP data files written by Professor Rich Pawlowicz (http://www.eos.ubc.ca/~rich/#RDADCP, rich@eos.ubc.ca). This code has been used extensively by the oceanographic community. A copy of the code (rdradcp.m) has *been placed on Alfresco at REFERENCE > DPS Artifacts >1341-00750_VELTURB >VELTURB_Code_adcp_matlab.zip*. The demo input file, mooredwh-bbdf.000 is in beam coordinates in **PD0** data format. Note that output of this example code is always named east_vel, north_vel, vert_vel, and error_vel regardless of the coordinate system. Thus, if velocity profile is in beam coordinates, actual meaning of those velocity profile variable names is beam1, beam2, beam3, and beam4 radial velocity.

```
function [adcp,cfg,ens,hdr]=rdradcp(name,varargin);
% RDRADCP   Read (raw binary) RDI ADCP files,
%   ADCP=RDRADCP(NAME) reads the raw binary RDI BB/Workhorse ADCP file NAME and
%   puts all the relevant configuration and measured data into a data structure
%   ADCP (which is self-explanatory). This program is designed for handling data
%   recorded by moored instruments (primarily Workhorse-type but can also read
%   Broadband) and then downloaded post-deployment. For vessel-mount data I
%   usually make p-files (which integrate nav info and do coordinate transformations)
%   and then use RDPADCP.
%
%   This current version does have some handling of VMDAS, WINRIVER, and WINRIVER2 output
%   files, but it is still 'beta'. There are (inadequately documented) timestamps
% of various kinds from VMDAS, for example, and caveat emptor on WINRIVER2 NMEA data.
%
%   [ADCP,CFG]=RDRADCP(...) returns configuration data in a
%   separate data structure.
%
%   Various options can be specified on input:
%   [..]=RDRADCP(NAME,NUMAV) averages NUMAV ensembles together in the result.
%   [..]=RDRADCP(NAME,NUMAV,NENS) reads only NENS ensembles (-1 for all).
%   [..]=RDRADCP(NAME,NUMAV,[NFIRST NEND]) reads only the specified range
%    of ensembles. This is useful if you want to get rid of bad data before/after
%    the deployment period.
%
%   Notes- sometimes the ends of files are filled with garbage. In this case you may
%          have to rerun things explicitly specifying how many records to read (or the
%          last record to read). I don't handle bad data very well. Also - in Aug/2007
%          I discovered that WINRIVER-2 files can have a varying number of bytes per
%          ensemble. Thus the estimated number of ensembles in a file (based on the
%          length of the first ensemble and file size) can be too high or too low.
%
%        - I don't read in absolutely every parameter stored in the binaries;
%          just the ones that are 'most' useful. Look through the code if
%          you want to get other things.
%
%        - chaining of files does not occur (i.e. read .000, .001, etc.). Sometimes
%          a ping is split between the end of one file and the beginning of another.
%          The only way to get this data is to concatentate the files, using
%            cat file1.000 file1.001 > file1      (unix)
%            copy file1.000/B+file2.001/B file3.000/B        (DOS/Windows)
%
%          (as of Dec 2005 we can probably read a .001 file)
%
%        - velocity fields are always called east/north/vertical/error for all
%          coordinate systems even though they should be treated as
%          1/2/3/4 in beam coordinates etc.
%
%   String parameter/option pairs can be added after these initial parameters:
```

```
%
%   'baseyear'      : Base century for BB/v8WH firmware (default to 2000).
%
%   'despike'       : [ 'no' | 'yes' | 3-element vector ]
%                     Controls ensemble averaging. With 'no' a simple mean is used
%                     (default). With 'yes' a mean is applied to all values that fall
%                     within a window around the median (giving some outlier rejection).
%                     This is useful for noisy data. Window sizes are [.3 .3 .3] m/s
%                     for [ horiz_vel vert_vel error_vel ] values. If you want to
%                     change these values, set 'despike' to the 3-element vector.
%
% R. Pawlowicz (rich@eos.ubc.ca) - 17/09/99

% R. Pawlowicz - 17/Oct/99
%          5/july/00 - handled byte offsets (and mysterious 'extra" bytes) slightly better, Y2K
%          5/Oct/00 - bug fix - size of ens stayed 2 when NUMAV==1 due to initialization,
%                     hopefully this is now fixed.
%          10/Mar/02 - #bytes per record changes mysteriously,
%                      tried a more robust workaround. Guess that we have an extra
%                      2 bytes if the record length is even?
%          28/Mar/02 - added more firmware-dependent changes to format; hopefully this
%                      works for everything now (put previous changes on firmer footing?)
%          30/Mar/02 - made cfg output more intuitive by decoding things.
%                    - An early version of WAVESMON and PARSE which split out this
%                      data from a wave recorder inserted an extra two bytes per record.
%                      I have removed the code to handle this but if you need it see line 509
%         29/Nov/02   - A change in the bottom-track block for version 4.05 (very old!).
%         29/Jan/03   - Status block in v4.25 150khzBB two bytes short?
%         14/Oct/03   - Added code to at least 'ignore' WinRiver GPS blocks.
%         11/Nov/03   - VMDAS navigation block, added hooks to output
%                       navigation data.
%         26/Mar/04   - better decoding of nav blocks
%                     - better handling of weird bytes at beginning and end of file
%                       (code fixes due to Matt Drennan).
%         25/Aug/04   - fixes to "junk bytes" handling.
%         27/Jan/05   - even more fixed to junk byte handling (move 1 byte at a time rather than
%                       two for odd lengths.
%   29/Sep/2005 - median windowing done slightly incorrectly in a way which biases
%                 results in a negative way in data is *very* noisy. Now fixed.
%
%    28/Dc/2005   - redid code for recovering from ensembles that mysteriously change length, added
%                   'checkheader' to make a complete check of ensembles.
%      Feb/2006   - handling of firmware version 9 (navigator)
%    23/Aug/2006 - more firmware updates (16.27)
%    23/Aug2006   - ouput some bt QC stiff
%    29/Oct/2006 - winriver bottom track block had errors in it - now fixed.
%    30/Oct/2006 - pitch_std, roll_std now uint8 and not int8 (thanks Felipe pimenta)
%    13/Aug/2007 - added Rio Grande (firmware v 10),
%                  better handling of those cursed winriver ASCII NMEA blocks whose
%                  lengths change unpredictably.
%                  skipping the inadequately documented 2022 WINRIVER-2 NMEA block
%    13/Mar/2010 - firmware version 50 for WH.


num_av=5;     % Block filtering and decimation parameter (# ensembles to block together).
nens=-1;    % Read all ensembles.
century=2000;   % ADCP clock does not have century prior to firmware 16.05.
vels='no';     % Default to simple averaging

lv=length(varargin);
if lv>=1 & ~isstr(varargin{1}),
  num_av=varargin{1}; % Block filtering and decimation parameter (# ensembles to block together).
