ALGORITHM THEORETICAL BASIS DOCUMENT (ATBD):

EDDY-COVARIANCE DATA PRODUCTS BUNDLE

PREPARED BY	ORGANIZATION	DATE
Stefan Metzger	NEON	2018-04-30
David Durden	NEON	2018-04-24
Christopher Florian	NEON	2018-04-23
Hongyan Luo	NEON	2018-03-07
Natchaya Pingintha-Durden	NEON	2018-04-19
Ke Xu	University of Wisconsin	2018-04-13

APPROVALS	ORGANIZATION	APPROVAL DATE

RELEASED BY	ORGANIZATION	RELEASE DATE

See configuration management system for approval history.

The National Ecological Observatory Network is a project solely funded by the National Science Foundation and managed under cooperative agreement by Battelle. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the National Science Foundation.

Change Record

REVISION	DATE	ECO #	DESCRIPTION OF CHANGE
A	06/21/2017	ECO-05623	Initial release

TABLE OF CONTENTS

LIST OF TABLES AND FIGURES

1 DESCRIPTION

The National Ecological Observatory Network (NEON) seeks to address grand challenges in continental-scale ecology through extensive observational infrastructure across the U.S. One core element are eddy-covariance (EC) flux measurements of ecologically-relevant energy, water, and trace gas fluxes, which are performed at 47 NEON Terrestrial Instrument System (TIS) sites. The underlying NEON observatory design is based on multivariate geographic clustering (11). At each NEON TIS site, tower-based EC-flux measurements are performed in coordination with a wide range of contextual observations (Appendix B). The EC subsystems aim to maximize data coverage and quality through close design integration of tower, a suite of sensors and auxiliary components (Appendix C). The full complement of resulting EC bundled data products (DP) is directly available from the NEON Data Portal. In addition, a set of standard outputs is regularly submitted to AmeriFlux and FLUXNET for cross-network harmonization and access.

1.1 Purpose

This Algorithm Theoretical Basis Document (ATBD) accompanies NEON's EC bundled DPs. It describes the theoretical background and entire algorithmic sequence used for determining the surface-atmosphere exchange (SAE) of momentum, heat, H_2O and CO_2 from sensor readings of the wind vector, temperature, and scalar concentrations. In addition, the isotopic composition of atmospheric H_2O and CO_2 is determined. Additional measurements can be proposed (Sect. 8).

1.2 Scope

The EC system consists of two subsystems, the eddy-covariance turbulent exchange subsystem (ECTE) and the eddy-covariance storage exchange subsystem (ECSE). The command, control, and configuration (C3) documents for ECTE [AD01] and ECSE [AD02] describe the corresponding instruments, set points, control parameters, conditions/constraints, and any necessary error handling for the physical implementation of the subsystems. Some basic information is also summarized in Appendix C. Data product levels relevant for the subsequent processing of sensor readings are:

- dp00: sensor readings in engineering units; e.g. concentration as infrared absorptance.
- dp0p: pre-conditioned data in scientific units; e.g. concentration as mole fraction.
- dp01: descriptive statistics.
- dp02: time-interpolated data.
- dp03: space-interpolated data.
- dp04: flux data.

Prior to the scientific processing described in this ATBD, all dp00 are pre-conditioned to dp0p as detailed in AD[02] and AD[03]. The present document outlines the scientific rationale and process implementation for transitioning dp0p to dp01 – dp04. At the time of writing, direct links to the corresponding workflows and functions with public access to the operational code in the https://github.com/NEONScience/eddy4R GitHub repository are in preparation. This combines a concise overview in this ATBD with the full transparency of each processing step, and facilitates direct community input on the continued development and operation of NEON EC DPs (Sect. 4). In the interim while this GitHub repository is being prepared for public access, the manuals for the eddy4R.base and eddy4R.qaqc R-packages are available in Appendix G.

This ATBD first introduces related documents, acronyms and conventions in Sect. 2. Inputs and outputs are described in Sect. 3, and Sect. 4 introduces the framework for community-driven algorithm development and operation. Throughout Sects. 5–7 the processing steps specific to turbulent exchange, storage exchange and net surface-atmosphere exchange are summarized, respectively. Each section is divided into subsections specific to data analysis, quality assurance and quality control, and uncertainty analysis, and focuses on algorithm theory and its implementation. Future plans and modifications are briefly outlined in Sect. 8.

2 RELATED DOCUMENTS, ACRONYMS AND VARIABLE NOMENCLATURE

Below applicable and reference documents are available from the NEON Document library. External references are listed in Sect. 11, and acronyms and variable nomenclature are tabulated in Appendix D – Appendix G.

2.1 Applicable Documents

AD[01]	NEON.DOC.000456 Eddy-covariance turbulent exchange subsystem Command, Control and Configuration document
AD[02]	NEON.DOC.000807 NEON Algorithm Theoretical Basis Document (ATBD) – Eddy Covariance Turbulent Exchange Subsystem Level 0 to Level 0 prime
AD[03]	NEON.DOC.004967 NEON Algorithm Theoretical Basis Document (ATBD) – Eddy Covariance Storage Exchange Subsystem Level 0 to Level 0 prime
AD[04]	NEON.DOC.000573 FIU plan for airshed QA/QC development
AD[05]	NEON.DOC.001113 Quality Flags and Quality Metrics for TIS Data Products ATBD
AD[06]	NEON.DOC.011081 ATBD QA/QC plausibility testing
AD[07]	NEON.DOC.001069 Preprocessing for TIS Level 1 Data Products
AD[08]	NEON.DOC.002651 Data Product Naming Convention
AD[09]	NEON.DOC.000465 Eddy-covariance storage exchange subsystem Command, Control and Configuration document

2.2 Reference Documents

RD[01]	NEON.DOC.000008 NEON Acronym List
RD[02]	NEON.DOC.000243 NEON Glossary of Terms

3 DATA PRODUCT DESCRIPTION

3.1 Variables Reported

The eddy-covariance related DPs provided by the algorithms documented in this ATBD are listed in Table 1. The DPs are provided in the form of HDF5 files (Sect. 3.2), including a description of all file structure and objects. Each Data Product encompasses several sub-products, and the units of the individual sub-products are provided in the HDF5 file. The data products will be produced and published in three phases, the initial transition, a science reviewed quality transition, and the epoch yearly transition (see Sect. 4 for more information).

Table 1. List of variables reported. For column "Location in HDF5 file", SITE refers to the four-letter acronym of a NEON TIS site, and DQU to one of {data, qfqm, ucrt} referring to respective locations of data, quality and uncertainty information. The "*" denotes products that are not in the current NEON HDF5 files, but will be added in future data releases.

Instrument system	Description	Temporal resolution	Data Product Number	Location in HDF5 file
Re-ingested	Triple Aspirated Air Temperature (**tempAirTop**)	1 min, 30 min	NEON.DP1.00003	SITE/dp01/DQU/tempAirTop
	Single Aspirated Air Temperature (**tempAirLvl**)	1 min, 30 min	NEON.DP1.00002	SITE/dp01/DQU/tempAirLvl
	*Soil Temperature (**tempSoil**)	1 min, 30 min	NEON.DP1.00041	SITE/dp01/DQU/tempSoil
	*Soil Heat Flux (**fluxHeatSoil**)	1 min, 30 min	NEON.DP1.00040	SITE/dp01/DQU/fluxHeatSoil
	*Shortwave and Longwave Radiation (**radiNet**)	1 min, 30 min	NEON.DP1.00023	SITE/dp01/DQU/radiNet
	*Soil water content and water salinity (**h2oSoilVol**)	1 min, 30 min	NEON.DP1.00094	SITE/dp01/DQU/h2oSoilVol
	*Barometric pressure (**presBaro**)	1 min, 30 min	NEON.DP1.00004	SITE/dp01/DQU/presBaro
EC turbulent exchange (ECTE)	3D Wind Speed, Direction and Sonic Temperature (**soni**)	1 min, 30 min	NEON.DP1.00007	SITE/dp01/DQU/soni
	3D Wind Attitude and Motion Reference (**amrs**)	1 min, 30 min	NEON.DP1.00010	SITE/dp01/DQU/amrs
	CO_2 Concentration – Turbulent (**co2Turb**)	1 min, 30 min	NEON.DP1.00034	SITE/dp01/DQU/co2Turb
	H_2O Concentration – Turbulent (**h2oTurb**)	1 min, 30 min	NEON.DP1.00035	SITE/dp01/DQU/h2oTurb
EC storage exchange (ECSE)	CO_2 Concentration – Storage (**co2Stor**)	2 min, 30 min	NEON.DP1.00099	SITE/dp01/DQU/co2Stor
	H_2O Concentration – Storage (**h2oStor**)	2 min, 30 min	NEON.DP1.00100	SITE/dp01/DQU/h2oStor
	Atmospheric CO_2 Isotopes (**isoCo2**)	9 min, 30 min	NEON.DP1.00036	SITE/dp01/DQU/isoCo2
	Atmospheric H_2O isotopes (**isoH2o**)	9 min, 30 min	NEON.DP1.00037	SITE/dp01/DQU/isoH2o

Instrument system	Description	Temporal resolution	Data Product Number	Location in HDF5 file
EC storage exchange (ECSE)	Temperature rate of change (**tempStor**)	30 min	NEON.DP2.00024	SITE/dp02/DQU/tempStor
	CO_2 concentration rate of change (**dp02 co2Stor**)	30 min	NEON.DP2.00008	SITE/dp02/DQU/co2Stor
	H_2O concentration rate of change (**dp02 h2oStor**)	30 min	NEON.DP2.00009	SITE/dp02/DQU/h2oStor
EC storage exchange (ECSE)	Temperature rate of change profile (**dp03 tempStor**)	30 min	NEON.DP3.00008	SITE/dp03/DQU/tempStor
	CO_2 concentration rate of change profile (**dp03 co2Stor**)	30 min	NEON.DP3.00009	SITE/dp03/DQU/co2Stor
	H_2O concentration rate of change profile (**dp03 h2oStor**)	30 min	NEON.DP3.00010	SITE/dp03/DQU/h2oStor
EC profile + turbulence combined (NSAE)	Sensible heat flux (**fluxHeat**)	30 min	NEON.DP4.00002	SITE/dp04/DQU/fluxHeat
	Momentum Flux (**fluxMome**)	30 min	NEON.DP4.00007	SITE/dp04/DQU/fluxMome
	Latent heat flux (**fluxH2o**)	30 min	NEON.DP4.00137	SITE/dp04/DQU/fluxH2o
	Carbon dioxide flux (**fluxCo2**)	30 min	NEON.DP4.00067	SITE/dp04/DQU/fluxCo2
	Footprint characteristics (**foot**)	30 min	NEON.DP4.00201	SITE/dp04/DQU/foot

3.2 HDF5 Representation

The DPs listed in Table 1 can be downloaded as HDF5 data product bundle from the NEON Data Portal: the Hierarchical Data Format (HDF), currently distributed as HDF5, provides a file format with high compressibility, fast efficient reading and writing capabilities, directory-style files, and metadata attachment. The contents of HDF5 files can be explored intuitively e.g. with tools freely available from the HDF Group such as HDFView, and is supported by all major programming languages. The HDF5 file format allows to package various data sets into a single file with built-in structure for managing both data and metadata. In addition, the NEON processing pipeline utilizes HDF5 files for input/output operations.

The specific locations of the individual EC DPs in the HDF5 file is provided in in Table 1, column "Location in HDF5 file". The HDF5 file itself contains a description of all terms used for naming objects ("objDesc"), and a readMe with examples and additional information. Notably, missing values in the HDF5 file are expressed as NaN for data and NA for metadata.

The underlying HDF5 file structure was developed following the NEON data product naming convention provided in AD[08], where portions of the naming convention were selected to develop the hierarchical structure of the HDF5 file as described below and illustrated in Figure 1:

NEON.DOM.SITE.DPL.PRNUM.REV.TERMS.HOR.VER.TMI, with:

NEON=NEON

DOM=DOMAIN, e.g. D10

SITE=SITE, e.g. STER

DPL=DATA PRODUCT LEVEL, e.g. DP1

PRNUM = PRODUCT NUMBER =>5 digit number. Set in data products catalog. TIS = 00000-09999

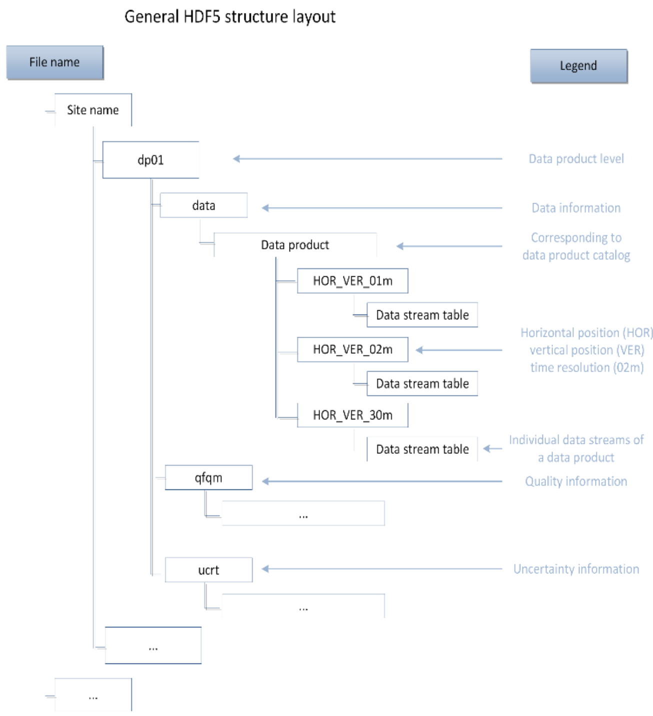
REV = REVISION, e.g. 001.

TERMS=From NEON's controlled list of terms. Index is unique across products.

HOR = HORIZONTAL INDEX. Semi-controlled. Examples: Tower=000, HUT=700.

VER = VERTICAL INDEX. Semi-controlled. Examples: Ground level=000, second tower level=020.

TMI=TEMPORAL INDEX. Examples: 001=1 minute, 030=30 minute, 999=irregular intervals.



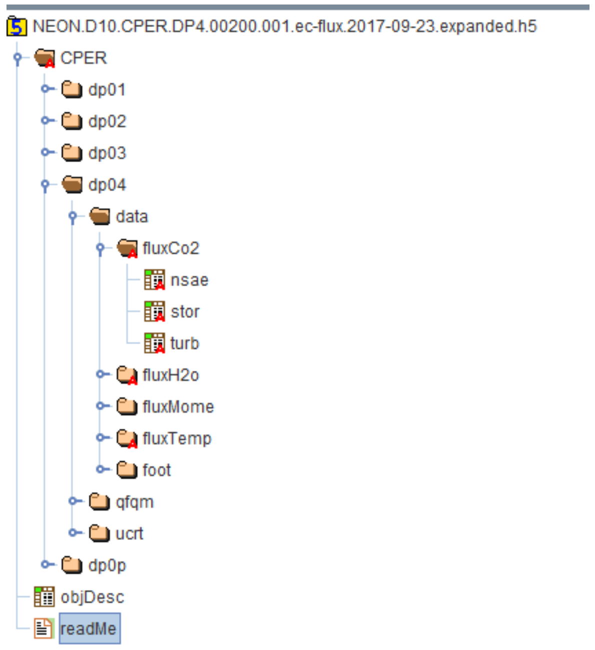


Figure 1. Theoretical diagram depicting the NEON HDF5 file structure following the NEON DP naming convention (left). An actual NEON HDF5 file screenshot depicting the hierarchical layout of the files (right). Note that PRNUM is replaced by the data product name associated with that PRNUM, e.g., **fluxCO2**.