```

```
   varargin(1)=[];
   lv=lv-1;
   if lv>=1 & ~isstr(varargin{1}),
      nens=varargin{1};
      varargin(1)=[];
      lv=lv-1;
   end;
end;

% Read optional args
while length(varargin)>0,
 switch varargin{1}(1:3),
            case 'bas',
               century = varargin{2};
            case 'des',
               if isstr(varargin{2}),
                if strcmp(varargin{2},'no'), vels='no';
                else vels=[.3 .3 .3]; end;
               else
                vels=varargin{2};
               end;
            otherwise,
               error(['Unknown command line option   ->' varargin{1}]);
    end;
    varargin([1 2])=[];
end;




% Check file information first

naminfo=dir(name);

if isempty(naminfo),
   fprintf('ERROR******* Can''t find file %s\n',name);
   return;
end;

fprintf('\nOpening file %s\n\n',name);
fd=fopen(name,'r','ieee-le');

% Read first ensemble to initialize parameters

[ens,hdr,cfg,pos]=rd_buffer(fd,-2); % Initialize and read first two records
if ~isstruct(ens) & ens==-1,
   disp('No Valid data found');
   adcp=[];
   return;
end;
fseek(fd,pos,'bof');                     % Rewind

if (cfg.prog_ver<16.05 & cfg.prog_ver>5.999) | cfg.prog_ver<5.55,
   fprintf('***** Assuming that the century begins year %d (info not in this firmware version) \n\n',century);
else
   century=0;   % century included in clock.
end;

dats=datenum(century+ens.rtc(1,:),ens.rtc(2,:),ens.rtc(3,:),ens.rtc(4,:),ens.rtc(5,:),ens.rtc(6,:)+ens.rtc(7,:)/100
);
t_int=diff(dats);
fprintf('Record begins at %s\n',datestr(dats(1),0));
```

```
fprintf('Ping interval appears to be    %s\n',datestr(t_int,13));


% Estimate number of records (since I don't feel like handling EOFs correctly,
% we just don't read that far!)


% Now, this is a puzzle - it appears that this is not necessary in
% a firmware v16.12 sent to me, and I can't find any example for
% which it *is* necessary so I'm not sure why its there. It could be
% a leftoever from dealing with the bad WAVESMON/PARSE problem (now
% fixed) that inserted extra bytes.
% ...So its out for now.
%if cfg.prog_ver>=16.05, extrabytes=2; else extrabytes=0; end; % Extra bytes
extrabytes=0;

nensinfile=fix(naminfo.bytes/(hdr.nbyte+2+extrabytes));
fprintf('\nEstimating %d ensembles in this file\n',nensinfile);

if length(nens)==1,
  if nens==-1,
    nens=nensinfile;
  end;
  fprintf('     Reading %d ensembles, reducing by a factor of %d\n',nens,num_av);
else
  fprintf('     Reading ensembles %d-%d, reducing by a factor of %d\n',nens,num_av);
  fseek(fd,(hdr.nbyte+2+extrabytes)*(nens(1)-1),'cof');
  nens=diff(nens)+1;
end;

% Number of records after averaging.

n=fix(nens/num_av);
fprintf('Final result %d values\n',n);

if num_av>1,
  if isstr(vels),
     fprintf('\n Simple mean used for ensemble averaging\n');
  else
     fprintf('\n Averaging after outlier rejection with parameters [%f %f %f]\n',vels);
  end;
end;


% Structure to hold all ADCP data
% Note that I am not storing all the data contained in the raw binary file, merely
% things I think are useful.

switch cfg.sourceprog,
  case 'WINRIVER',
    adcp=struct('name','adcp','config',cfg,'mtime',zeros(1,n),'number',zeros(1,n),'pitch',zeros(1,n),...
                'roll',zeros(1,n),'heading',zeros(1,n),'pitch_std',zeros(1,n),...
                'roll_std',zeros(1,n),'heading_std',zeros(1,n),'depth',zeros(1,n),...
                'temperature',zeros(1,n),'salinity',zeros(1,n),...
                'pressure',zeros(1,n),'pressure_std',zeros(1,n),...

        'east_vel',zeros(cfg.n_cells,n),'north_vel',zeros(cfg.n_cells,n),'vert_vel',zeros(cfg.n_cells,n),...
                'error_vel',zeros(cfg.n_cells,n),'corr',zeros(cfg.n_cells,4,n),...
                'status',zeros(cfg.n_cells,4,n),'intens',zeros(cfg.n_cells,4,n),...
                'bt_range',zeros(4,n),'bt_vel',zeros(4,n),...
                'bt_corr',zeros(4,n),'bt_ampl',zeros(4,n),'bt_perc_good',zeros(4,n),...
                'nav_mtime',zeros(1,n),...
```

```
                        'nav_longitude',zeros(1,n),'nav_latitude',zeros(1,n));
  case 'VMDAS',
    adcp=struct('name','adcp','config',cfg,'mtime',zeros(1,n),'number',zeros(1,n),'pitch',zeros(1,n),...
                        'roll',zeros(1,n),'heading',zeros(1,n),'pitch_std',zeros(1,n),...
                        'roll_std',zeros(1,n),'heading_std',zeros(1,n),'depth',zeros(1,n),...
                        'temperature',zeros(1,n),'salinity',zeros(1,n),...
                        'pressure',zeros(1,n),'pressure_std',zeros(1,n),...

           'east_vel',zeros(cfg.n_cells,n),'north_vel',zeros(cfg.n_cells,n),'vert_vel',zeros(cfg.n_cells,n),...
                        'error_vel',zeros(cfg.n_cells,n),'corr',zeros(cfg.n_cells,4,n),...
                        'status',zeros(cfg.n_cells,4,n),'intens',zeros(cfg.n_cells,4,n),...
                        'bt_range',zeros(4,n),'bt_vel',zeros(4,n),...
                        'bt_corr',zeros(4,n),'bt_ampl',zeros(4,n),'bt_perc_good',zeros(4,n),...
                        'nav_smtime',zeros(1,n),'nav_emtime',zeros(1,n),...
                        'nav_slongitude',zeros(1,n),'nav_elongitude',zeros(1,n),...
                        'nav_slatitude',zeros(1,n),'nav_elatitude',zeros(1,n),'nav_mtime',zeros(1,n));
  otherwise
    adcp=struct('name','adcp','config',cfg,'mtime',zeros(1,n),'number',zeros(1,n),'pitch',zeros(1,n),...
                        'roll',zeros(1,n),'heading',zeros(1,n),'pitch_std',zeros(1,n),...
                        'roll_std',zeros(1,n),'heading_std',zeros(1,n),'depth',zeros(1,n),...
                        'temperature',zeros(1,n),'salinity',zeros(1,n),...
                        'pressure',zeros(1,n),'pressure_std',zeros(1,n),...

           'east_vel',zeros(cfg.n_cells,n),'north_vel',zeros(cfg.n_cells,n),'vert_vel',zeros(cfg.n_cells,n),...
                        'error_vel',zeros(cfg.n_cells,n),'corr',zeros(cfg.n_cells,4,n),...
                        'status',zeros(cfg.n_cells,4,n),'intens',zeros(cfg.n_cells,4,n),...
                        'bt_range',zeros(4,n),'bt_vel',zeros(4,n),...
                        'bt_corr',zeros(4,n),'bt_ampl',zeros(4,n),'bt_perc_good',zeros(4,n));
end;


% Calibration factors for backscatter data

clear global ens
% Loop for all records
for k=1:n,

  % Gives display so you know something is going on...

  if rem(k,50)==0,   fprintf('\n%d',k*num_av);end;
  fprintf('.');

  % Read an ensemble

  ens=rd_buffer(fd,num_av);

  if ~isstruct(ens), % If aborting...
    fprintf('Only %d records found..suggest re-running RDRADCP using this parameter\n',(k-1)*num_av);
    fprintf('(If this message preceded by a POSSIBLE PROGRAM PROBLEM message, re-run using
%d)\n',(k-1)*num_av-1);
    break;
  end;


dats=datenum(century+ens.rtc(1,:),ens.rtc(2,:),ens.rtc(3,:),ens.rtc(4,:),ens.rtc(5,:),ens.rtc(6,:)+ens.rtc(7,:)/100
);
  adcp.mtime(k)=median(dats);
  adcp.number(k)       =ens.number(1);
  adcp.heading(k)      =mean(ens.heading);
  adcp.pitch(k)        =mean(ens.pitch);
  adcp.roll(k)         =mean(ens.roll);
  adcp.heading_std(k) =mean(ens.heading_std);
```

```
  adcp.pitch_std(k)     =mean(ens.pitch_std);
  adcp.roll_std(k)      =mean(ens.roll_std);
  adcp.depth(k)         =mean(ens.depth);
  adcp.temperature(k) =mean(ens.temperature);
  adcp.salinity(k)      =mean(ens.salinity);
  adcp.pressure(k)      =mean(ens.pressure);
  adcp.pressure_std(k)=mean(ens.pressure_std);

  if isstr(vels),
    adcp.east_vel(:,k)      =nmean(ens.east_vel ,2);
    adcp.north_vel(:,k)     =nmean(ens.north_vel,2);
    adcp.vert_vel(:,k)      =nmean(ens.vert_vel ,2);
    adcp.error_vel(:,k)     =nmean(ens.error_vel,2);
  else
   adcp.east_vel(:,k)      =nmedian(ens.east_vel   ,vels(1),2);
   adcp.north_vel(:,k)     =nmedian(ens.north_vel,vels(1),2);
   adcp.vert_vel(:,k)      =nmedian(ens.vert_vel   ,vels(2),2);
   adcp.error_vel(:,k)     =nmedian(ens.error_vel,vels(3),2);
  end;

  adcp.corr(:,:,k)         =nmean(ens.corr,3);            % added correlation RKD 9/00
  adcp.status(:,:,k)          =nmean(ens.status,3);

  adcp.intens(:,:,k)        =nmean(ens.intens,3);
  adcp.perc_good(:,:,k)   =nmean(ens.percent,3);   % felipe pimenta aug. 2006


  adcp.bt_range(:,k)     =nmean(ens.bt_range,2);
  adcp.bt_vel(:,k)        =nmean(ens.bt_vel,2);

  adcp.bt_corr(:,k)=nmean(ens.bt_corr,2);                % felipe pimenta aug. 2006
  adcp.bt_ampl(:,k)=nmean(ens.bt_ampl,2);                 %    "
  adcp.bt_perc_good(:,k)=nmean(ens.bt_perc_good,2);%    "

  switch cfg.sourceprog,
    case 'WINRIVER',
     adcp.nav_mtime(k)=nmean(ens.smtime);
     adcp.nav_longitude(k)=nmean(ens.slongitude);
     adcp.nav_latitude(k)=nmean(ens.slatitude);
   case 'VMDAS',
     adcp.nav_smtime(k)     =ens.smtime(1);
     adcp.nav_emtime(k)     =ens.emtime(1);
     adcp.nav_slatitude(k)=ens.slatitude(1);
     adcp.nav_elatitude(k)=ens.elatitude(1);
     adcp.nav_slongitude(k)=ens.slongitude(1);
     adcp.nav_elongitude(k)=ens.elongitude(1);
     adcp.nav_mtime(k)=nmean(ens.nmtime);
  end;
end;

fprintf('\n');
fprintf('Read to byte %d in a file of size %d bytes\n',ftell(fd),naminfo.bytes);
if ftell(fd)+hdr.nbyte<naminfo.bytes,
  fprintf('-->There may be another %d ensembles unread\n',fix((naminfo.bytes-ftell(fd))/(hdr.nbyte+2)));
end;

fclose(fd);

%-------------------------------------
function valid=checkheader(fd);

%disp('checking');
```

```
valid=0;
numbytes=fread(fd,1,'int16');              % Following the header bytes is numbytes
if numbytes>0,                                            % and we move forward numbytes>0
   status=fseek(fd,numbytes-2,'cof');
   if status==0,
     cfgid=fread(fd,2,'uint8');
     if length(cfgid)==2,                   % will Skip the last ensemble (sloppy code)
       fseek(fd,-numbytes-2,'cof');
    %% fprintf([dec2hex(cfgid(1)) ' ' dec2hex(cfgid(2)) '\n']);
       if cfgid(1)==hex2dec('7F') & cfgid(2)==hex2dec('7F')          % and we have *another* 7F7F
         valid=1;                                                        % ...ONLY THEN it is valid.
       end;
     end;
   end;
 else
   fseek(fd,-2,'cof');
 end;


%-------------------------------------
function [hdr,pos]=rd_hdr(fd);
% Read config data
% Changed by Matt Brennan to skip weird stuff at BOF (apparently
% can happen when switching from one flash card to another
% in moored ADCPs).