One departure from the DP naming convention is the use of the data product name (e.g. **irgaTurb**) in place of PRNUM in the Data Product ID group level, this change was made to improve readability. At the top level of the provided HDF5 file a readme and object description (objDesc) are provided to explain the contents of the file. An additional HDF5 group level was added to separate the data (data), quality flags and quality metrics (qfqm), and uncertainty quantification (ucrt). It should also be noted that level 3 and 4 data products (dp03 and dp04) are spatially interpolated and only provided at 30 minute aggregation periods; thus, the HOR_VER_TMI level is not present (see example in Figure 1).

3.3 Input Dependencies

Below Table 2 -Table 5, and Table 12 detail the Eddy-covariance turbulent exchange (ECTE) related dp0p DPs used to produce dp01 DPs in this ATBD.

Table 2. List of eddy-covariance turbulent exchange ultrasonic anemometer/thermometer (**soni**)-related dp0p DPs that are ingested in this ATBD.

dp0p Description	**dp0p Term Name**	**Sample Frequency**	**Units**
Measured along-axis wind speed (u_m)	veloXaxs	20 Hz	ms^{-1}
Measured cross-axis wind speed (v_m)	veloYaxs	20 Hz	ms^{-1}
Measured vertical-axis wind speed (w_m)	veloZaxs	20 Hz	ms^{-1}
Measured speed of sound (c_m)	veloSoni	20 Hz	ms^{-1}
Sonic temperature (T_{SONIC})	tempSoni	20 Hz	K
Sample count	idx	20 Hz	NA
Sensor error flag ($QF_{SONIC,o1}$: Sensor unresponsive)	qfSoniUnrs	20 Hz	NA
Sensor error flag ($QF_{SONIC,o2}$: No data available)	qfSoniData	20 Hz	NA
Sensor error flag ($QF_{SONIC,o3}$: Sensor trigger source lost)	qfSoniTrig	20 Hz	NA
Sensor error flag ($QF_{SONIC,o4}$: SDM communications error)	qfSoniComm	20 Hz	NA
Sensor error flag ($QF_{SONIC,o5}$: Wrong embedded sensor code)	qfSoniCode	20 Hz	NA
Sensor signal flag ($QF_{SONIC,s1}$: Axes T_{SONIC} difference > 4 K)	qfSoniTemp	20 Hz	NA
Sensor signal flag ($QF_{SONIC,s2}$: Poor signal lock)	qfSoniSgnlPoor	20 Hz	NA
Sensor signal flag ($QF_{SONIC,s3}$: High signal amplitude)	qfSoniSgnlHigh	20 Hz	NA
Sensor signal flag ($QF_{SONIC,s4}$: Low signal amplitude)	qfSoniSgnlLow	20 Hz	NA

Table 3. List of eddy-covariance turbulent exchange attitude and motion reference (**amrs**)-related dp0p DPs that are ingested in this ATBD.

dp0p DP	**dp0p Term Name**	**Sample Frequency**	**Units**
Measured along-axis acceleration ($acc_{x,m}$)	accXaxs	40 Hz	$m\ s^{-2}$
Measured cross-axis acceleration ($acc_{y,m}$)	accYaxs	40 Hz	$m\ s^{-2}$

dp0p DP	dp0p Term Name	Sample Frequency	Units
Measured vertical-axis acceleration ($acc_{z,m}$)	accZaxs	40 Hz	m s^{-2}
Along-axis free acceleration	accXaxsDiff	40 Hz	m s^{-2}, positive forward
Cross-axis free acceleration	accYaxsDiff	40 Hz	m s^{-2}, positive left
Vertical-axis free acceleration	accZaxsDiff	40 Hz	m s^{-2}, positive up
Pitch rate	avelYaxs	40 Hz	rad s^{-1}
Roll rate	avelXaxs	40 Hz	rad s^{-1}
Yaw rate	avelZaxs	40 Hz	rad s^{-1}
Measured pitch angle (θ_{m})	angYaxs	40 Hz	rad
Measured roll angle (ϕ_{m})	angXaxs	40 Hz	rad
Yaw angle (ψ)	angZaxs	40 Hz	rad
Index value	idx	40 Hz	NA
Sensor signal flag: Selftest	qfAmrsVal	40 Hz	NA
Sensor signal flag: Filter Valid	qfAmrsFilt	40 Hz	NA
Sensor signal flag: NoVelocityUpdate status	qfAmrsVelo	40 Hz	NA
Sensor signal flag: Clipping indication	qfAmrsRng	40 Hz	NA

Table 4. List of eddy-covariance turbulent exchange infrared gas analyzer (irgaTurb)-related dp0p DPs that are ingested in this ATBD.

dp0p DP	dp0p Term Name	Sample Frequency	Units
Cell temperature in (at sensor head inlet)	tempIn	20 Hz	K
Cell temperature out (at sensor head inlet)	tempOut	20 Hz	K
Cell temperature (weighted average of head inlet and outlet temperature)	tempMean	20 Hz	K
Block temperature	tempRefe	20 Hz	K
Ambient pressure (LI–7550 box pressure)	presAtm	20 Hz	Pa
Head pressure (differential pressure head-box)	presDiff	20 Hz	Pa
Total pressure (LI–7550 box pressure + head pressure)	presSum	20 Hz	Pa
H_2O sample power	powrH2oSamp	20 Hz	W
H_2O reference power	powrH2oRefe	20 Hz	W
H_2O raw absorptance	asrpH2o	20 Hz	-
H_2O molar density	densMoleH2o	20 Hz	mol m^{-3}
H_2O dry mole fraction	rtioMoleDryH2o	20 Hz	mol mol^{-1}
CO_2 sample power	powrCo2Samp	20 Hz	W
CO_2 reference power	powrCo2Refe	20 Hz	W
CO_2 raw absorptance	asrpCo2	20 Hz	-
CO_2 molar density	densMoleCo2	20 Hz	mol m^{-3}

dp0p DP	dp0p Term Name	Sample Frequency	Units
CO_2 dry mole fraction	rtioMoleDryCo2	20 Hz	mol mol^{-1}
Sequence number	idx	20 Hz	NA
LI-7200 diagnostic value 2 (sync clocks)	diag02	20 Hz	NA
LI-7200 cooler voltage	potCool	20 Hz	V
CO_2 signal strength	ssiCo2	20 Hz	-
H_2O signal strength	ssiH2o	20 Hz	-
Sensor flag (f_{L01}: Head detect)	qfIrgaHead	20 Hz	NA
Sensor flag (f_{L02}: Outlet temperature)	qfIrgaTempOut	20 Hz	NA
Sensor flag (f_{L03}: Inlet temperature)	qfIrgaTempIn	20 Hz	NA
Sensor flag (f_{L04}: Aux input)	qfIrgaAux	20 Hz	NA
Sensor flag (f_{L05}: Differential pressure)	qfIrgaPres	20 Hz	NA
Sensor flag (f_{L06}: Chopper)	qfIrgaChop	20 Hz	NA
Sensor flag (f_{L07}: Detector)	qfIrgaDetc	20 Hz	NA
Sensor flag (f_{L08}: PLL)	qfIrgaPll	20 Hz	NA
Sensor flag (f_{L09}: Sync)	qfIrgaSync	20 Hz	NA
Sensor flag (f_{L10}: AGC)	qfIrgaAgc	20 Hz	-

Table 5. List of eddy-covariance turbulent exchange infrared gas analyzer sampling mass flow controller (mfcSampTurb)-related dp0p DPs that are ingested in this ATBD.

dp0p DP	dp0p Term Name	Sample Frequency	Units
Sampling mass flow rate set point	frtSet00	20 Hz	m^3 s^{-1}
Sampling mass flow rate	frt00	20 Hz	m^3 s^{-1}
Sampling volumetric flow rate	frt	20 Hz	m^3 s^{-1}
Sampling gas pressure	presAtm	20 Hz	Pa
Sampling gas temperature	temp	20 Hz	K

Below Table 6 – Table 10 detail the eddy-covariance storage exchange (ECSE) related dp0p DPs used to produce dp01 DPs in this ATBD.

Table 6. List of eddy-covariance storage exchange carbon dioxide (**co2Stor**)-related dp0p DPs that are ingested in this ATBD.

dp0p DP	dp0p Term Name	Sample Frequency	Units
Sampling mass flow rate	frt00	1 Hz	m^3 s^{-1}
Sampling gas pressure	pres	1 Hz	Pa
CO_2 dry mole fraction	rtioMoleDryCo2	1 Hz	mol mol^{-1}
CO_2 wet mole fraction	rtioMoleWetCo2	1 Hz	mol mol^{-1}
Sampling gas temperature	temp	1 Hz	K

Table 7. List of eddy-covariance storage exchange water vapor (**h2oStor**)-related dp0p DPs that are ingested in this ATBD.

dp0p DP	**dp0p Term Name**	**Sample Frequency**	**Units**
Sampling mass flow rate	frt00	1 Hz	$m^3\ s^{-1}$
Sampling gas pressure	pres	1 Hz	Pa
H_2O dry mole fraction	rtioMoleDryH2o	1 Hz	$mol\ mol^{-1}$
H_2O wet mole fraction	rtioMoleWetH2o	1 Hz	$mol\ mol^{-1}$
Sampling gas temperature	temp	1 Hz	K

Table 8. List of eddy-covariance storage exchange carbon dioxide isotope (**isoCo2**)-related dp0p DPs that are ingested in this ATBD.

dp0p DP	**dp0p Term Name**	**Sample Frequency**	**Units**
Ratio of stable isotopes ^{13}C to ^{12}C in CO_2	dlta13CCo2	1 Hz	‰
Gas spectrum ID	idGas	1 Hz	NA
Pressure	pres	1 Hz	Pa
$^{12}CO_2$ in CO_2 dry mole fraction	rtioMoleDry12CCo2	1 Hz	$mol\ mol^{-1}$
$^{13}CO_2$ in CO_2 dry mole fraction	rtioMoleDry13CCo2	1 Hz	$mol\ mol^{-1}$
CO_2 dry mole fraction	rtioMoleDryCo2	1 Hz	$mol\ mol^{-1}$
H_2O dry mole fraction	rtioMoleDryH2o	1 Hz	$mol\ mol^{-1}$
$^{12}CO_2$ in CO_2 wet mole fraction	rtioMoleWet12CCo2	1 Hz	$mol\ mol^{-1}$
$^{13}CO_2$ in CO_2 wet mole fraction	rtioMoleWet13CCo2	1 Hz	$mol\ mol^{-1}$
CO_2 wet mole fraction	rtioMoleWetCo2	1 Hz	$mol\ mol^{-1}$
H_2O wet mole fraction	rtioMoleWetH2o	1 Hz	$mol\ mol^{-1}$
Instrument status	sensStus	1 Hz	NA
Temperature (temp)	temp	1 Hz	K
Temperature (temp) measured at PICARRO warm box	tempWbox	1 Hz	K

Table 9. List of eddy-covariance storage exchange water vapor isotope (**isoH2o**)-related dp0p DPs that are ingested in this ATBD.

dp0p DP	**dp0p Term Name**	**Sample Frequency**	**Units**
Ratio of stable isotopes 18O:16O in H_2O	dlta18OH2o	1 Hz	‰
Ratio of stable isotopes 2H:1H in H_2O	dlta2HH2o	1 Hz	‰
Pressure	pres	1 Hz	Pa
H_2O dry mole fraction	rtioMoleDryH2o	1 Hz	$mol\ mol^{-1}$
H_2O (wet mole fraction)	rtioMoleWetH2o	1 Hz	$mol\ mol^{-1}$
Instrument Status	sensStus	1 Hz	NA
Signal to indicate if the instrument is processing the data for N2 gas or background air. 0=air mode, 1=N2 mode	stusN2	1 Hz	NA
Temperature (temp)	temp	1 Hz	K

dp0p DP	**dp0p Term Name**	**Sample Frequency**	**Units**
Temperature (temp) measured at PICARRO warm box	tempWbox	1 Hz	K
State of external solenoid valves if attached to PICARRO L2130-i	valvCrdH2o	1 Hz	NA

Table 10. List of eddy-covariance storage exchange sampling mass flow controller (**mfcSampStor**)-related dp0p DPs that are ingested in this ATBD.

dp0p DP	**dp0p Term Name**	**Sample Frequency**	**Units**
Sampling mass flow rate set point	frtSet00	1 Hz	m3 s-1
Sampling mass flow rate	frt00	1 Hz	m3 s-1
Sampling volumetric flow rate	frt	1 Hz	m3 s-1
Sampling gas pressure	presAtm	1 Hz	Pa
Sampling gas temperature	temp	1 Hz	K

In addition, standard NEON TIS sensor plausibility tests are applied at dp00 temporal resolution to the dp0p DPs listed above. The corresponding pass/fail flags per Table 11 are generated for each test according to AD[06]. (Note. We will not be carrying out the "gap test" or "null test" since the regularization is being applied according to AD[07]).

Table 11. Plausibility quality flags to be applied to all dp0p DPs

Flag	**Term modifier**	**Description**
QF_{Cal}	qfCal	Quality flag for the Invalid Calibration test
QF_{Pers}	qfPers	Quality flag for the Persistence test
QF_{Rng}	qfRng	Quality flag for the Range test
QF_{Step}	qfStep	Quality flag for the Step test

The flags are applied to all dp0p DP following a uniform naming convention, whereby the dp0p DP term name is augmented with the plausibility test flag term modifier. For example, the quality flag for the step test for measured along-axis wind speed will be "qfStepVeloXaxs".

Solenoid flags that indicate ECTE validation periods are outlined in Table 12. These data do not have quality flag information associated with the measurements.

Table 12. List of ECTE IRGA solenoid for validation gas system in NEMA enclosure (valvValiNemaTurb) - related dp0p DPs that are ingested in this ATBD

dp0p DP	**dp0p Term Name**	**Sample Frequency**	**Units**
Validation gas 1-5 status NEMA enclosure	qfGas01 – qfGas05	0.2 Hz	NA

3.4 Product Instances

Each NEON Core site with terrestrial infrastructure will produce an instance of the reported variables in Table 1. Each NEON relocatable site will produce an identical instance of the reported variables in Table 1 except for Atmospheric H_2O isotopes.

3.5 Temporal Resolution and Extent

The temporal resolution/extent of all reported variables in ECTE instrument system under Table 1 is 1 min and 30 min. The temporal resolution of most input variables in ECTE instrument system in Table 2 – Table 5 is 0.05 s (20 Hz) (design described in AD[01]), with exception of the mean **soniAmrs** variables collected at measurement frequency of 0.025 s (40 Hz). The temporal extent of all input variables is 0.5 h, i.e. a data set of 0.5 h duration shall be considered for each implementation of the presented algorithms.

The temporal resolution/extent of dp01 CO_2 and H_2O concentration data products under Table 1 is 2 min (or 9 min) and 30 min for ECSE instrument system (dp01, sect. 6.2.1.2). The temporal resolution of all input variables in

Table 6–Table 7 is 1 s (1 Hz). The temporal extent of all input variables is 1 day, i.e. a data set of 1 day duration shall be considered for each implementation of the presented algorithms.

3.6 Spatial Resolution and Extent

The input variables for ECTE used in this ATBD are measured at a single position in space at the tower top. Consequently both, input variables and reported variables are not spatially resolved. The 3D boom accelerations (acc_x, acc_y, acc_z) are point measurements. The spatial extent (path length) of all remaining variables is ≈10 cm (AD[01]). The spatial representativeness of the means, variances and covariances reported in this ATBD is a function of several factors such as measurement height $d_{z,m}$, displacement height $d_{z,d}$, wind speed and direction, atmospheric stability and surface roughness. From dispersion modeling (e.g., Schmid, 1994; Vesala et al., 2008) it is found that $\approx 10\ (d_{z,m} - d_{z,d}) < d_{x,FP90} < 100\ (d_{z,m} - d_{z,d})$, where $d_{x,FP90}$ is the cross-wind integrated upwind extent from within which 90% of a measured flux value is sourced. The spatial representativeness for each observation of the reported variables will be quantified during the implementation of AD[04].

ECSE analyzers (CO_2 and H_2O gas analyzer and isotopic CO_2 and H_2O analyzers) are located inside the instrument hut at the bottom of the tower. However, the analyzer's measurements reflect the points in space where the gas sample inlets are located on the tower infrastructure at different vertical levels, which will be site-specific. The array of aspirated air temperature measurements is made at the same vertical heights as above gas sample inlets. The vertical profile measurements of the air temperature, CO_2 and H_2O concentration will be integrated into higher-level derived data products of time rate of change (dp02, sect. 6.2.1.3), vertical resolved time rate of change (dp03, sect. 6.2.1.3) and storage flux (dp04, sect. 0).

4 OVERALL ALGORITHMIC IMPLEMENTATION

NEON utilizes the eddy4R-Docker EC data processing environment (Metzger et al., 2017) to routinely perform the calculations outlined in the following sections. eddy4R-Docker relies on the eddy4R family of open-source packages for EC raw data processing, analyses and modeling in the R Language for Statistical Computing (R Core Team, 2016), wrapped into a Docker filesystem that contains only the minimal context needed to run. The eddy4R-Docker EC data processing environment is publicly available and extensible, and continuously solicits community input through a Development and Systems Operations (DevOps) approach (Figure 2). At the time of writing, the repository and a detailed Wiki are being prepared for public access.

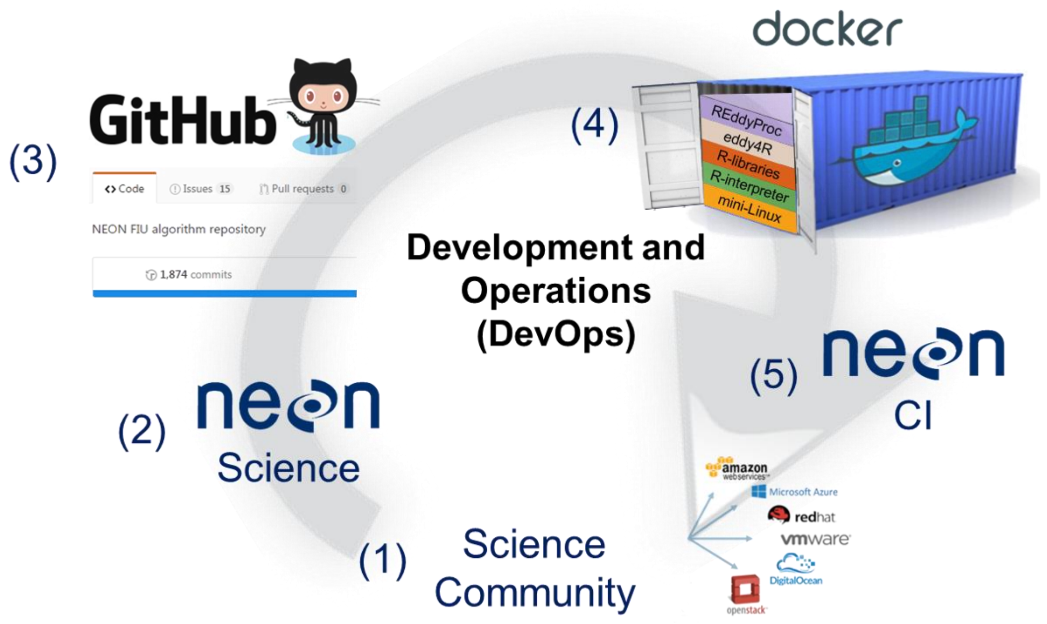


Figure 2. NEON's DevOps framework consists of a periodic sequence: The science community contributes algorithms and best practices (1) which together with NEON Science (2) are compiled into eddy4R packages via the GitHub distributed version control system (3). NEON Science releases an eddy4R version from GitHub, which automatically builds an eddy4R-Docker image on DockerHub as specified in a "Dockerfile" (4). The eddy4R-Docker image is immediately available for deployment by NEON Cyberinfrastructure (CI; 5), the Science Community (1) and NEON Science (2) alike. This DevOps cycle can be repeated for continuous development and integration of requests and future methodological improvements, resulting in the next release.

To then perform a defined series of processing steps, the eddy4R-Docker image is called with an instruction set, resulting in a running instance called Docker container (Figure 3). Through this mechanism, an arbitrary number of eddy4R-Docker containers can be run simultaneously performing identical or different services depending on the workflow file. This provides an ideal framework for scaled deployment using e.g. high-throughput compute architectures, cloud-based services etc.

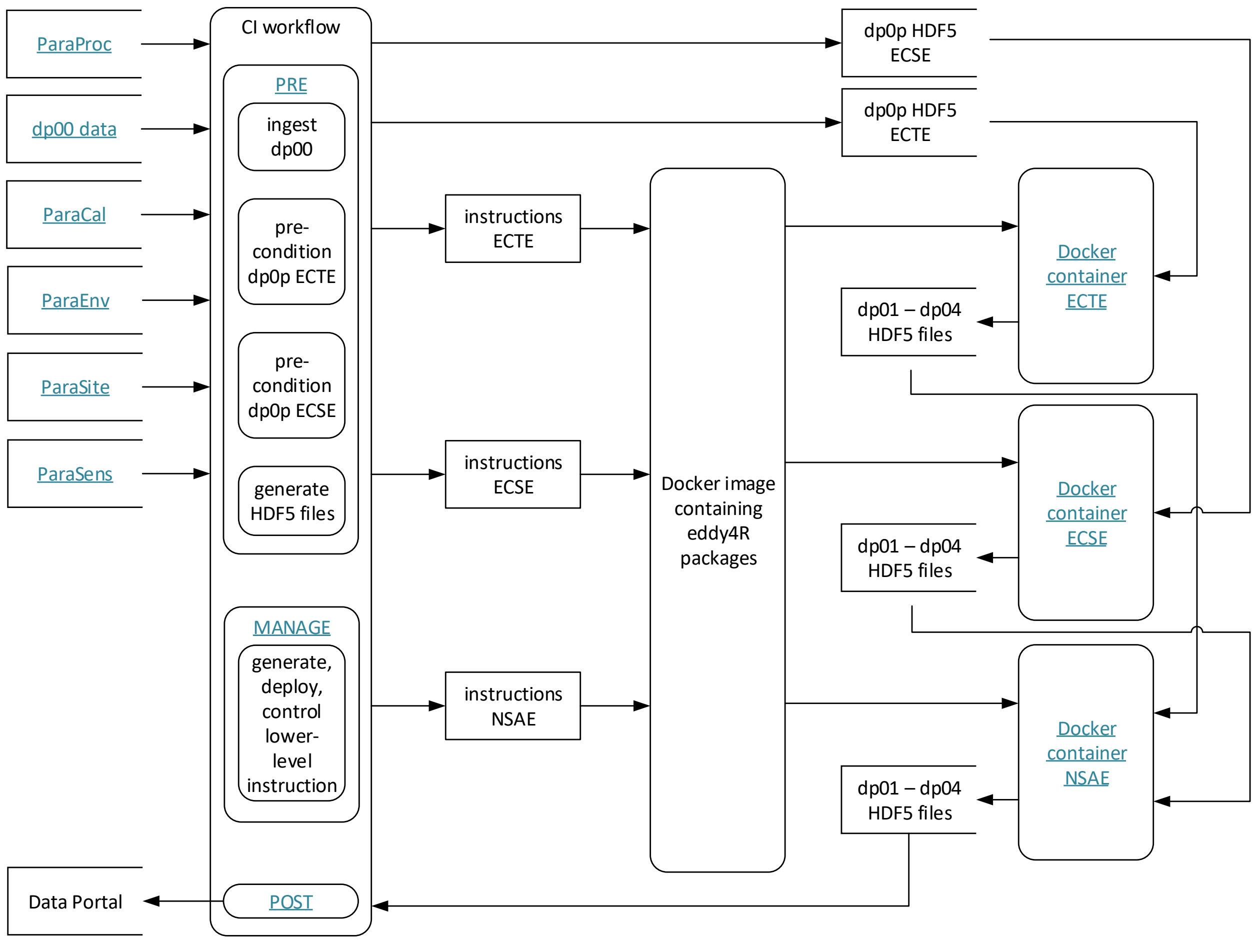


Figure 3. NEON's eddy4R-Docker EC processing framework. Individual components are described in the text.

The overall processing framework begins with ingesting information from various data sources on a site-by-site basis (Figure 3 top left). This includes EC raw data (Level 0, or dp00 data) alongside contextual information on measurement site (ParaSite), environment (ParaEnv), sensor (ParaSens), calibration (ParaCal), as well as processing parameters (ParaProc). Next, the raw data is preconditioned and all information is hierarchically combined into a compact and easily transferable HDF5 file (Figure 3 panel "CI workflow"). Each file contains the calibrated raw data (dp0p) and metadata for one site and one day, either for EC turbulent exchange or storage exchange. Together with the corresponding turbulence (ECTE), storage (ECSE) or net surface-atmosphere exchange (NSAE) instruction sets the HDF5 dp0p data file is passed to the eddy4R-Docker image, where a running Docker container is spawned that performs the specified computations (Figure 3 top right). The resulting higher-level data products (Level 1 – Level 4, or dp01-dp04) are collected and, together with all contextual information, are combined into a daily dp01-dp04 HDF5 data file that is served on the data portal (Figure 3 bottom left). In addition to the daily output files, monthly concatenated files are also available for download from the NEON data portal.

At the time of writing, the processing described in this ATBD is actively being rolled out across NEON sites, resulting in varying site and temporal coverage for download from the NEON Data Portal. During subsequent nominal operations, we plan to produce and publish the data products in three phases, to accommodate a variety of use cases: the initial near-real-time transition, a science reviewed quality transition, and the epoch yearly transition. The initial near-real-time transition is scheduled to process daily files at a 5-day delay after data collection to accommodate a 9-day centered planar-fit window (see Sect. 5.2.1.2 for details). If the data has not been received from the field it will attempt to process daily for 30 days, and if not all data is available after this window a force execution is performed populating a HDF5 file with metadata and filling data with NaN's. The monthly file will be produced after all daily files are available, no later than 30 days after the last daily file was initially attempted to be processed. An example of this transition schedule is outlined in Table 13.

Table 13. Initial data processing transition schedule example.

Date (data recorded at site)	**2017-09-01**	**2017-09-30**	**comments**
Date (first daily processing attempt)	2017-09-06	2017-10-05	Processing needs to accommodate 5 days delay (for 9 day planar-fit window)
Date (last daily processing attempt)	2017-10-06	2017-11-04	Try for up to 30 days
Date (daily processing force execution)	2017-10-07	2017-11-05	After last daily processing attempt (this example: 1 day after), process eddy4R with dp0p file that is populated with metadata + data (NaNs)
Date (first monthly processing attempt)	-	2017-10-06	First opportunity for monthly processing to succeed (provided all daily file processing attempts succeeded)
Date (last monthly processing attempt)	-	2017-11-05	Try for up to 30 days
Date (monthly processing force execution)	-	2017-11-06	After last monthly processing attempt (this example: 1 day after)

After the initial transition, the NEON science team has a one month window to manually flag data that were identified as suspect through field-based problem tracking and resolution tickets or through additional manual data quality analysis. Then, the science-reviewed transition will occur, and the data will be republished to the data portal. The last transition type is part of the yearly epoch versioning, which provides a fully quality assured and quality controlled version of the data using the latest full release of the processing code. This transition is scheduled to occur 18 months after the initial data collection, which is to provide sufficient time for all sensors to be re-calibrated in the calibration and validation laboratory (CALVAL) to determine and apply drift corrections.

5 TURBULENT EXCHANGE

The calculation of eddy-covariance momentum, heat, water vapor and carbon dioxide fluxes provides higher-level DPs with ecological relevance. These DPs have many applications within ecology and atmospheric science, and play a crucial role in constraining, calibrating and validating process-based models (e.g., Rastetter et al., 2010). This shall enable the detection of continental scale ecological change and the forecasting of its impacts.

5.1 Theory of Measurement

The exchange of momentum, heat, water vapor, CO_2 and other scalars between the earth's surface and the atmosphere is mainly governed by turbulent transport. Buoyancy as well as shear stress result in a turbulent wind field for most of the day (e.g., Stull, 1988). The eddy-covariance (EC) technique measures the properties of the turbulent wind field directly. This makes it the least invasive method currently available for direct and continuous observations of the surface-air exchange. The technique is based on the concept of mass conservation and makes use of the Reynolds decomposition (isolation of mean and fluctuating part) of relevant terms in the Navier-Stokes equation (e.g., Foken, 2008; Stull, 1988). With several restrictions (AD[04]) the net flux *F* into or out of an ecosystem can be expressed as (e.g., Loescher et al., 2006);

$$F = \underbrace{\int_0^{d_{z,m}} \frac{\partial \overline{X}}{\partial t} dz}_{\text{I}} + \underbrace{\int_0^{d_{z,m}} \frac{\partial \overline{u'X'}}{\partial x} dz}_{\text{II}} + \underbrace{\int_0^{d_{z,m}} \frac{\partial \overline{v'X'}}{\partial y} dz}_{\text{III}} + \underbrace{\int_0^{d_{z,m}} \frac{\partial \overline{w'X'}}{\partial z} dz}_{\text{IV}} \tag{1}$$

$$+ \underbrace{\int_0^{d_{z,m}} \frac{\partial \overline{u}\overline{X}}{\partial x} dz}_{\text{V}} + \underbrace{\int_0^{d_{z,m}} \frac{\partial \overline{v}\overline{X}}{\partial y} dz}_{\text{VI}} + \underbrace{\int_0^{d_{z,m}} \frac{\partial \overline{w}\overline{X}}{\partial z} dz}_{\text{VII}},$$

with overbars denoting means, and primes denoting deviations from the mean. Here, *X* is a scalar quantity such as H_2O or CO_2 mixing ratios; *u*, *v* and *w* are along-, cross-, and vertical wind speeds with respect to the Cartesian coordinates *x*, *y*, and *z*; *t* is time, and $d_{z,m}$ is the measurement height. Term I in Eq. (1) represents the positive or negative rate of change of X in the vertical column below the sensor, equivalent to storage. Terms II–IV represent the turbulent flux divergence, and terms V–VII represent advection through the layer between the surface and sensor. If the conditions at the measurement site fulfill several assumptions (details provided in AD[04]), terms I–III and V–VII cancel from Eq. (1), and term IV can be further simplified to;

$$F = \overline{w'X'}. \tag{2}$$

That is, in this case the net flux into or out of an ecosystem can be expressed as the covariance between the vertical wind and the scalar, which can be computed from ECTE measurements alone. Whether or not this reduction of Eq. (1) is valid is assessed in a series of tests during the complete implementation of AD[04]. Wherever possible, auxiliary measurements will be used to re-substitute non-negligible terms in Eq. (2), e.g. the storage term I (Sect. 6).