cfgid=fread(fd,2,'uint8');
nread=0;
while (cfgid(1)~=hex2dec('7F') | cfgid(2)~=hex2dec('7F'))   | ~checkheader(fd),
    nextbyte=fread(fd,1,'uint8');
    pos=ftell(fd);
    nread=nread+1;
    if isempty(nextbyte),   % End of file
        disp('EOF reached before finding valid cfgid');
        hdr=NaN;
        return;
    end
    cfgid(2)=cfgid(1);cfgid(1)=nextbyte;
    if mod(pos,1000)==0
        disp(['Still looking for valid cfgid at file position ' num2str(pos) '...'])
    end
end;

pos=ftell(fd)-2;
if nread>0,
  disp(['Junk found at BOF...skipping ' int2str(nread) ' bytes until ']);
  disp(['cfgid=' dec2hex(cfgid(1)) dec2hex(cfgid(2)) ' at file pos ' num2str(pos)])
end;

hdr=rd_hdrseg(fd);

%-------------------------------------
function cfg=rd_fix(fd);
% Read config data

cfgid=fread(fd,1,'uint16');
if cfgid~=hex2dec('0000'),
 warning(['Fixed header ID ' cfgid 'incorrect - data corrupted or not a BB/WH raw file?']);
end;

cfg=rd_fixseg(fd);
```

```
%--------------------------------------
function [hdr,nbyte]=rd_hdrseg(fd);
% Reads a Header

hdr.nbyte          =fread(fd,1,'int16');
fseek(fd,1,'cof');
ndat=fread(fd,1,'int8');
hdr.dat_offsets    =fread(fd,ndat,'int16');
nbyte=4+ndat*2;

%-------------------------------------
function opt=getopt(val,varargin);
% Returns one of a list (0=first in varargin, etc.)
if val+1>length(varargin),
        opt='unknown';
else
    opt=varargin{val+1};
end;

%
%-------------------------------------
function [cfg,nbyte]=rd_fixseg(fd);
% Reads the configuration data from the fixed leader

%%disp(fread(fd,10,'uint8'))
%%fseek(fd,-10,'cof');

cfg.name='wh-adcp';
cfg.sourceprog='instrument';   % default - depending on what data blocks are
                                  % around we can modify this later in rd_buffer.
cfg.prog_ver       =fread(fd,1,'uint8')+fread(fd,1,'uint8')/100;

% 8,9,16 - WH navigator
% 10 -rio grande
% 15, 17 - NB
% 19 - REMUS, or customer specific
% 11- H-ADCP
% 31 - Streampro
% 34 - NEMO
% 50 - WH, no bottom track (built on 16.31)
% 51 - WH, w/ bottom track
% 52 - WH, mariner

if fix(cfg.prog_ver)==4 | fix(cfg.prog_ver)==5,
     cfg.name='bb-adcp';
elseif fix(cfg.prog_ver)==8 | fix(cfg.prog_ver)==9 | fix(cfg.prog_ver)==10 | fix(cfg.prog_ver)==16 ...
      | fix(cfg.prog_ver)==50 | fix(cfg.prog_ver)==51 | fix(cfg.prog_ver)==52,
     cfg.name='wh-adcp';
elseif fix(cfg.prog_ver)==14 | fix(cfg.prog_ver)==23,   % phase 1 and phase 2
     cfg.name='os-adcp';
else
     cfg.name='unrecognized firmware version'    ;
end;

config           =fread(fd,2,'uint8');   % Coded stuff
cfg.config         =[dec2base(config(2),2,8) '-' dec2base(config(1),2,8)];
 cfg.beam_angle      =getopt(bitand(config(2),3),15,20,30);
 cfg.numbeams        =getopt(bitand(config(2),16)==16,4,5);
 cfg.beam_freq       =getopt(bitand(config(1),7),75,150,300,600,1200,2400,38);
 cfg.beam_pattern    =getopt(bitand(config(1),8)==8,'concave','convex'); % 1=convex,0=concave
```

```
 cfg.orientation      =getopt(bitand(config(1),128)==128,'down','up');      % 1=up,0=down
cfg.simflag           =getopt(fread(fd,1,'uint8'),'real','simulated'); % Flag for simulated data
fseek(fd,1,'cof');
cfg.n_beams           =fread(fd,1,'uint8');
cfg.n_cells           =fread(fd,1,'uint8');
cfg.pings_per_ensemble=fread(fd,1,'uint16');
cfg.cell_size         =fread(fd,1,'uint16')*.01;   % meters
cfg.blank             =fread(fd,1,'uint16')*.01;            % meters
cfg.prof_mode         =fread(fd,1,'uint8');             %
cfg.corr_threshold =fread(fd,1,'uint8');
cfg.n_codereps        =fread(fd,1,'uint8');
cfg.min_pgood         =fread(fd,1,'uint8');
cfg.evel_threshold =fread(fd,1,'uint16');
cfg.time_between_ping_groups=sum(fread(fd,3,'uint8').*[60 1 .01]'); % seconds
coord_sys         =fread(fd,1,'uint8');                                         % Lots of bit-mapped info
  cfg.coord=dec2base(coord_sys,2,8);
  cfg.coord_sys         =getopt(bitand(bitshift(coord_sys,-3),3),'beam','instrument','ship','earth');
  cfg.use_pitchroll   =getopt(bitand(coord_sys,4)==4,'no','yes');
  cfg.use_3beam         =getopt(bitand(coord_sys,2)==2,'no','yes');
  cfg.bin_mapping      =getopt(bitand(coord_sys,1)==1,'no','yes');
cfg.xducer_misalign=fread(fd,1,'int16')*.01;      % degrees
cfg.magnetic_var     =fread(fd,1,'int16')*.01;  % degrees
cfg.sensors_src      =dec2base(fread(fd,1,'uint8'),2,8);
cfg.sensors_avail   =dec2base(fread(fd,1,'uint8'),2,8);
cfg.bin1_dist         =fread(fd,1,'uint16')*.01;  % meters
cfg.xmit_pulse        =fread(fd,1,'uint16')*.01;% meters
cfg.water_ref_cells=fread(fd,2,'uint8');
cfg.fls_target_threshold =fread(fd,1,'uint8');
fseek(fd,1,'cof');
cfg.xmit_lag          =fread(fd,1,'uint16')*.01; % meters
nbyte=40;

if fix(cfg.prog_ver)==8 | fix(cfg.prog_ver)==10 | fix(cfg.prog_ver)==16 ...
     | fix(cfg.prog_ver)==50 | fix(cfg.prog_ver)==51 | fix(cfg.prog_ver)==52,

  if cfg.prog_ver>=8.14,   % Added CPU serial number with v8.14
    cfg.serialnum         =fread(fd,8,'uint8');
    nbyte=nbyte+8;
  end;

  if cfg.prog_ver>=8.24,   % Added 2 more bytes with v8.24 firmware
    cfg.sysbandwidth   =fread(fd,2,'uint8');
    nbyte=nbyte+2;
  end;

  if cfg.prog_ver>=16.05,                             % Added 1 more bytes with v16.05 firmware
    cfg.syspower         =fread(fd,1,'uint8');
    nbyte=nbyte+1;
  end;

  if cfg.prog_ver>=16.27,     % Added bytes for REMUS, navigators, and HADCP
    cfg.navigator_basefreqindex=fread(fd,1,'uint8');
    nbyte=nbyte+1;
    cfg.remus_serialnum=fread(fd,4,'uint8');
    nbyte=nbyte+4;
    cfg.h_adcp_beam_angle=fread(fd,1,'uint8');
    nbyte=nbyte+1;
  end;

elseif fix(cfg.prog_ver)==9,

  if cfg.prog_ver>=9.10,   % Added CPU serial number with v8.14
```

```
      cfg.serialnum         =fread(fd,8,'uint8');
      nbyte=nbyte+8;
      cfg.sysbandwidth   =fread(fd,2,'uint8');
      nbyte=nbyte+2;
   end;

elseif fix(cfg.prog_ver)==14 | fix(cfg.prog_ver)==23,

      cfg.serialnum         =fread(fd,8,'uint8');    % 8 bytes 'reserved'
      nbyte=nbyte+8;

end;

% It is useful to have this precomputed.