5.2 Data Analysis

5.2.1 Theory of Algorithm

The subject of this ATBD is the mathematical derivation of statistical quantities in Eq. (1). These quantities are used to (i) express the net flux according to Eq. (2), and (ii) quantify the fulfillment of assumptions on the site conditions during the implementation of AD[05].

5.2.1.1 De-spiking

The time series signal despiking algorithm by (Brock, 1986) is used, including the additional threshold by (Starkenburg et al., 2016). This study concluded that the median filter approach resulted in robust despiking results with little to no misclassification of spikes.

5.2.1.2 Planar-fit coordinate rotation

After careful consideration of several options, the planar-fit coordinate rotation method is used to align the vector basis of the mass conservation Eq. (1) with the average streamlines over a synoptic timescale (default: 9 days). Table 14 provides an overview of the advantages and disadvantages of three principal coordinate rotation methods. The main advantages of the double rotation method (Kaimal and Finnigan, 1994; McMillen, 1988; Tanner and Thurtell, 1969) are its applicability for online flux computation, and its robustness against alignment changes of the sonic anemometer. However, the method also suffers from several disadvantages which are overcome by the planar-fit (Kondo and Sate, 1982; Lee et al., 2004; Mahrt et al., 1996; Wilczak et al., 2001) and surface-fit (Baldocchi et al., 2000; Finnigan, 1999; Lee, 1998; Paw U et al., 2000) methods. In particular, instrument offsets, low wind periods or transient mean vertical flows $\overline{w} \neq 0$ can result in over-rotation. E.g., 0.05 m s^{-1} mean vertical flow at 2 m s^{-1} mean horizontal flow results in 1.5° over-rotation. Errors >10% per 1° over-rotation and ≤5% per 2° over-rotation have been reported for measurements of shear stress (Wilczak et al., 2001) and scalar flux (Lee et al., 2004), respectively. These errors are not distributed randomly, but a function of the 3-D flow pattern at the measurement site. Over complex terrain with diurnal flow patterns, resulting biases of the daily flux integrals in the order of 5% have been observed (Turnipseed et al., 2003).

Table 14. Properties of three coordinate rotation methods for aligning the vector basis of the mass conservation equation with the mean streamlines. Advantages of individual methods are highlighted with underline.

Property	**Double rotation**	**Planar fit**	**Surface fit**
References	Kaimal and Finnigan (1994); McMillen (1988); Tanner and Thurtell (1969)	Kondo and Sate (1982); Lee et al. (2004); Mahrt et al. (1996); Wilczak et al. (2001)	Baldocchi et al. (2000); Finnigan (1999); Lee (1998); Paw U et al. (2000)
Vector basis	Average streamline	Aerodynamic plane	Aerodynamic surface
Data basis	Individual averaging period	Ensemble of averaging periods	Ensemble of averaging periods
Computation	<u>Real-time</u>	Delayed	Delayed
Change in anemometer alignment	<u>Automatic adaptation</u>	New set of rotation angles required	New set of rotation angles required
Over-rotation	Problematic	<u>Eliminated for simple slopes</u>	<u>Eliminated for complex terrain</u>
Information on vertical advection	No	<u>For simple slopes</u>	<u>For complex terrain</u>
High-pass filtering and cross-axis folding	Yes	<u>No</u>	<u>No</u>
Consistent vector basis for flux QA/QC	No	<u>Yes</u>	<u>Yes</u>
Tracking instrument tilt	No	<u>Yes</u>	<u>Yes</u>
Uncertainty propagation	No	<u>Yes</u>	<u>Yes</u>

In contrast, the planar-fit and surface-fit methods eliminate over-rotation by identifying and distinguishing (i) an ensemble mean regression offset, and (ii) the transient mean vertical flows during each averaging period. For flows over uniformly tilted slopes the planar-fit method is applicable, while over more complex surfaces only the surface fit method is capable of this differentiation, because it not only considers wind direction (e.g., sectorial planar fit), but also wind magnitude. The transient mean vertical flows can contribute up to 25% to the total surface-atmosphere exchange over complex topography (Finnigan et al., 2003), and are only quantifiable with latter regression methods. Moreover, these methods (i) avoid high-pass filtering and cross-axis folding, (ii) provide a consistent frame of reference to assess the quality of the flux measurements over multiple days, (iii) enable tracking of the instrument alignment, and (iv) enable quantification of the uncertainty related to coordinate rotations.

In an application of the sectorial planar-fit method, Yuan et al. (2011) find that two hundred 30-min data sets (i.e., little over four days of data) are sufficient to derive stable planar fit coefficients. Based on the time-frequency work by Xu et al. (2017), we extended the initial planar fit period to 9 days. This intends to be long enough to capture a full cycle of synoptic-scale atmospheric motions, and to be short enough to resolve changing ecosystem phenology, both of which physical determinants of a stable (and meaningful) aerodynamic reference plane. As with all processing parameters, 9 days represents an observatory-wide initial value that is intended to be adjusted on site-specific (and potentially season-specific) basis during nominal operations of the NEON.

5.2.1.3 Lag-correction

Application of Eq. (1) requires that the instantaneous vertical wind w and scalar X are measured at the same place and at the same time, which is not presently possible, primarily due flow distortion issues with the sonic anemometer. Consequently, before applying Eq. (1), the recorded time series must be adjusted by a certain time lag to ensure spatiotemporal coincidence. The delay between the two time series is mainly caused by differences in electronic signal treatment, spatial separation between wind and scalar sensors, and air travel through the tubes in closed-path gas analyzers. Assuming joint stationarity, the lag time l can be estimated for each averaging interval by performing a cross correlation analysis between the quantities of interest;

$$\mathrm{abs}\left(\frac{\overline{w\prime(t)\cdot X\prime(t+l)}}{\overline{w\prime\cdot X\prime}}\right) \rightarrow \max, \tag{3}$$

for samples collected at times t and t+l. This is equivalent to comparing the correlations between the quantities lagged by different delays (Figure 4).

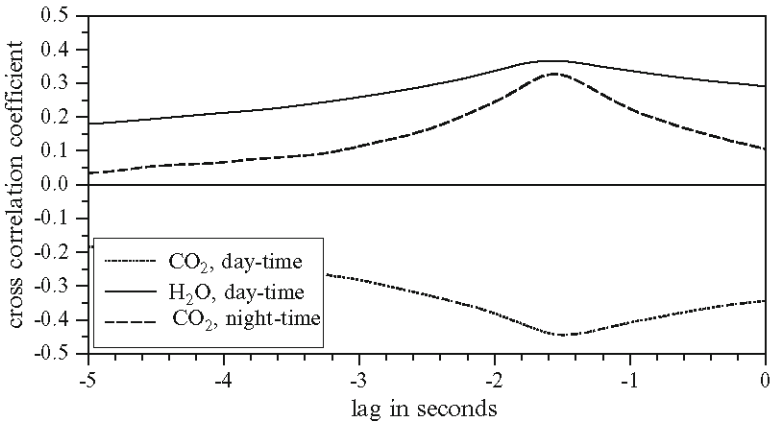


Figure 4. From Rebmann et al. (2012): Cross-correlation between the vertical wind component and CO_2 and H_2O for different lag times.

The time lag that results in the highest correlation is selected. However, when correlations are small this procedure can result in ambiguous lag times. Hence, high-pass filtering and pre-defining the maximum size of the cross-correlation search window aids in constraining the lag times to physically feasible values. The maximum size of the search window is found on the basis of known electronic delays, sensor separation and typical wind speeds, as well as mass flow and tube dimensions of closed-path gas analyzers. In cases where these limits are exceeded, Rebmann et al. (2012) recommend to use the value of the preceding averaging interval.

5.2.1.4 Sonic temperature conversions

A cross-wind correction (Campbell Scientific, 2011; Liu et al., 2001) is not necessary for the sonic anemometer operated at NEON sites (Campbell Scientific Inc., model CSAT-3 firmware: 3.0f; Logan, Utah, USA; Appendix C). However, the speed of sound in air is not only a function of air temperature, but also of humidity. Hence the temperature measurement by an ultrasonic anemometer/thermometer (SONIC) T_{SONIC} does not equal the air temperature, but includes a cross-dependence on humidity. A conversion is required to cancel this humidity dependence and to yield means, variances and covariances of air temperature T_{air}, respectively (Schotanus et al., 1983);

$$\overline{T_{\mathrm{air}}} = \frac{\overline{T_{\mathrm{SONIC}}}}{1+0.51\,\overline{FW_{\mathrm{mass,H2O}}}}, \tag{4}$$

$$\overline{T_{\mathrm{air}}'^2} = \overline{T_{\mathrm{SONIC}}'^2} - 1.02\,\overline{T_{\mathrm{air}}}\;\overline{T_{\mathrm{air}}' FW_{\mathrm{mass,H2O}}'} - (0.51\,\overline{T_{\mathrm{air}}})^2\;\overline{FW_{\mathrm{mass,H2O}}'^2}, \tag{5}$$

$$\overline{w' T_{\mathrm{air}}'} = \overline{w' T_{\mathrm{SONIC}}'} - 0.51\,\overline{T_{\mathrm{air}}}\;\overline{w' FW_{\mathrm{mass,H2O}}'}\,, \tag{6}$$

with wet mass fraction (specific humidity) $FW_{\mathrm{mass,H2O}}$. Eqs. (4)–(6) are linear approximations and ignore higher-order terms in their exact definitions. The magnitude of these conversions is in the order of 1–2%, and the accuracy of the approximation for temperature is $\leqslant$0.03 K for $0<FW_{\mathrm{mass,H2O}}<40$ g kg^{-1} H_2O, i.e. better than the accuracy of a sonic thermometer. It can be seen that the conversion of variance and covariance (Eqs. (5)–(6)) are subject to cross-dependence on ambient temperature T_{air} in terms $\overline{T_{\mathrm{air}}}$ and $\overline{T_{\mathrm{air}}' FW_{\mathrm{mass,H2O}}'}$ on the right-hand side. Hence Eqs. (5)–(6) must be solved iteratively, by first substituting T_{air} in respective terms on the right hand side with T_{SONIC}, and subsequently updating T_{air} with the outcomes after each cycle until the results for Eqs. (5)–(6) change by no more than 0.01% between iterations (e.g., Mauder and Foken, 2011). Moreover, T_{SONIC} closely resembles the virtual temperature T_{v}, with a difference in the humidity-related conversion in the order of 0.1%;

$$\overline{T_{\mathrm{air}}} = \frac{\overline{T_{\mathrm{v}}}}{1+0.61\,\overline{FW_{\mathrm{mass,H2O}}}}. \tag{7}$$

Consequently, $\overline{w' T_{\mathrm{SONIC}}'}$ is often used as surrogate for the buoyancy flux, e.g. in the computation of the Monin-Obukhov length (Rebmann et al., 2012).

5.2.1.5 Calculation of means, variance and standard error

The arithmetic mean of a quantity *X* (such as wind components *u*, *v*, *w*) with sample size *N* is calculated as;

$$\bar{X} = \frac{1}{N}\sum_{i=1}^{N} X_\mathrm{i}. \tag{8}$$

From here, the sample variance (*N*−1) and standard deviation of *X* are calculated;

$$\overline{X'^2} = \frac{1}{N-1}\sum_{i=1}^{N}(X_\mathrm{i} - \bar{X})^2, \tag{9}$$

$$\frac{\mathrm{std}_\mathrm{err}(X)=\sqrt{\overline{X'^2}}}{\sqrt{N}}. \tag{10}$$

5.2.1.6 High-frequency spectral correction

EC measurement systems, like all instruments, act as filters, removing both high- and low-frequency components of a signal. High-frequency losses are mainly due to inadequate sensor frequency response, line averaging, sensor separation and, in closed-path infrared gas analyzer (IRGA) systems, air transport through a filter and a tube (Foken et al., 2012). Figure 5 schematically illustrates the impact of high frequency loss in the measurement of an atmospheric scalar *X*, such as CO_2 or H_2O dry mole fraction, on spectral density. The frequency range of attenuation depends on the instrumental setup and especially the length and conditioning of the sample tube. It is often confined to frequencies beyond the spectral peak, which is referred to as the inertial subrange (ISR) of atmospheric turbulence, and for the NEON ECTE system design beyond 1 Hz under most conditions (Metzger et al., 2016). As can be seen from Eq. (2), high frequency losses in *X* (and, to a lesser degree, *w*) propagate into the ECTE flux measurement, and the corresponding cospectrum CO(*w*,*X*). Low-frequency losses result from the finite sampling duration, with the averaging period not always being sufficiently long to include all relevant low frequencies. The subject of this section is the correction of high-frequency losses, in order to avoid underestimating the variances and covariances of outputs from ECTE sensors. Low-frequency losses are planned to be addressed as part of future developments (Sect. 8).

Nordbo and Katul (2012, in the following referred to as NK12) have presented a Wavelet-based approach to high-frequency spectral corrections which (i) directly corrects the high-frequency data instead of the cospectrum, (ii) corrects each individual averaging period, and is thus able to take into account variations in environmental conditions (e.g., flow rate, relative humidity), (iii) does not assume cospectral similarity with heat, and (iv) does not rely on a theoretical shape for the velocity–scalar cospectrum, thereby making it advantageous to employ in non-ideal conditions. Furthermore, the method is not gas-specific, and can be used with very little input information at various sites. The method's largest insufficiency is its inability to correct attenuation starting already near the peak of power spectra, which however is explicitly taken into account in the design of NEON's ECTE. Consequently this is the method of choice for NEON, as it overcomes the drawbacks of the conventional "theoretical" and "empirical" approaches (Foken, 2017), and is fully automatable. Cospectral attenuation through sensor separation is not considered by the NK12 method. Instead it is explicitly addressed in Sect. 5.2.1.3 through maximization of the cross-correlation and by using an exponential decay model. NEON currently uses a simplified version of the NK12 method, which is described in Sect. 5.2.2.

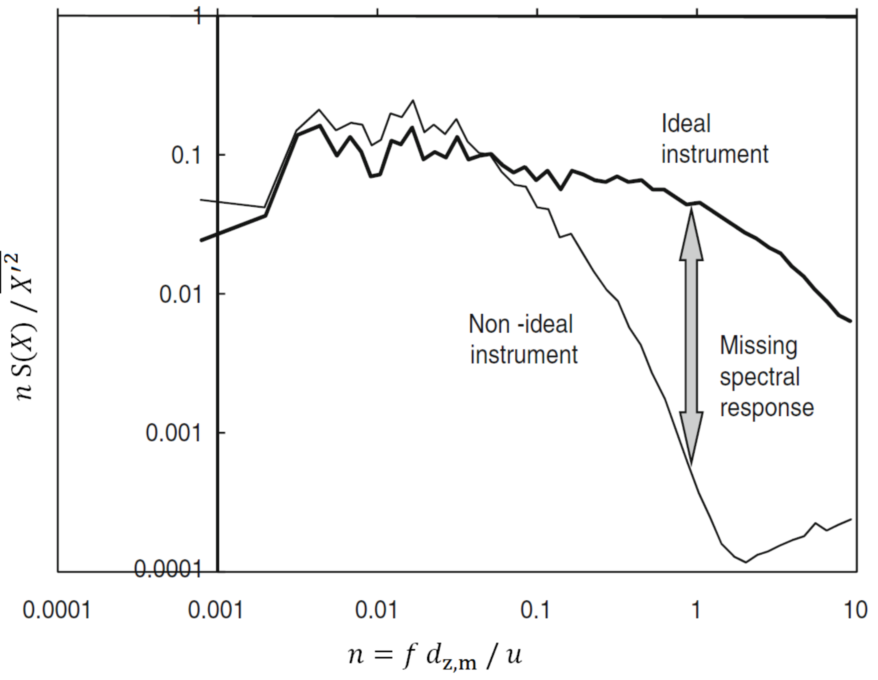


Figure 5. Normalized power spectrum for an ideal instrument which measures the unaffected spectrum of turbulence, and for a non-ideal instrument (Modified after Foken et al. (2012)).

The missing energy between both response curves must be corrected (normalized frequency *n*; measurement frequency *f*, measurement height $d_{z,m}$, along-wind speed *u*, power spectrum and variance of atmospheric scalar *X*, S(*X*) and $\overline{X'^2}$, respectively). The calculation of power spectra are detailed in AD[07].

5.2.1.7 Footprint modeling

A footprint model is used to determine where on the ground surface emissions measured by the ECTE system originated from. This allows interpretation of observed emission rates against hour-to-hour variations in flux footprint over surface properties such as land cover, soil moisture etc. e.g. from gridded remote-sensing data products (Figure 6). An in-depth review into footprint models and their continued development can be found in Leclerc and Foken (2014). Here, we initially use the footprint model described by Metzger et al. (2012). This builds upon the cross-wind integrated footprint model of Kljun et al. (2004), which quantifies the flux contribution relative to the distance away from the measurement position, into the prevailing wind direction. Metzger et al. (2012) coupled the model with a cross-wind distribution function, permitting to spatially resolve also flux contributions perpendicular to the wind direction. We intend to add outputs for the Kljun et al. (2015) footprint model as part of future developments (Sect. 8).

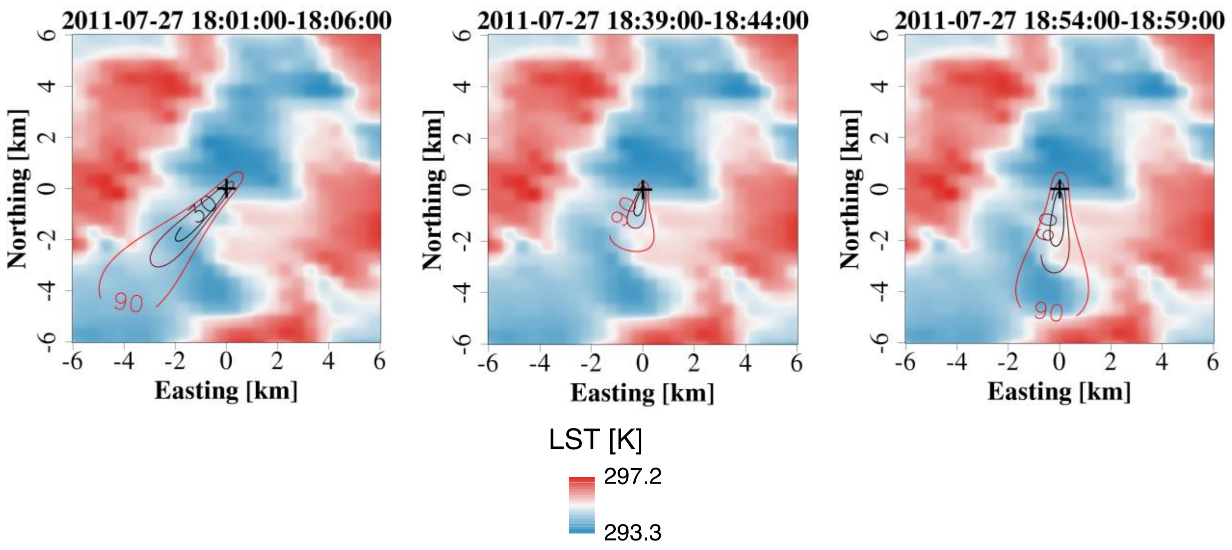


Figure 6. From Xu et al. (2017): example flux footprints (30%, 60% and 90%, contour lines) over MODIS-land surface temperature (LST).

5.2.2 Algorithmic implementation

The EC turbulent exchange data analysis is implemented as part of the eddy4R-Docker EC processing framework (Sect. 4). The corresponding R workflow flow.turb.tow.neon.dp04.r (link to public GitHub repo in preparation) and its algorithmic sequence are summarized in Figure 7. The calculations described in Sects. 5.2.1.1 – 5.2.1.4 are applied to yield the ECTE dp01 and dp04 data products in Table 1. The process is coded and documented in detail in the R-packages eddy4R.base and eddy4R.qaqc, consisting of the following sequence (incl. function references):

- Calculation is performed for datasets of 30 min time resolution (plus 1 min in case of dp01).
- De-spiking is performed at dp0p temporal resolution using the eddy4R.qaqc::def.dspk.br86() function.
- Derived variables at dp0p temporal resolution are calculated using the eddy4R.base::wrap.derv.prd.day() function. For example, specific humidity is calculated from dp0p inputs, so it can later be used in Eqs. (4)–(6).

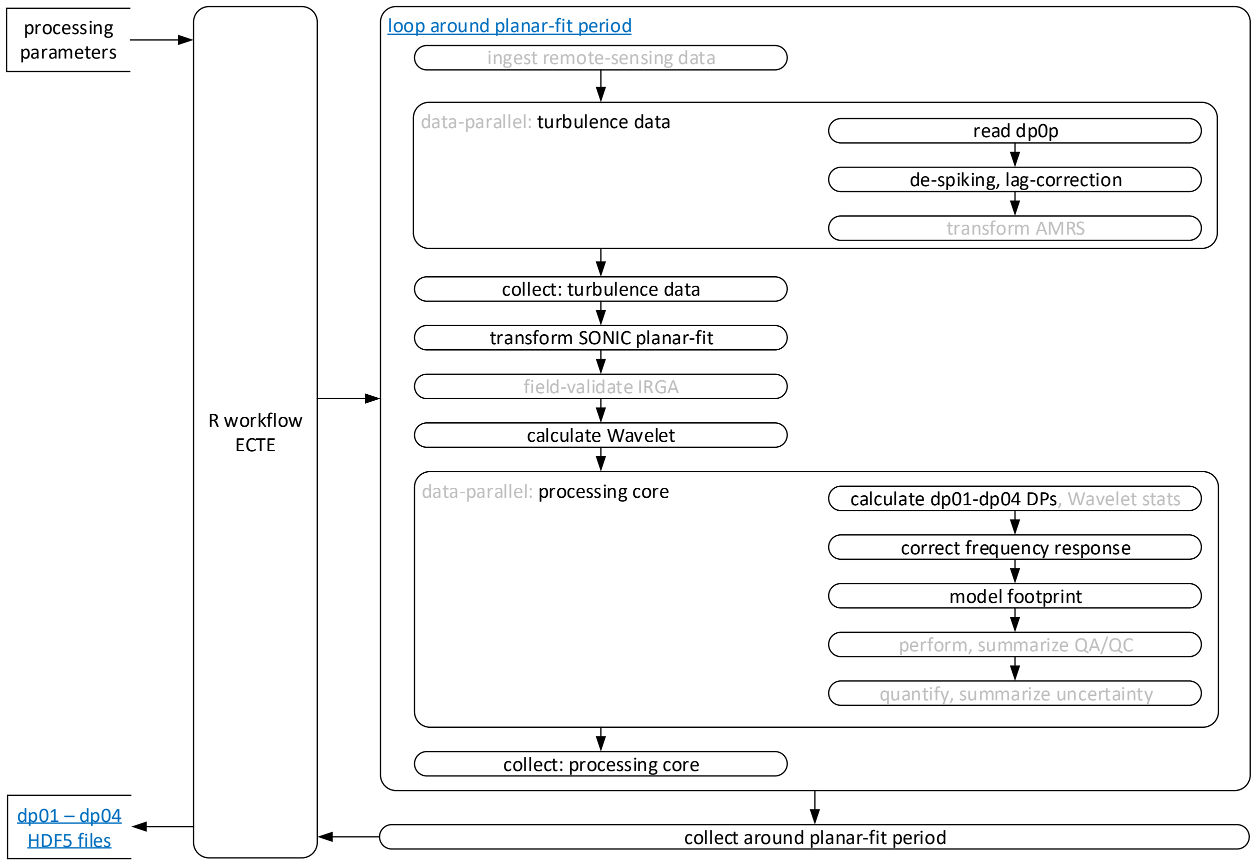

processing parameters
R workflow ECTE
loop around planar-fit period
ingest remote-sensing data
data-parallel: turbulence data
read dp0p
de-spiking, lag-correction
transform AMRS
collect: turbulence data
transform SONIC planar-fit
field-validate IRGA
calculate Wavelet
data-parallel: processing core
calculate dp01-dp04 DPs, Wavelet stats
correct frequency response
model footprint
perform, summarize QA/QC
quantify, summarize uncertainty
collect: processing core
collect around planar-fit period
dp01 – dp04 HDF5 files

Figure 7. The EC turbulent exchange workflow within the eddy4R-Docker EC processing framework (Sect. 4).

- Regression of the planar-fit coefficients is performed using the eddy4R.turb::PFIT_det() function over a moving, centered window of 9 days of 20 Hz dp0p data. Data points corresponding to bad sensor diagnostics and spikes are omitted from the regression. The eddy4R.turb::PFIT_apply function is then used to apply the regression coefficients and perform the planar-fit coordinate rotation for the central day of the moving window (day 5).
- Lag-correction is performed at dp0p temporal resolution using the eddy4R.base::def.lag() function.
- Sonic temperature is converted to air temperature.
- Descriptive statistics are calculated for averaging periods of 1 min and 30 min using the eddy4R.base::wrap.neon.dp01() function, and are available in the HDF5 file at:
  - SITE/dp01/data/amrs
  - SITE/dp01/data/co2Turb
  - SITE/dp01/data/h2oTurb
  - SITE/dp01/data/soni
- Turbulent fluxes are calculated for averaging periods of 30 min using the eddy4R.turb::REYNflux_FD_mole_dry() function, and are available in the HDF5 file at:
  - SITE/dp04/data/fluxCo2

- SITE/dp04/data/fluxH2o
- SITE/dp04/data/fluxMome
- SITE/dp04/data/fluxTemp

- Wavelet-based high-frequency spectral correction is performed on 30-min basis through the following sequence automated in the wrapper function eddy4R.turb::wrap.wave().
    - Periods with missing values >10% are being omitted. For all other periods, < 10% missing values are linearly interpolated.
    - The Waves::cwt() function then uses a Morlet mother Wavelet to perform the continuous Wavelet transform of the 3-D wind components, air temperature, as well as H_2O and CO_2 concentration.
    - In the eddy4R.turb::def.vari.wave() function the cross-scalograms with the vertical wind are calculated, and the absolute spectral power is scale-wise integrated to co-spectra. Then the power-law coefficient in the ISR of the unweighted co-spectrum is regressed in the frequency range 0.1 … 0.5 Hz. In case the coefficient exceeds the range of −1.8 … −1.3, the standard −5/3 power law decay is used. The reference spectral coefficients following the power slope are calculated, and the transfer function against the observed co-spectra is determined in the frequency range >0.5 Hz. The transfer function is then applied directly to the corresponding cross-scalogram. The ratio of the global Wavelet covariance after and before application of the transfer function provides the flux-specific correction factor, which is applied to the classical EC flux Eq. (2).
- Footprint calculation is performed on 30-min basis though the following sequence.
    - footprint model inputs incl. turbulence statistics are prepared and constrained to within the valid range of the Kljun et al. (2004) parameterization
        - relative measurement height above displacement (distZaxsMeasDisp)
        - wind direction (angZaxsErth)
        - standard deviation of the cross-wind (veloYaxsHorSd) and vertical wind (veloZaxsHorSd)
        - friction velocity (veloFric)
        - roughness length (distZaxsRgh; calculated via call to eddy4R.turb::def.dist.rgh())
        - boundary layer height (distZaxsAbl) is set to 1000 m by default
        - footprint matrix cell size (distReso) is set equal to relative measurement height above displacement, and rounded to 10 m
    - the square footprint weight matrix with 301 x 301 cells and the tower at its center is calculated through calling eddy4R.turb::footK04()
    - footprint statistics are calculated
        - along-wind distance of the 90 percent crosswind-integrated cumulative footprint (distXaxs90)
        - along-wind distance of contribution peak (distXaxsMax)
        - one-sided cross-wind distance of the 90 percent along-wind integrated cumulative footprint (distYaxs90)

- location of results in HDF5 file
    - model inputs and footprint statistics are included in the basic and expanded HDF5 files: SITE/dp04/DQU/foot/stat
    - half-hourly footprint weight matrices are only included in the expanded HDF5 files: SITE/dp04/DQU/foot/grid/turb

5.3 Quality Assurance and Quality Control analysis

In general, the quality flags (*QF*s) are generated for each test and each *QF* can be set to one of three states as shown in Eq. (11) (AD[06]).

$$QF = \begin{cases} 1 \text{ if the quality test failed} \\ 0 \text{ if the quality test passed} \\ -1 \text{ if NA i. e. not able to be run due to a lack of ancillary data} \end{cases} \quad (11)$$

In extension, specifically for EC data products, combinations of data quality and data availability signifiers are used to express a number of conditions in the HDF5 file (Table 15):

- Condition A: Data are available and good
    - Data are expected and available [Data ≠ NaN]
    - Data pass a critical number of quality tests [QF = 0]
- Condition B: Data are available but bad
    - Data are expected and available [Data ≠ NaN]
    - However, data fail a critical number of quality tests [QF = 1]
- Condition C: Data are available but user discretion advised
    - Data are expected and available [Data ≠ NaN]
    - However, not all quality tests can be evaluated due to missing dependency data [QF = −1]
- Condition D: Data are not available but expected
    - Data are expected from a particular sensor or measurement level, but are not available [Data = NaN]
    - Data quality cannot be assessed due to missing data or dependency [QF = −1]
    - For example: a quality test requires variables from auxiliary sensors such as the mass flow controller: in the case that mass flow controller data are not available the test cannot be executed and the test result of QF=-1 is assigned.
- Condition E: Data are not available and not expected
    - Data are not expected from a particular sensor or measurement level [Data = NaN]
    - Data quality is not assessed as data is not expected [QF = NA]
    - For example: Profile system with a single analyzer cycles through measurement levels, thus data availability at individual levels is discontinuous. After regularization to create a continuous time series, these data points are not expected to be measured by the analyzer and are represented by NaN, thus the QF = NA.

Table 15. Lookup table of joint data availability and data quality conditions which apply to data products in this ATBD.

	QF = 0	**QF = 1**	**QF= −1**	**QF= NA**
Data ≠ NaN	A	B	C	Not applicable
Data = NaN	Not applicable	Not applicable	D	E

Sensor and statistical QA/QC tests are performed on and reported for the dp0p data (e.g. 20 Hz), while flux QA/QC tests are reported on time-integrated data per flux averaging period (e.g. 30 min). Here, we utilize the NEON data quality framework as described in AD[06] and Smith et al. (2014) to summarize the results from sensors test and QA/QC tests in a way that is transparent and easily interpretable. In the following, these sensor health and statistical QA/QC tests are first aggregated to the flux averaging period, and then combined with the results for flux QA/QC tests to determine the QF_{FINAL}.