cfg.ranges=cfg.bin1_dist+[0:cfg.n_cells-1]'*cfg.cell_size;
if cfg.orientation==1, cfg.ranges=-cfg.ranges; end


%-----------------------------
function [ens,hdr,cfg,pos]=rd_buffer(fd,num_av);

% To save it being re-initialized every time.
global ens hdr

% A fudge to try and read files not handled quite right.
global FIXOFFSET SOURCE

% If num_av<0 we are reading only 1 element and initializing
if num_av<0,
 SOURCE=0;
end;
% This reinitializes to whatever length of ens we want to average.
if num_av<0 | isempty(ens),
 FIXOFFSET=0;
 n=abs(num_av);
 [hdr,pos]=rd_hdr(fd);
 if ~isstruct(hdr), ens=-1; cfg=NaN; return; end;
 cfg=rd_fix(fd);
 fseek(fd,pos,'bof');
 clear global ens
 global ens


ens=struct('number',zeros(1,n),'rtc',zeros(7,n),'BIT',zeros(1,n),'ssp',zeros(1,n),'depth',zeros(1,n),'pitch',zeros(
1,n),...
                  'roll',zeros(1,n),'heading',zeros(1,n),'temperature',zeros(1,n),'salinity',zeros(1,n),...
                  'mpt',zeros(1,n),'heading_std',zeros(1,n),'pitch_std',zeros(1,n),...
                  'roll_std',zeros(1,n),'adc',zeros(8,n),'error_status_wd',zeros(1,n),...
                  'pressure',zeros(1,n),'pressure_std',zeros(1,n),...
                  'east_vel',zeros(cfg.n_cells,n),'north_vel',zeros(cfg.n_cells,n),'vert_vel',zeros(cfg.n_cells,n),...
                  'error_vel',zeros(cfg.n_cells,n),'intens',zeros(cfg.n_cells,4,n),'percent',zeros(cfg.n_cells,4,n),...
                  'corr',zeros(cfg.n_cells,4,n),'status',zeros(cfg.n_cells,4,n),...
                  'bt_range',zeros(4,n),'bt_vel',zeros(4,n),...
                  'bt_corr',zeros(4,n),'bt_ampl',zeros(4,n),'bt_perc_good',zeros(4,n),...
                  'smtime',zeros(1,n),'emtime',zeros(1,n),'slatitude',zeros(1,n),...
                  'slongitude',zeros(1,n),'elatitude',zeros(1,n),'elongitude',zeros(1,n),...
                  'nmtime',zeros(1,n),'flags',zeros(1,n));
   num_av=abs(num_av);
end;

k=0;
```

```
while k<num_av,

    % This is in case junk appears in the middle of a file.
    num_search=6000;

    id1=fread(fd,2,'uint8');

    search_cnt=0;
    while search_cnt<num_search & ((id1(1)~=hex2dec('7F') | id1(2)~=hex2dec('7F') ) | ~checkheader(fd) ),
         search_cnt=search_cnt+1;
         nextbyte=fread(fd,1,'uint8');
         if isempty(nextbyte),    % End of file
              disp(['EOF reached after ' num2str(search_cnt) ' bytes searched for next valid ensemble start'])
              ens=-1;
              return;
         end;
         id1(2)=id1(1);id1(1)=nextbyte;
% fprintf([dec2hex(id1(1)) '--' dec2hex(id1(2)) '\n']);
    end;
    if search_cnt==num_search,
           error(sprintf('Searched %d entries...Not a workhorse/broadband file or bad data encountered: ->
%x',search_cnt,id1));
    elseif search_cnt>0
         disp(['Searched ' int2str(search_cnt) ' bytes to find next valid ensemble start'])
    end


    startpos=ftell(fd)-2;    % Starting position.


    % Read the # data types.
    [hdr,nbyte]=rd_hdrseg(fd);
    byte_offset=nbyte+2;
%% fprintf('# data types = %d\n    ',(length(hdr.dat_offsets)));
%% fprintf('Blocklen = %d\n    ',hdr.nbyte);
      % Read all the data types.
    for n=1:length(hdr.dat_offsets),

      id=dec2hex(fread(fd,1,'uint16'),4);
%%     fprintf('ID=%s SOURCE=%d\n',id,SOURCE);

      % handle all the various segments of data. Note that since I read the IDs as a two
      % byte number in little-endian order the high and low bytes are exchanged compared to
      % the values given in the manual.
      %
      winrivprob=0;

      switch id,
       case '0000',      % Fixed leader
        [cfg,nbyte]=rd_fixseg(fd);
        nbyte=nbyte+2;

      case '0080'      % Variable Leader
        k=k+1;
        ens.number(k)              =fread(fd,1,'uint16');
        ens.rtc(:,k)              =fread(fd,7,'uint8');
        ens.number(k)              =ens.number(k)+65536*fread(fd,1,'uint8');
        ens.BIT(k)                 =fread(fd,1,'uint16');
        ens.ssp(k)                 =fread(fd,1,'uint16');
        ens.depth(k)               =fread(fd,1,'uint16')*.1;      % meters
        ens.heading(k)             =fread(fd,1,'uint16')*.01;    % degrees
        ens.pitch(k)              =fread(fd,1,'int16')*.01;      % degrees
```

```
ens.roll(k)             =fread(fd,1,'int16')*.01;    % degrees
ens.salinity(k)         =fread(fd,1,'int16');        % PSU
ens.temperature(k)      =fread(fd,1,'int16')*.01;    % Deg C
ens.mpt(k)              =sum(fread(fd,3,'uint8').*[60 1 .01]'); % seconds
ens.heading_std(k)      =fread(fd,1,'uint8');        % degrees
ens.pitch_std(k)        =fread(fd,1,'uint8')*.1;     % degrees
ens.roll_std(k)         =fread(fd,1,'uint8')*.1;     % degrees
ens.adc(:,k)            =fread(fd,8,'uint8');
nbyte=2+40;

if strcmp(cfg.name,'bb-adcp'),

    if cfg.prog_ver>=5.55,
        fseek(fd,15,'cof'); % 14 zeros and one byte for number WM4 bytes
        cent=fread(fd,1,'uint8');              % possibly also for 5.55-5.58 but
        ens.rtc(:,k)=fread(fd,7,'uint8');      % I have no data to test.
        ens.rtc(1,k)=ens.rtc(1,k)+cent*100;
        nbyte=nbyte+15+8;
    end;

elseif strcmp(cfg.name,'wh-adcp'), % for WH versions.