5.3.1 Theory of Algorithm

A wide range of qualitative and quantitative algorithmic processing routines are applied to EC data products including:

1. Tests related to sensor diagnostics (AD[02]);
2. Statistical plausibility tests, e.g. range, persistence, step (AD[02] and AD[06]);
3. EC-specific tests based on the degree of fulfillment of one or several methodological assumptions, e.g. detection limit, homogeneity and stationarity, development of turbulence tests.

5.3.1.1 Sensor quality flags

Most of quality flags due to the sensor health and statistical plausibility tests are generated as part of the dp0p report variables (AD[02]). In addition, the IRGA validation flag (qfIrgaVali) and IRGA automatic gain control quality flag (qfIrgaAgc) were also generated in this ATBD, which are defined below.

1. **IRGA Validation flag** (qfIrgaVali) – is generated to indicate when the sensor is operated under validation period (1 = validation period, 0 = normal operating condition, −1 = NA). The IRGA validation flag is determined from the IRGA sampling mass flow controller flow rate set point as follow:

$$\mathrm{qfIrgaVali} = \left| \begin{array}{l} 1 \text{ if } \mathrm{frtSet00} = 0 \\ \\ 0 \text{ if } 0.0001333 \le \mathrm{frtSet00} \le 0.00025 \\ \\ -1 \; otherwise. \end{array} \right. \qquad (12)$$

where frtSet00 is the flow rate set point from IRGA sampling mass flow controller (irgaMfcSamp) in the unit of m^3 s^{-1}.

2. ***IRGA automatic gain control quality flag*** *(qfIrgaAgc)* is indicating when the sensor is operating with low signal strength using 50 percent as the default threshold (1 = when qfIrgaAgc <= 0.50, 0 = when qfIrgaAgc >= 0.50, -1 = NA).

5.3.1.2 Quality budget (QFQM)

The theory of algorithm, the definition of quality flag (*QF*), quality metric (*QM*), alpha (α) and beta (β) *QF*s and *QM*s are detailed in AD[05]. Each of EC DP will have QF_{FINAL}, QM_{α}, and QM_{β} associated with it. Aside from QF_{FINAL}, QM_{α}, and QM_{β}, each EC DP will also be accompanied by *QM* results for individual tests, representing the fractional occurrence of each state that a quality flag can take.

In order to determine the QF_{FINAL} (Eq. (12) – (13)) individually for each DP, the sensor health and statistical plausibility tests are first used to calculate QM_{α}, and QM_{β} over the averaging period:

$$QF_{\text{FINAL}} = \begin{cases} 1 & if\ (a \cdot QM_{\beta}) + (b \cdot QM_{\alpha}) \geq 20\% \text{ or} \\ & QF_{\text{spec}} = 1 \text{ or} \\ & QF_{\text{sciRevw}} = 1 \\ 0 & otherwise \end{cases} \tag{13}$$

where a and b are the ratio of QM_{α} to QM_{β} with maximums of 10% for QM_{α} and 20% for QM_{β} (more details can be found in AD[05] and (Smith and Metzger, 2013). Therefore, by default a and b are set to 1 and 2, respectively. Then, the results of EC specific (QF_{spec}) tests (i.e., detection limit, homogeneity and stationarity, development of turbulence tests) are taken into account to determine whether the data product is flagged as valid (QF_{FINAL} = 0) or invalid (QF_{FINAL} = 1). If the scientific review flag (QF_{sciRevw}) is set high during science operation management (SOM) review then QF_{FINAL} will be set high.

5.3.2 Algorithmic implementation

The EC turbulent exchange data quality analysis is implemented as part of the eddy4R-Docker EC processing framework (Sect. 4), and the algorithmic sequence is summarized in Figure 7. The calculations described in Sects. 5.3.1.1 – 5.3.1.2 are applied to all data products of product level "dp01 statistics" in Table 1. The process is coded and documented in detail in the R-package eddy4R.qaqc, consisting of the following sequence (incl. function references):

- Calculation is performed individually for datasets of 1 min and 30 min duration.

- Derived quality flags at dp0p temporal resolution for IRGA validation period and AGC are calculated using the eddy4R.qaqc::def.qf.irga.vali and eddy4R.qaqc::def.qf.irga.agc functions, respectively.
- Quality flags are combined into quality metrics using the eddy4R.qaqc::wrap.neon.dp01.qfqm function.
  - The final quality flag for each reported dp01 are included in the basic and expanded hdf5 files:
    - SITE/dp01/qfqm/amrs
    - SITE/dp01/qfqm/co2Turb
    - SITE/dp01/qfqm/h2oTurb
    - SITE/dp01/qfqm/soni
  - The quality metrics, alpha and beta quality metrics are only included in expanded HDF5 files:
    - SITE/dp01/qfqm/amrs
    - SITE/dp01/qfqm/co2Turb
    - SITE/dp01/qfqm/h2oTurb
    - SITE/dp01/qfqm/soni
- As part of future developments (Sect. 8) we plan to implement end-to-end quality propagation also to dp04 data products (fluxes).

5.4 Uncertainty analysis

5.4.1 Theory of Algorithm

Random errors are defined as the errors due to time averaging over an insufficient period for the time mean to converge to the ensemble mean by the ergodic hypothesis (Lenschow and Stankov, 1986; Lenschow et al., 1994; Lumley and Panofsky, 1964; Mann and Lenschow, 1994).

Here, the random sampling error is estimated using the method of Salesky et al. (2012). In comparison to other available approaches (e.g., Finkelstein and Sims, 2001; Hollinger and Richardson, 2005; Lenschow et al., 1994), the Salesky et al. (2012) method does not require an estimate of the integral time scale or replicate tower measurements. It is also equally applicable to statistical moments of any order, i.e. means, variances and covariance alike. Principally, the method consists of three parts, (i) a local time-series decomposition, (ii) the fitting of a power-law, and (iii) the inter- or extrapolation of the power law.

(i) The dp0p time-series is low-pass filtered using a running mean filter. This is performed for several filter window sizes $time_{filt}$ in the range 10 $time_{scal} < time_{filt} < time_{agr}$ / 10. Here, $time_{scal}$ is the integral time scale of the process (assumed to be ~1 s), and $time_{agr}$ the duration of the dataset available for aggregation (1,800 s). For each low-pass filtered time-series the standard deviation is calculated as representation of the random error associated with averaging over *time*$_{\text{filt}}$. The random error decreases with increasing window-size of the low-pass filter (Figure 8).

(ii) Next, a power-law in the form of $\sigma = coef_{01}\ time_{\text{filt}}{}^{coef_{02}}$ is regressed to the results (Figure 8). Here, *coef*$_{01}$ and *coef*$_{02}$ define the slope and convexity of the uncertainty reduction with increasing window-size of the low-pass filter, respectively. Salesky et al. (2012) relate a value of *coef*$_{02}$ = −1/2 to the power law decay of random error as derived e.g. by Lenschow et al. (1994) for Gaussian and stationary turbulence. Salesky et al. (2012) restrict their analysis to stationary data and thus permit regression only of *coef*$_{01}$. In order to also accommodate non-stationary data we additionally permit regression of −1/2 < *coef*$_{02}$ < 0. It should be noted that for *coef*$_{02}$ → 0 the power law becomes less convex, resulting in less uncertainty reduction with increasing window-size. The resulting algorithm such provides a conservative random error estimate for non-stationary data.

(iii) Lastly, *time*$_{\text{filt}}$ in the resulting power law is substituted with the target averaging periods, yielding the corresponding random error.

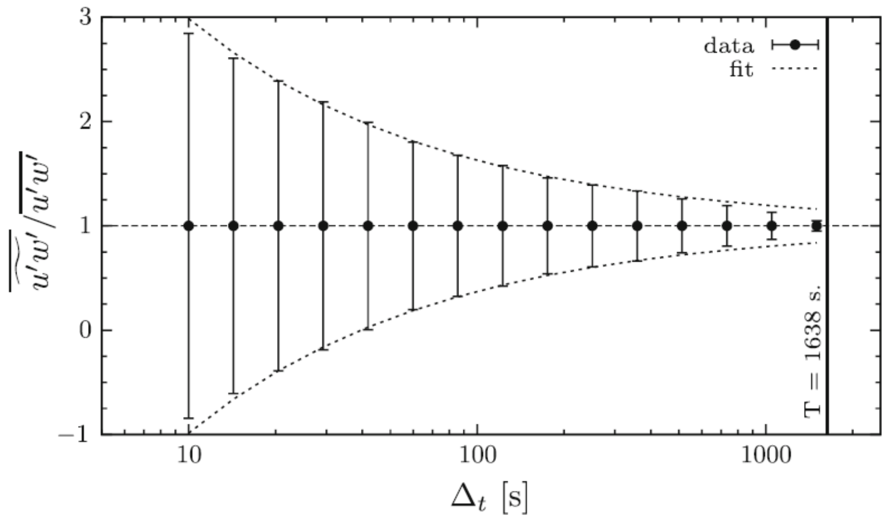


Figure 8. Reduction of standard deviation with increasing window size of the low-pass filter, from Salesky et al. (2012). The error bars denote the standard deviation, and the dashed line denotes a power-law fit.

5.4.2 Algorithmic implementation

The EC turbulent exchange data uncertainty analysis is implemented as part of the eddy4R-Docker EC processing framework (Sect. 4), and the algorithmic sequence is summarized in Figure 7. The random error calculation described in Sect. 5.4.1 is applied to all data products in Table 1. It is coded and documented in detail in the R-function eddy4R.ucrt:: def.ucrt.samp.filt(). In short:

- Calculation is performed individually for datasets of 30 min duration.

- Calculation is only performed if there are less than 10% missing values in the dataset. If less than 10% missing values, those are filled using linear interpolation.
- The signal is de-trended and tapered.
- Filtering is performed using Fast Fourier transform for 10 exponentially spaced filter widths in the range 10 s < $time_{\text{filt}}$ < 180 s.
- Nonlinear least squares regression is used to fit the power law.
- The random sampling error is calculated for averaging periods of 1 min and 30 min, and available in the HDF5 file at:
  - SITE/dp01/ucrt/amrs
  - SITE/dp01/ucrt/co2Turb
  - SITE/dp01/ucrt/h2oTurb
  - SITE/dp01/ucrt/soni
- As part of future developments (Sect. 8) we plan to implement end-to-end uncertainty quantification and propagation to dp04 data products (fluxes).

6 STORAGE EXCHANGE

The Eddy Covariance Storage Exchange Assembly (or EC profile assembly, hereafter referred to as the ECSE) consists of a suite of sensors such as temperature, CO_2 and H_2O gas analyzer and isotopic CO_2 and H_2O analyzers. The EC profile assembly is served to provide the measurements of temperature, CO_2 and H_2O concentration, the stable isotope of δ^{13}C in CO_2, δ^{18}O, and δ^{2}H in water vapor in the atmosphere at each tower measurement level. The vertical profile measurements of temperature, CO_2 and H_2O concentration will be used to calculate the storage fluxes, which will be incorporated into the calculation of the net ecosystem exchange of temperature, CO_2 and H_2O.

6.1 Theory of Measurement

In Eq. (2), NSAE of the control volume is expressed by the turbulent flux alone, based on several assumptions. Strict stationary is one of those assumptions, which implies that storage flux is negligible, i.e. the abundance of the scalar in the control volume remains constant. Because in tall and dense canopies this assumption is frequently violated, NEON measures and calculates the storage flux explicitly. With storage flux data products, NSAE is then calculated as the sum of storage flux and turbulent flux, which is described in more detail in Sect. 7.

An IRGA that is housed in the instrument hut switches between different measurement levels (usually between 4 and 6 levels) of each flux tower. At each measurement level, the IRGA measures CO_2 and H_2O for about 2 minutes. The storage of heat is calculated from the temperature profile measurements. Here we calculate the storage flux of the control volume based on the assumption that the temperature profile and the switched IRGA measurements at several levels can represent the vertical integral of time rate of change of the scalar over the entire control volume.

6.2 Data Analysis

6.2.1 Theory of Algorithm

The subject of this ATBD section is to describe the theory of algorithms to process the stable isotope of δ^{13}C in CO$_2$, δ^{18}O and δ^{2}H in water vapor in the atmosphere, and describe the mathematical derivation of the storage term, $\int_0^{d_{z,m}} \frac{\partial \bar{X}}{\partial t} dz$, in Eq. (1) in Sect. 5.1 and expressed as $fluxStor$ in Eq. (15) below in Sect. 0. Only dp01 stable isotope data will be computed in this ATBD, while dp01 to dp04 data products will be computed for CO$_2$,H$_2$O , and temperature. The computation for dp01-dp04 data products follows the steps described below.

6.2.1.1 De-spiking

This part is the same as Sect. 5.2.1.1. This is applied to all time series signals collected for ECSE assembly.

6.2.1.2 Calculation of means, variance and standard error

This part is the same as Sect. 5.2.1.5, but X in the equations refers to temperature, CO$_2$ and H$_2$O concentration, stable isotope of δ^{13}C in CO$_2$, δ^{18}O and δ^{2}H in water vapor in the atmosphere in ECSE dp01, instead of turbulent fluxes in ECTE dp01.

6.2.1.3 Calculation of time rate of change

The time rate of change in ECSE dp02 is calculated from the time average of the four-minute measurement at the end of a half hour minus the time average of four minute measurements at the beginning of the same half hour (Eq.(14)). For example, assuming that storage flux estimates are to be computed for timestamps 07:30:00, then by convention the 07:30:00 timestamp represents the flux corresponding to observations between 07:30:00 and 08:00:00. dX, where X stands for temperature, CO$_2$ concentration, or H$_2$O concentration, will then be computed from the time average of measurements from 07:58:00 to 08:02:00 minus the time average of measurements from 07:28:00 to 07:32:00.

$$\frac{\mathrm{dX}}{dt} = \frac{\bar{\mathrm{X}}_{t \geq t_e - 120s \; and \; t < t_e + 120s} - \bar{\mathrm{X}}_{t \geq t_b - 120s \; and \; t < t_b + 120s}}{30 \; min} \tag{14}$$

Where t_b is the beginning time of the first minute in the 30 minute block, and t_e is the last minute of the 30 minute block.

To calculate ECSE dp02, ECSE dp01 are firstly linearly interpolated into 1 minute resolution. Both the interpolation method and resolved temporal resolution can be adjusted for different locations and in different applications.

The averaging time is chosen as four minutes at the beginning and end of each half hour window because following Finnigan (2006), storage flux estimates influenced by single eddies penetrating inside the canopy should be avoided. The time period for the storage flux computation should be long enough to capture an adequate number of these eddies biasing the profiles or single observational points. Here, the period of storage flux computation is chosen based on 10 times of the time variable τ , the integral time scale of the turbulent time series between these eddies. Given the wide range in turbulence characteristics existing at the NEON ecosystem sites, and initially missing experimental evidence for all site conditions, we consider an adequate time of integration a period between 180 s and 300 s, with the shorter integration time to be reserved to turbulent conditions as observed in short canopies, 300 s to be reserved for dense canopies of 30 m or above, and the median value 240 s as default value. The temporal resolution of ECSE dp02 is set to be half hour for combination with ECTE data products.