    ens.error_status_wd(k)=fread(fd,1,'uint32');
    nbyte=nbyte+4;;

    if fix(cfg.prog_ver)==8 | fix(cfg.prog_ver)==10 | fix(cfg.prog_ver)==16 ...
         | fix(cfg.prog_ver)==50 | fix(cfg.prog_ver)==51 | fix(cfg.prog_ver)==52,

        if cfg.prog_ver>=8.13,  % Added pressure sensor stuff in 8.13
            fseek(fd,2,'cof');
            ens.pressure(k)       =fread(fd,1,'uint32');
            ens.pressure_std(k)   =fread(fd,1,'uint32');
            nbyte=nbyte+10;
        end;

        if cfg.prog_ver>=8.24,  % Spare byte added 8.24
            fseek(fd,1,'cof');
            nbyte=nbyte+1;
        end;


        if ( cfg.prog_ver>=10.01 & cfg.prog_ver<=10.99 ) ...
            | cfg.prog_ver>=16.05,    % Added more fields with century in clock 16.05
            cent=fread(fd,1,'uint8');
            ens.rtc(:,k)=fread(fd,7,'uint8');
            ens.rtc(1,k)=ens.rtc(1,k)+cent*100;
            nbyte=nbyte+8;
        end;

    elseif fix(cfg.prog_ver)==9,

            fseek(fd,2,'cof');
            ens.pressure(k)       =fread(fd,1,'uint32');
            ens.pressure_std(k)   =fread(fd,1,'uint32');
            nbyte=nbyte+10;

        if cfg.prog_ver>=9.10,  % Spare byte added 8.24
            fseek(fd,1,'cof');
            nbyte=nbyte+1;
        end;

    end;
```

```
   elseif strcmp(cfg.name,'os-adcp'),

        fseek(fd,16,'cof'); % 30 bytes all set to zero, 14 read above
        nbyte=nbyte+16;

        if cfg.prog_ver>23,
             fseek(fd,2,'cof');
             nbyte=nbyte+2;
        end;
   end;

  case '0100',   % Velocities
   vels=fread(fd,[4 cfg.n_cells],'int16')'*.001;      % m/s
   ens.east_vel(:,k) =vels(:,1);
   ens.north_vel(:,k)=vels(:,2);
   ens.vert_vel(:,k) =vels(:,3);
   ens.error_vel(:,k)=vels(:,4);
   nbyte=2+4*cfg.n_cells*2;

  case '0200',   % Correlations
   ens.corr(:,:,k)    =fread(fd,[4 cfg.n_cells],'uint8')';
   nbyte=2+4*cfg.n_cells;

  case '0300',   % Echo Intensities
   ens.intens(:,:,k)    =fread(fd,[4 cfg.n_cells],'uint8')';
   nbyte=2+4*cfg.n_cells;

  case '0400',   % Percent good
   ens.percent(:,:,k)    =fread(fd,[4 cfg.n_cells],'uint8')';
   nbyte=2+4*cfg.n_cells;

  case '0500',   % Status
   if strcmp(cfg.name,'os-adcp'),
     fseek(fd,00,'cof');
     nbyte=2+00;
   else
      % Note in one case with a 4.25 firmware SC-BB, it seems like
      % this block was actually two bytes short!
     ens.status(:,:,k)    =fread(fd,[4 cfg.n_cells],'uint8')';
     nbyte=2+4*cfg.n_cells;
   end;

  case '0600', % Bottom track
              % In WINRIVER GPS data is tucked into here in odd ways, as long
              % as GPS is enabled.
   if SOURCE==2,
        fseek(fd,2,'cof');
        long1=fread(fd,1,'uint16');
        fseek(fd,6,'cof');
        cfac=180/2^31;
        ens.slatitude(k)  =fread(fd,1,'int32')*cfac;
        if ens.slatitude(k)==0, ens.slatitude(k)=NaN; end;
%%fprintf('\n k %8.3f',ens.slatitude(k));
   else
        fseek(fd,14,'cof'); % Skip over a bunch of stuff
   end;
   ens.bt_range(:,k)=fread(fd,4,'uint16')*.01; %
   ens.bt_vel(:,k)  =fread(fd,4,'int16');
   ens.bt_corr(:,k)=fread(fd,4,'uint8');       % felipe pimenta aug. 2006
   ens.bt_ampl(:,k)=fread(fd,4,'uint8');        % "
   ens.bt_perc_good(:,k)=fread(fd,4,'uint8'); % "
```

```
        if SOURCE==2,
            fseek(fd,2,'cof');
            ens.slongitude(k)=(long1+65536*fread(fd,1,'uint16'))*cfac;
%%fprintf('\n k %d %8.3f %f ',long1,ens.slongitude(k),(ens.slongitude(k)/cfac-long1)/65536);
            if ens.slongitude(k)>180, ens.slongitude(k)=ens.slongitude(k)-360; end;
            if ens.slongitude(k)==0, ens.slongitude(k)=NaN; end;
            fseek(fd,16,'cof');
            qual=fread(fd,1,'uint8');
            if qual==0,
%%              fprintf('qual==%d,%f %f',qual,ens.slatitude(k),ens.slongitude(k));
                ens.slatitude(k)=NaN;ens.slongitude(k)=NaN;
            end;
            fseek(fd,71-45-21,'cof');
        else
            fseek(fd,71-45,'cof');
        end;
        nbyte=2+68;
        if cfg.prog_ver>=5.3,    % Version 4.05 firmware seems to be missing these last 11 bytes.
         fseek(fd,78-71,'cof');
         ens.bt_range(:,k)=ens.bt_range(:,k)+fread(fd,4,'uint8')*655.36;
         nbyte=nbyte+11;

         if strcmp(cfg.name,'wh-adcp'),

           if cfg.prog_ver>=16.20,    % RDI documentation claims these extra bytes were added in v 8.17
               fseek(fd,4,'cof');  % but they don't appear in my 8.33 data - conversation with
               nbyte=nbyte+4;      % Egil suggests they were added in 16.20
           end;
         end;
        end;

% The raw files produced by VMDAS contain a binary navigation data
% block.

      case '2000',   % Something from VMDAS.
        cfg.sourceprog='VMDAS';
        if SOURCE~=1, fprintf('\n***** Apparently a VMDAS file \n\n'); end;
        SOURCE=1;
        utim   =fread(fd,4,'uint8');
        mtime =datenum(utim(3)+utim(4)*256,utim(2),utim(1));
        ens.smtime(k)       =mtime+fread(fd,1,'uint32')/8640000;
        fseek(fd,4,'cof');   % PC clock offset from UTC
        cfac=180/2^31;
        ens.slatitude(k)   =fread(fd,1,'int32')*cfac;
        ens.slongitude(k) =fread(fd,1,'int32')*cfac;
        ens.emtime(k)       =mtime+fread(fd,1,'uint32')/8640000;
        ens.elatitude(k)   =fread(fd,1,'int32')*cfac;
        ens.elongitude(k) =fread(fd,1,'int32')*cfac;
        fseek(fd,12,'cof');
        ens.flags(k)         =fread(fd,1,'uint16');
        fseek(fd,6,'cof');
        utim   =fread(fd,4,'uint8');
        mtime =datenum(utim(1)+utim(2)*256,utim(4),utim(3));
        ens.nmtime(k)       =mtime+fread(fd,1,'uint32')/8640000;
                             % in here we have 'ADCP clock' (not sure how this
                             % differs from RTC (in header) and UTC (earlier in this block).
        fseek(fd,16,'cof');
        nbyte=2+76;

      case '2022',   % New NMEA data block from WInRiverII

        cfg.sourceprog='WINRIVER2';
```

```
        if SOURCE~=2, fprintf('\n***** Apparently a WINRIVER file - Raw NMEA data handler not yet
implemented\n\n'); end;
        SOURCE=2;

        specID=fread(fd,1,'uint16');
        msgsiz=fread(fd,1,'int16');
        deltaT=fread(fd,8,'uchar');
        nbyte=2+12;

        fseek(fd,msgsiz,'cof');
        nbyte=nbyte+msgsiz;

   %%      fprintf(' %d ',specID);
        switch specID,
          case 100,
          case 101,
          case 102,
          case 103,
         end;


% The following blocks come from WINRIVER files, they aparently contain
% the raw NMEA data received from a serial port.
%
% Note that for WINRIVER files somewhat decoded data is also available
% tucked into the bottom track block.
%
% I've put these all into their own block because RDI's software apparently completely ignores the
% stated lengths of these blocks and they very often have to be changed. Rather than relying on the
% error coding at the end of the main block to do this (and to produce an error message) I will
% do it here, without an error message to emphasize that I am kludging the WINRIVER blocks only!

      case {'2100','2101','2102','2103','2104'}

      winrivprob=1;

      switch id,

        case '2100', % $xxDBT    (Winriver addition) 38
          cfg.sourceprog='WINRIVER';
          if SOURCE~=2, fprintf('\n***** Apparently a WINRIVER file - Raw NMEA data handler not yet
implemented\n\n'); end;
          SOURCE=2;
          str=fread(fd,38,'uchar')';
          nbyte=2+38;

        case '2101', % $xxGGA    (Winriver addition) 94 in manual but 97 seems to work
                          % Except for a winriver2 file which seems to use 77.
          cfg.sourceprog='WINRIVER';
          if SOURCE~=2, fprintf('\n***** Apparently a WINRIVER file - Raw NMEA data handler not yet
implemented\n\n'); end;
          SOURCE=2;
          str=setstr(fread(fd,97,'uchar')');
          nbyte=2+97;
          l=strfind(str,'$GPGGA');
          if ~isempty(l),

ens.smtime(k)=(sscanf(str(l+7:l+8),'%d')+(sscanf(str(l+9:l+10),'%d')+sscanf(str(l+11:l+12),'%d')/60)/60)/24;
          end;
   %      disp(['->' setstr(str(1:50)) '<-']);

        case '2102', % $xxVTG    (Winriver addition) 45 (but sometimes 46 and 48)
```

```
          cfg.sourceprog='WINRIVER';
          if SOURCE~=2, fprintf('\n***** Apparently a WINRIVER file - Raw NMEA data handler not yet
implemented\n\n'); end;
          SOURCE=2;
          str=fread(fd,45,'uchar')';
          nbyte=2+45;
  %         disp(setstr(str));

        case '2103', % $xxGSA    (Winriver addition) 60
          cfg.sourceprog='WINRIVER';
          if SOURCE~=2, fprintf('\n***** Apparently a WINRIVER file - Raw NMEA data handler not yet
implemented\n\n'); end;
          SOURCE=2;
          str=fread(fd,60,'uchar')';
  %         disp(setstr(str));
          nbyte=2+60;

        case '2104',    %xxHDT or HDG (Winriver addition) 38
          cfg.sourceprog='WINRIVER';
          if SOURCE~=2, fprintf('\n***** Apparently a WINRIVER file - Raw NMEA data handler not yet
implemented\n\n'); end;
          SOURCE=2;
          str=fread(fd,38,'uchar')';
  %         disp(setstr(str));
          nbyte=2+38;
        end;




      case '0701', % Number of good pings
        fseek(fd,4*cfg.n_cells,'cof');
        nbyte=2+4*cfg.n_cells;

      case '0702', % Sum of squared velocities
        fseek(fd,4*cfg.n_cells,'cof');
        nbyte=2+4*cfg.n_cells;

      case '0703', % Sum of velocities
        fseek(fd,4*cfg.n_cells,'cof');
        nbyte=2+4*cfg.n_cells;

% These blocks were implemented for 5-beam systems

      case '0A00', % Beam 5 velocity (not implemented)
        fseek(fd,cfg.n_cells,'cof');
        nbyte=2+cfg.n_cells;

      case '0301', % Beam 5 Number of good pings (not implemented)
        fseek(fd,cfg.n_cells,'cof');
        nbyte=2+cfg.n_cells;

      case '0302', % Beam 5 Sum of squared velocities (not implemented)
        fseek(fd,cfg.n_cells,'cof');
        nbyte=2+cfg.n_cells;

      case '0303', % Beam 5 Sum of velocities (not implemented)
        fseek(fd,cfg.n_cells,'cof');
        nbyte=2+cfg.n_cells;

      case '020C', % Ambient sound profile (not implemented)
```

```
      fseek(fd,4,'cof');
      nbyte=2+4;

    case '3000',    % Fixed attitude data format for OS-ADCPs (not implemented)
      fseek(fd,32,'cof');
      nbyte=2+32;

     otherwise,
      % This is pretty idiotic - for OS-ADCPs (phase 2) they suddenly decided to code
      % the number of bytes into the header ID word. And then they don't really
      % document what they did! So, this is cruft of a high order, and although
      % it works on the one example I have - caveat emptor....
      %
      % Anyway, there appear to be codes 0340-03FC to deal with. I am not going to
      % decode them but I am going to try to figure out how many bytes to
      % skip.
      if strcmp(id(1:2),'30'),
        % I want to count the number of 1s in the middle 4 bits of the
        % 2nd two bytes.
        nflds=sum(dec2base(bitand(hex2dec(id(3:4)),hex2dec('3C')),2)=='1');
        % I want to count the number of 1s in the highest 2 bits of byte 3
        dfac= sum(dec2base(bitand(hex2dec(id(3)),hex2dec('C')),2)=='1');
        fseek(fd,12*nflds*dfac,'cof');
        nbyte=2+12*nflds*dfac;

      else
        fprintf('Unrecognized ID code: %s\n',id);
        nbyte=2;
      end;
     %% ens=-1;
     %% return;


    end;

    % here I adjust the number of bytes so I am sure to begin
    % reading at the next valid offset. If everything is working right I shouldn't have
    % to do this but every so often firware changes result in some differences.