ECSE dp03 is the vertically resolved time rate of change based on ECSE dp02. The interpolation method is linear interpolation. The resolved vertical resolution is prescribed as 0.1 m. Both the interpolation method and vertical resolution can be adjusted for different locations and different applications.

6.2.1.4 Calculation of storage flux

The storage flux in ECSE dp04 is integrated from vertical profiles of time rate of change for each date product using the equation:

$$fluxStor(X) = \frac{\mathrm{d}\overline{\mathrm{X}}}{dt} \cdot (\max(DistZaxsLvlMeasTow)) \tag{15}$$

Where *fluxStor* is storage flux, *X* is a scalar quantity such as H_2O or CO_2 mixing ratios, $DistZaxsLvlMeasTow$ is profile measurement heights.

This storage flux is part of the carbon dioxide flux listed as dp04 data product in Table 1.

6.2.2 Algorithmic implementation

The ECSE data analysis is implemented as part of the eddy4R-Docker EC processing framework (Sect. 4). The corresponding R workflow flow.stor.towr.neon.R (link to public GitHub repo in preparation) and its algorithmic sequence are summarized in Figure 9. The calculations described in Sects. 6.2.1.1 – 6.2.1.4 are applied to generate the ECSE dp01 – dp04 in Table 1. The process is coded and documented in detail in the R-packages eddy4R.base and eddy4R.qaqc.

6.2.2.1 ECSE dp01

IRGA CO_2 concentration (co2Stor) and IRGA H_2O concentration (h2oStor)

ECSE dp01 data (appears as SITE/dp01/data/co2Stor and dp01/data/h2oStor in HDF5 files) includes the descriptive statistics, mean, minimum, maximum, variance, standard deviation, number of samples, as well as begin time and end time, of **co2Stor** and **h2oStor** sub- data products at 2 min and 30 min resolution. The current NEON processing design utilizes the eddy4R package within a Docker framework to read in ECSE dp0p HDF5 files, do de-spiking, calculate descriptive statistics, and output HDF5 dp01.

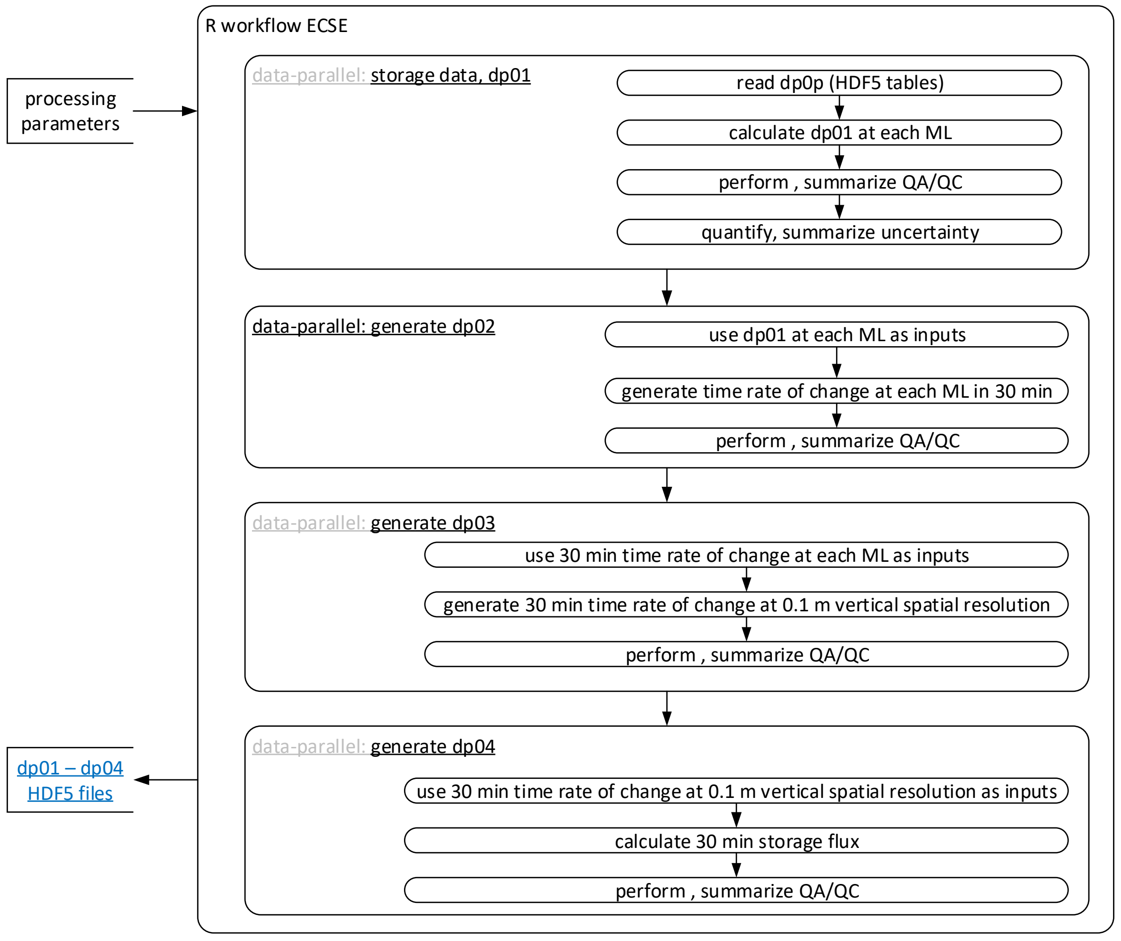

R workflow ECSE
processing parameters
data-parallel: storage data, dp01
read dp0p (HDF5 tables)
calculate dp01 at each ML
perform , summarize QA/QC
quantify, summarize uncertainty
data-parallel: generate dp02
use dp01 at each ML as inputs
generate time rate of change at each ML in 30 min
perform , summarize QA/QC
data-parallel: generate dp03
use 30 min time rate of change at each ML as inputs
generate 30 min time rate of change at 0.1 m vertical spatial resolution
perform , summarize QA/QC
data-parallel: generate dp04
use 30 min time rate of change at 0.1 m vertical spatial resolution as inputs
calculate 30 min storage flux
perform , summarize QA/QC
dp01 – dp04 HDF5 files

Figure 9. R workflow for eddy-covariance storage exchange (ECSE). Note: ML stands for measurement level.

Data flow for signal processing of dp01 IRGA CO_2 concentration (**co2Stor**) and IRGA H_2O concentration (**h2oStor**) will be treated in the following order.

- Calculation is performed individually for datasets of 2 min and 30 min duration.
- For each measurement of sampling data, e.g. the data under 000_0n0 folder, the middle two minute data after the first one minute critical time are selected for further calculation, while the critical data are set to be NaN.
- For each measurement of validation data, e.g. the data under co2XXX, the middle two minute before the last 20 s are selected for further calculation, while the last 20 s are set to be NaN.
- De-spiking is performed at dp0p temporal resolution using the eddy4R.qaqc::def.dspk.br86() function.
- Descriptive statistics are calculated using the eddy4R.base::wrap.neon.dp01() function.

- 2 and 30 min averages of description statistics are included in the basic and expanded HDF5 files: SITE/dp01/data/co2Stor and SITE/dp01/data/h2oStor.

Stable isotope of δ^{13}C in CO_2 (isoCo2)

ECSE dp01 data (appears as dp01/data/isoCo2 in HDF5 files) includes the descriptive statistics, mean, minimum, maximum, variance, standard deviation, number of samples, as well as begin time and end time, of **isoCo2** sub-data products at 9 and 30 min resolution for sampling and validation periods. The current NEON processing design utilizes the eddy4R package within a Docker framework to read in ECSE dp0p HDF5 files, do de-spiking, calculate descriptive statistics, and output HDF5 dp01.

Before the descriptive statistics are calculated, the data in the prescribed temporal interval should be selected. During the sampling and validation period, each measurement level or each validation gas type is sampled for 10 min. Each time the first 1 min existing data are discarded in order to make sure the gas from previous sample has been cleaned from the flow. Therefore, the descriptive statistics of **isoCo2** sub-data products are calculated only using the next 9 min measurements.

Data flow for signal processing of dp01 is as follows:

- De-spiking is performed at dp0p temporal resolution using the eddy4R.qaqc::def.dspk.br86() function.
- Determine which data in the timestamp will be used in descriptive statistics calculation for each temporal interval using the eddy4R.base::def.idx.agr() function.
- Descriptive statistics are calculated using the eddy4R.base::wrap.neon.dp01() function
    - 9 and 30 min averages of description statistics are included in the basic and expanded HDF5 files: SITE/dp01/data/isoCo2

Stable isotopes of δ^{18}O, and δ^{2}H in water vapor (isoH2o)

ECSE dp01 data (appears as dp01/data/isoH2o in HDF5 files) includes the descriptive statistics, mean, minimum, maximum, variance, standard deviation, number of samples, as well as begin time and end time, of **isoH2o** sub-data products at 9 and 30 min resolution for sampling period and 3 and 30 min for validation period. The current NEON processing design utilizes the eddy4R package within a Docker framework to read in ECSE dp0p HDF5 files, do de-spiking, calculate descriptive statistics, and output HDF5 dp01.

Before the descriptive statistics are calculated, the data in the prescribed temporal interval should be selected. During the sampling period, each measurement level is sampled for 10 min. Each time the first 1 min existing data are discarded in order to make sure the gas from previous sample has been cleaned from the flow. Therefore, the descriptive statistics of **isoH2o** sub-data products are calculated only using the next 9 min measurements.

During the routine field validation of the CRD H_2O, the analyzer will cease to measure the atmospheric vapor from the tower profiles and measure water standards by using the zero air as a carrier gas. Water standards are injected through the vaporizer using the autosampler and a syringe. Field validation is performed for 3 standards (NEON Tertiary Low, Mid, and High standard) and each standard is injected 6 times. The procedure typically takes around 9 minutes per injection, the descriptive statistics are calculated using the data when the water concentration is stabilized which defined as the 3 min measurements right before the last 15 s in each injection time.

Data flow for signal processing of dp01 **isoH2o** is as follows:

- De-spiking is performed at dp0p temporal resolution using the eddy4R.qaqc::def.dspk.br86() function.
- Determine which data in the timestamp will be used in descriptive statistics calculation for each temporal interval using the eddy4R.base::def.idx.agr() function.
- Descriptive statistics are calculated using the using the eddy4R.base::wrap.neon.dp01() function.
    - 9 and 30 min averages of description statistics during sampling period are included in the basic and expanded HDF5 files: SITE/dp01/data/isoH2o
    - 3 and 30 min averages of description statistics during validation period are included in the basic and expanded HDF5 files: SITE/dp01/data/isoH2o

6.2.2.2 ECSE dp02

Data flow for signal processing of ECSE dp02 is as follows:

- Linear interpolation is performed for ECSE dp01 at temporal resolution into 1 min resolution with the maximum gap of 40 min.
- Time rate of change of temperature, CO_2 concentration and H_2O concentration for each measurement level was calculated using Eq. (14) in Sect. 6.2.1.3.
    - half-hourly time rate changes of temperature are included in the basic and expanded HDF5 files: SITE/dp02/data/tempStor
    - half-hourly time rate changes of CO_2 concentration are included in the basic and expanded HDF5 files: SITE/dp02/data/co2Stor
    - half-hourly time rate changes of H_2O concentration are included in the basic and expanded HDF5 files: SITE/dp02/data/h2oStor

6.2.2.3 ECSE dp03

Data flow for signal processing of ECSE dp03 is as follows:

- Linear interpolation is performed for all ECSE dp02 spatially into 0.1 m spatial resolution by default. If only one measurement level is available at a time, it is assigned to all vertical levels at the time in ECSE dp03. If no measurement level is available at a time, NaN is assigned to all vertical levels at the time.
    - half-hourly time rate changes at 0.1 m vertical spatial resolution of temperature are included in the basic and expanded HDF5 files: SITE/dp03/data/tempStor
    - half-hourly time rate changes at 0.1 m vertical spatial resolution of CO_2 concentration are included in the basic and expanded HDF5 files: SITE/dp03/data/co2Stor
    - half-hourly time rate changes at 0.1 m vertical spatial resolution of H_2O concentration are included in the basic and expanded HDF5 files: SITE/dp03/data/h2oStor

6.2.2.4 ECSE dp04

Data flow for signal processing of ECSE dp04 is as follows:

- Storage flux is calculated based on all ECSE dp03 according to Eq. (15)
    - half-hourly storage fluxes of temperature are included in the basic and expanded HDF5 files: SITE/dp04/data/fluxTemp/stor
    - half-hourly storage fluxes of CO_2 are included in the basic and expanded HDF5 files: SITE/dp04/data/fluxCo2/stor
    - half-hourly storage fluxes of H_2O are included in the basic and expanded HDF5 files: SITE/dp04/data/fluxH2o/stor

6.3 Quality Assurance and Quality Control analysis

The basic quality assurance and quality control analysis can be found in Sect. 5.3. However, sensor and statistical QA/QC tests are performed on and reported for the 1 Hz data for ECSE.

6.3.1 Theory of Algorithm

Details can be found in Sect. 5.3.1.

6.3.1.1 Sensor quality flags

Most of quality flags due to the sensor health and statistical plausibility tests are generated as part of the dp0p report variables (AD[03]). In addition, the water validation quality flag (qfValiH2o) was also generated in this ATBD, which is defined below.

Water Validation quality flag (qfValiH2o) – is indicating when the validation of crdH2o sensor is good (0) or bad (1). Field validation of crdH2o is performed using 3 standards (NEON Tertiary Low, Mid, and High standard) and each standard is injected 6 times. The qfValiH2o of the first 3 injections of each standard (injection number 1, 2, 3, 7, 8, 9, 13, 14, and 15) will be set to 1. For the rest (injection number 4, 5, 6, 10, 11, 12, 16, 17, and 18), the qfValiH2o is set to 1 if the measurements values of dlta18OH2o and dlta2HH2o are greater or less than the thresholds. As default, thresholds can be can calculated as the reference value ± 30% of reference value.

6.3.1.2 Quality budget (QFQM)

Details of the basic quality budget that applied to ECSE dp01 will be identical to ECTE, which can be found in Sect. 5.3.1.2. The QF_{FINAL} of ECSE dp02 and dp03 is estimated from surrounding values. For example, if one or more QF_{FINAL} of ECSE dp01 which used to determine ECSE dp02 is equal to 1, assign QF_{FINAL} of that ECSE dp02 to 1. Similarly, QF_{FINAL} of that ECSE dp03 is assigned to 1 if one or more QF_{FINAL} of ECSE dp02 which used to determine ECSE dp03 is equal to 1.

6.3.2 Algorithmic implementation

The calculations described in Sects. 6.3.2.1 - 6.3.2.4 are applied to ECSE dp01 – dp04 in Table 1. The process is coded and documented in detail in the R-packages eddy4R.base and eddy4R.qaqc, consisting of the following sequence (incl. function references).

6.3.2.1 ECSE dp01

Data flow for QA/QC processing of ECSE "dp01 statistics" (dp01) will be treated in the following order.