    %%fprintf('#bytes is %d, original offset is %d\n',nbyte,byte_offset);
    byte_offset=byte_offset+nbyte;

    if n<length(hdr.dat_offsets),
      if hdr.dat_offsets(n+1)~=byte_offset,
        if ~winrivprob, fprintf('%s: Adjust location by %d\n',id,hdr.dat_offsets(n+1)-byte_offset); end;
        fseek(fd,hdr.dat_offsets(n+1)-byte_offset,'cof');
      end;
      byte_offset=hdr.dat_offsets(n+1);
    else
      if hdr.nbyte-2~=byte_offset,
        if ~winrivprob, fprintf('%s: Adjust location by %d\n',id,hdr.nbyte-2-byte_offset); end;
        fseek(fd,hdr.nbyte-2-byte_offset,'cof');
      end;
      byte_offset=hdr.nbyte-2;
    end;
  end;

  % Now at the end of the record we have two reserved bytes, followed
  % by a two-byte checksum = 4 bytes to skip over.

  readbytes=ftell(fd)-startpos;
  offset=(hdr.nbyte+2)-byte_offset; % The 2 is for the checksum
```

```
  if offset ~=4 & FIXOFFSET==0,
    fprintf('\n*****************************************************\n');
    if feof(fd),
      fprintf(' EOF reached unexpectedly - discarding this last ensemble\n');
      ens=-1;
    else
      fprintf('Adjust location by %d (readbytes=%d, hdr.nbyte=%d)\n',offset,readbytes,hdr.nbyte);
      fprintf(' NOTE - If this appears at the beginning of the read, it is\n');
      fprintf('             is a program problem, possibly fixed by a fudge\n');
      fprintf('             PLEASE REPORT TO rich@eos.ubc.ca WITH DETAILS!!\n\n');
      fprintf('          -If this appears at the end of the file it means\n');
      fprintf('            The file is corrupted and only a partial record has    \n');
      fprintf('            has been read\n');
    end;
    fprintf('*****************************************************\n');
    FIXOFFSET=offset-4;
  end;
  fseek(fd,4+FIXOFFSET,'cof');

  % An early version of WAVESMON and PARSE contained a bug which stuck an additional two
  % bytes in these files, but they really shouldn't be there
  %if cfg.prog_ver>=16.05,
  %        fseek(fd,2,'cof');
  %end;

end;

% Blank out stuff bigger than error velocity
% big_err=abs(ens.error_vel)>.2;
big_err=0;

% Blank out invalid data
ens.east_vel(ens.east_vel==-32.768 | big_err)=NaN;
ens.north_vel(ens.north_vel==-32.768 | big_err)=NaN;
ens.vert_vel(ens.vert_vel==-32.768 | big_err)=NaN;
ens.error_vel(ens.error_vel==-32.768 | big_err)=NaN;




%--------------------------------------
function y=nmedian(x,window,dim);
% Copied from median but with handling of NaN different.

if nargin==2,
  dim = min(find(size(x)~=1));
  if isempty(dim), dim = 1; end
end

siz = [size(x) ones(1,dim-ndims(x))];
n = size(x,dim);

% Permute and reshape so that DIM becomes the row dimension of a 2-D array
perm = [dim:max(length(size(x)),dim) 1:dim-1];
x = reshape(permute(x,perm),n,prod(siz)/n);

% Sort along first dimension
x = sort(x,1);
[n1,n2]=size(x);

if n1==1,
```

```
 y=x;
else
  if n2==1,
   kk=sum(isfinite(x),1);
   if kk>0,
     x1=x(fix((kk-1)/2)+1);
     x2=x(fix(kk/2)+1);
     x(abs(x-(x1+x2)/2)>window)=NaN;
   end;
   x = sort(x,1);
   kk=sum(isfinite(x),1);
   x(isnan(x))=0;
   y=NaN;
   if kk>0,
    y=sum(x)/kk;
   end;
  else
   kk=sum(isfinite(x),1);
   ll=kk<n1-2;
   kk(ll)=0;x(:,ll)=NaN;
   x1=x(fix((kk-1)/2)+1+[0:n2-1]*n1);
   x2=x(fix(kk/2)+1+[0:n2-1]*n1);

   x(abs(x-ones(n1,1)*(x1+x2)/2)>window)=NaN;
   x = sort(x,1);
   kk=sum(isfinite(x),1);
   x(isnan(x))=0;
   y=NaN+ones(1,n2);
   if any(kk),
    y(kk>0)=sum(x(:,kk>0))./kk(kk>0);
   end;
  end;
end;

% Permute and reshape back
siz(dim) = 1;
y = ipermute(reshape(y,siz(perm)),perm);

%--------------------------------------
function y=nmean(x,dim);
% R_NMEAN Computes the mean of matrix ignoring NaN
%              values
%     R_NMEAN(X,DIM) takes the mean along the dimension DIM of X.
%

kk=isfinite(x);
x(~kk)=0;

if nargin==1,
  % Determine which dimension SUM will use
  dim = min(find(size(x)~=1));
  if isempty(dim), dim = 1; end
end;

if dim>length(size(x)),
 y=x;                    % For matlab 5.0 only!!! Later versions have a fixed 'sum'
else
  ndat=sum(kk,dim);
  indat=ndat==0;
  ndat(indat)=1; % If there are no good data then it doesn't matter what
                      % we average by - and this avoid div-by-zero warnings.
```

```
    y = sum(x,dim)./ndat;
    y(indat)=NaN;
end;


function [U,V] = mag_var(theta,u,v)
% magnetic variation correction
%
% input:
%   theta (degree)
%   u,v: horizontal velocity profile
%
%
% output:
%   U,V
%
theta_rad = deg2rad(theta);

M = [cos(theta_rad) sin(theta_rad);-sin(theta_rad) cos(theta_rad)];

tmp   = M*[u,v]';
U     = tmp(1);
V     = tmp(2);
```

Appendix B Output Accuracy

The output accuracy for the OOI L1 Velocity Profile core data product calculated by this algorithm is equivalent to the accuracy of the instrument. The manufacturer, TRDI, provides accuracy as stated below.

For fixed assets.

Instrument class	Output	Accuracy
VADCP: TRDI Workhorse Sentinal, modified (600kHz ADCP)	≤ ±500 mm/s	± 5 mm/s
	> ±500 mm/s	1%

Relevant requirements from the OOI requirements database (DOORS, L2_Science_Requirements_ReferenceOnly_Baseline_Version_2.34 exported from DOORS SL CCB 2013-01-29) are listed below.

Turbulent Water Velocity Profiles shall be measured with a minimum accuracy of 0.01 ms-1

Turbulent Water Velocity Profile measurements shall have a resolution of 0.001 ms-1.

No accuracy information is known for the OOI L1 Echo Intensity core data product.

Appendix C Sensor Calibration Effects

None.