- Calculation is performed individually for datasets of each duration.
- Using the eddy4R.base::def.idx.agr function to determine the datasets of 2 min and 30 min duration for **co2Stor** and **h2oStor**, 9 min and 30 min duration for **isoCo2**, 9 min and 30 min duration for **isoH2o** during sampling period and 3 min and 30 min duration for **isoH2o** during validation period.
- Derived qfValiH2o at dp0p temporal resolution for **isoH2o**.
- Using eddy4R.qaqc::def.neon.dp01.qf.grp function to indicate which quality flags are used as the input variables to determine alpha and beta quality metrics, and final quality flag for each reported dp01.
- Calculated quality metrics, alpha and beta quality metrics, and final quality flag for each reported dp01 using the eddy4R.qaqc::wrap.neon.dp01.qfqm function.
    - The final quality flag for each reported dp01 are included in the basic and expanded HDF5 files:
        - SITE/dp01/qfqm/co2Stor
        - SITE/dp01/qfqm/h2oStor
        - SITE/dp01/qfqm/isoCo2
        - SITE/dp01/qfqm/isoH2o
    - The quality metrics, alpha and beta quality metrics are only included in expanded HDF5 files:
        - SITE/dp01/qfqm/co2Stor
        - SITE/dp01/qfqm/h2oStor
        - SITE/dp01/qfqm/isoCo2
        - SITE/dp01/qfqm/isoH2o

6.3.2.2 ECSE dp02

Data flow for QA/QC processing of ECSE dp02 is as follows:

- Interpolated QF_{FINAL} of ECSE dp01 at temporal resolution into 1 min resolution by assigning the QF_{FINAL} to 1 to ECSE dp01 that fell in between two adjacent available data and if one or more QF_{FINAL} of two adjacent available data is equal to 1. Otherwise, interpolated QF_{FINAL} values are equal to 0.
- For each half-hourly, calculated QF_{FINAL} of ECSE dp02 by assigning the QF_{FINAL} of that half-hourly to 1 if one or more QF_{FINAL} of ECSE dp01 which used to determine that ECSE dp02 is equal to 1. Otherwise, QF_{FINAL} of ECSE dp02 at that half-hourly is equal to 0.

- half-hourly summarized of the final quality flag of time rate changes of temperature are included in the basic and expanded HDF5 files: SITE/dp02/qfqm/tempStor
- half-hourly summarized of the final quality flag of time rate changes of CO_2 are included in the basic and expanded HDF5 files: SITE/dp02/qfqm/co2Stor
- half-hourly summarized of the final quality flag of time rate changes of H_2O are included in the basic and expanded HDF5 files: SITE/dp02/qfqm/h2oStor

6.3.2.3 ECSE dp03

Data flow for QA/QC processing of ECSE dp03 is as follows:

- For each half-hourly, interpolated QF_{FINAL} of ECSE dp03 into 0.1 m spatial resolution from QF_{FINAL} of ECSE dp02. Assign the QF_{FINAL} to 1 to ECSE dp03 that fell in between two adjacent measurement levels and if one or more QF_{FINAL} of two adjacent measurement levels is equal to 1. Otherwise, spatial interpolated QF_{FINAL} values are equal to 0.
    - half-hourly summarized of the final quality flag of time rate changes at 0.1 m vertical spatial resolution of temperature are included in the basic and expanded HDF5 files: SITE/dp03/qfqm/tempStor
    - half-hourly summarized of the final quality flag of time rate changes at 0.1 m vertical spatial resolution of CO_2 concentration are included in the basic and expanded HDF5 files: SITE/dp03/qfqm/co2Stor
    - half-hourly summarized of the final quality flag of time rate changes at 0.1 m vertical spatial resolution of H_2O concentration are included in the basic and expanded HDF5 files: SITE/dp03/qfqm/h2oStor

6.3.2.4 ECSE dp04

Data flow for QA/QC processing of ECSE dp04 is as follows:

- For each half-hourly, calculated QF_{FINAL} of ECSE dp04 by assigning the QF_{FINAL} of that half-hourly to 1 if one or more QF_{FINAL} of ECSE dp03 which used to determine that ECSE dp04 is equal to 1. Otherwise, QF_{FINAL} of ECSE dp04 at that half-hourly is equal to 0.
    - half-hourly summarized of the final quality flag of storage fluxes of temperature are included in the basic and expanded HDF5 files: SITE/dp04/qfqm/fluxTemp/stor
    - half-hourly summarized of the final quality flag of storage fluxes of CO_2 are included in the basic and expanded HDF5 files: SITE/dp04/qfqm/fluxCo2/stor
    - half-hourly summarized of the final quality flag of storage fluxes of H_2O are included in the basic and expanded HDF5 files: SITE/dp04/qfqm/fluxH2o/stor

6.4 Uncertainty analysis

6.4.1 Theory of Algorithm

Similar to ECTE (Sect. 5.4.1), the random sampling error is estimated using the method of Salesky et al. (2012). However, the minimum and maximum time filter width are adjusted as recommended by

(Salesky et al. (2012). Therefore, the filtering is performed using Fast Fourier transform for 10 exponentially spaced filter widths in the range 2 $time_{\text{scal}}$ < $time_{\text{filt}}$ < $time_{\text{agr}}$ / 4.

6.4.2 Algorithmic implementation

The calculations described in this section are applied to ECSE dp01–dp04 in Table 1. The process is coded and documented in detail in the R-packages eddy4R.base and eddy4R.ucrt, consisting of the following sequence (incl. function references).

6.4.2.1 ECSE dp01

The random error calculation described in Sect. 5.4.1 and 6.4.1 is applied to all ECSE dp01 data products in Table 1. It is coded and documented in detail in the R-function eddy4R.ucrt::def.ucrt.samp.filt() and eddy4R.ucrt::wrap.neon.dp01.ucrt.ecse(). Data flow is as follows:

- Calculation is performed individually for datasets of each duration.
- Using the eddy4R.base::def.idx.agr function to determine the input datasets of 2 min and 30 min duration for **co2Stor** and **H2oStor**, 9 min and 30 min duration for **isoCo2**, 9 min and 30 min duration for **isoH2o** during sampling period and 3 min and 30 min duration for **isoH2o** during validation period.
- As default, the calculation is only performed if there are less than 10% missing values in the dataset. However, the missing values are adjusted to
  - 50% for dlta13CCo2, rtioMoleDry12CCo2, rtioMoleDry13CCo2, rtioMoleDryCo2, rtioMoleWet12CCo2, rtioMoleWet13CCo2, and rtioMoleWet13CCo2 measured by **isoCo2**
  - 85% for rtioMoleDryH2o and rtioMoleWetH2o measured by **isoCo2**
- If the missing data are less than the values as mentioned above, those are filled using linear interpolation.
- Filtering is performed using Fast Fourier transform for 10 exponentially spaced filter widths in the range of:
  - 2 s < $time_{\text{filt}}$ < 30 s for **co2Stor** and **h2oStor** during both sampling and validation period
  - 2 s < $time_{\text{filt}}$ < 135 s for **isoCo2** during both sampling and validation period
  - 2 s < $time_{\text{filt}}$ < 135 s for **isoH2o** during sampling period and 2 s < $time_{\text{filt}}$ < 45 s for **isoH2o** during sampling period
- Nonlinear least squares regression is used to fit the power law.
- The random sampling error is calculated for each target averaging periods of:
  - 2 min for **co2Stor** and **h2oStor** during both sampling and validation period
  - 9 min for **isoCo2** during both sampling and validation period
  - 9 min for **isoH2o** during sampling period and 3 min for **isoH2o** during sampling period
- The random sampling error for 30 min averaging period can be determine by:
  - First, determined how many small averaging periods falling into each 30 min window.
  - Then, calculated the random sampling error for each of small averaging period that falling into each 30 min window.

  - Lastly, calculated the median out of the random sampling error from the previous step and used that results to represent the random sampling error of that 30 min averaging period
- The uncertainty results for each reported dp01 are included in the basic and expanded HDF5 files:
  - SITE/dp01/ucrt/co2Stor
  - SITE/dp01/ucrt/h2oStor
  - SITE/dp01/ucrt/isoCo2
  - SITE/dp01/ucrt/isoH2o

7 NET SURFACE-ATMOSPHERE EXCHANGE

7.1 Theory of Measurement

7.2 Data Analysis

7.2.1 Theory of Algorithm

Here, net surface-atmosphere exchange (NSAE) is defined as the sum of storage flux and turbulent flux, on a 30 min basis, Eq. (16). The constituent terms I and II are derived, respectively, in Sect 5 and Sect. 6.

$$NSAE = \underbrace{\int_0^{d_{z,m}} \frac{\partial \overline{X}}{\partial t} dz}_{\text{I}} + \underbrace{\overline{w'X'}}_{\text{II}} \tag{16}$$

7.2.2 Algorithmic implementation

The NSAE data analysis is implemented as part of the eddy4R-Docker EC processing framework (Sect. 4). The corresponding R workflow flow.nsae.R (link to public GitHub repo in preparation) and its algorithmic sequence are summarized in Figure 10. The calculations are applied to all data products of product level "dp04" in Table 1. The process is coded and documented in detail in the R-packages eddy4R.base, consisting of the following sequence:

- calculation is performed for datasets of 30 min time resolution.
- total (wet) air density, dry air density, and latent heat of vaporization are calculated.
- ECTE and ECSE heat and water vapor fluxes are converted from kinematic units to units of energy [W m-2].
- ECTE and ECSE CO_2 fluxes are converted to units [µmolCO_2 m^{-2} s^{-1}].
- ECTE and ECSE fluxes are combined per Eq. (16) to yield NSAE fluxes.
- location of results in HDF5 file: SITE/dp04/data/FLUX/nsae, where FLUX is one of {**fluxCo2**, **fluxH2o**, **fluxTemp**}.

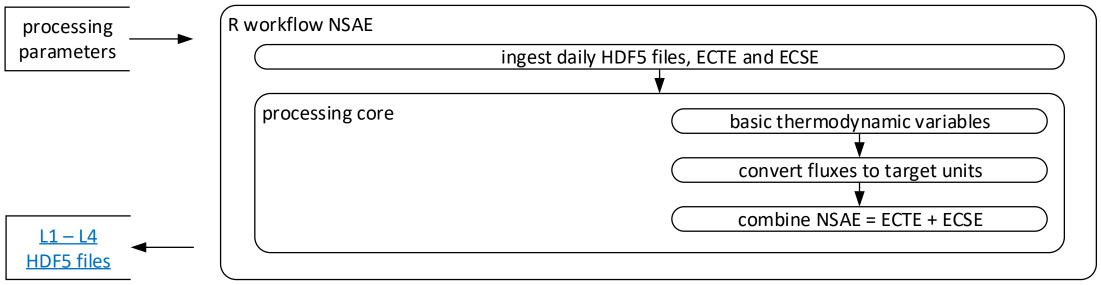

processing parameters
R workflow NSAE
ingest daily HDF5 files, ECTE and ECSE
processing core
basic thermodynamic variables
convert fluxes to target units
combine NSAE = ECTE + ECSE
L1 – L4 HDF5 files

Figure 10. The EC net surface-atmosphere exchange workflow within the eddy4R-Docker EC processing framework (Sect. 4).

7.3 Quality Assurance and Quality Control analysis

Scheduled for implementation as part of future plans and modifications (Sect. 8).

7.3.1 Theory of Algorithm

7.3.2 Algorithmic implementation

7.4 Uncertainty analysis

Scheduled for implementation as part of future plans and modifications (Sect. 8).

7.4.1 Theory of Algorithm

7.4.2 Algorithmic implementation

8 FUTURE PLANS AND MODIFICATIONS

Additional measurements can be proposed through NEON's Assignable Asset program.

At the time of writing, implementation of NEON's EC processing focusses on completing data products (Table 1) and making the DevOps framework (Sect. 4) publicly accessible for continued development. This DevOps framework intends to incentivize justified and reasonable community requests and contributions, and thus to continuously tailor NEON EC DPs and the publicly available eddy4R-Docker software to user needs. Requests and contributions are kept and moderated in a central backlog. The Surface Atmosphere Exchange Technical Working Group will be consulted for regular prioritization of scientific return-on-investment, and activation of capability development following requests. Prominent capability requests include end-to-end quality and uncertainty budgets, adding more footprint parameterizations, mapping naming conventions to other networks, and many more.

9 ACKNOWLEDGEMENTS

We thank the members of the Surface Atmosphere Exchange Technical Working Group for many years of constructive advise in creating NEON's eddy-covariance infrastructure. Special thanks for reviewing and editing this version of the Algorithm Theoretical Basis Document go to George Burba, Ankur Desai, Tarek El-Madany, Tobias Gerken, Jordan Goodrich, Eugenie Paul-Limoges, and Sebastian Wolf.

10 CITATION

Metzger, S., Durden, D., Florian, C., Luo, H., Pingintha-Durden, N., and Xu, K.: Algorithm theoretical basis document: eddy-covariance data products bundle, National Ecological Observatory Network, NEON.DOC.004571, Revision A (2018-04-30), http://data.neonscience.org/documents, Boulder, U.S.A., 51 pp., 2018.

11 BIBLIOGRAPHY

Baldocchi, D., Finnigan, J., Wilson, K., Paw U, K. T., and Falge, E.: On measuring net ecosystem carbon exchange over tall vegetation on complex terrain, Boundary Layer Meteorol., 96, 257-291, doi:10.1023/a:1002497616547, 2000.

Brock, F. V.: A nonlinear filter to remove impulse noise from meteorological data, J. Atmos. Oceanic Technol., 3, 51-58, doi:10.1175/1520-0426(1986)003<0051:anftri>2.0.co;2, 1986.

Campbell Scientific: CSAT3 three dimensional sonic anemometer instruction manual, Campbell Scientific, Logan, USA, 72, 2011.

Dyer, A. J., and Hicks, B. B.: Flux-gradient relationships in the constant flux layer, Q. J. R. Meteorolog. Soc., 96, 715-721, 10.1002/qj.49709641012, 1970.

Finkelstein, P. L., and Sims, P. F.: Sampling error in eddy correlation flux measurements, J. Geophys. Res. Atmos., 106, 3503-3509, doi:10.1029/2000JD900731, 2001.

Finnigan, J.: A comment on the paper by Lee (1998): "On micrometeorological observations of surface-air exchange over tall vegetation", Agric. For. Meteorol., 97, 55-64, doi:10.1016/s0168-1923(99)00049-0, 1999.

Finnigan, J.: The storage term in eddy flux calculations, Agric. For. Meteorol., 136, 108-113, doi:10.1016/j.agrformet.2004.12.010, 2006.

Finnigan, J. J., Clement, R., Malhi, Y., Leuning, R., and Cleugh, H. A.: A re-evaluation of long-term flux measurement techniques. Part 1: Averaging and coordinate rotation, Boundary Layer Meteorol., 107, 1-48, doi:10.1023/A:1021554900225, 2003.

Foken, T.: Micrometeorology, Springer, Berlin, Heidelberg, 306 pp., 2008.

Foken, T., Leuning, R., Oncley, S. P., Mauder, M., and Aubinet, M.: Corrections and data quality control, in: Eddy covariance: A practical guide to measurement and data analysis, edited by: Aubinet, M., Vesala, T., and Papale, D., Springer, Dordrecht, Heidelberg, London, New York, 85-131, 2012.

Foken, T.: Micrometeorology, 2 ed., Springer, Berlin, Heidelberg, 362 pp., 2017.

Hargrove, W. W., and Hoffman, F. M.: Using multivariate clustering to characterize ccoregion borders, Computing in Science and Engineering, 1, 18-25, doi:10.1109/5992.774837, 1999.

Hargrove, W. W., and Hoffman, F. M.: Potential of multivariate quantitative methods for delineation and visualization of ecoregions, Environmental Management, 34, S39-S60, doi:10.1007/s00267-003-1084-0, 2004.

Hicks, B. B.: Wind profile relationships from the 'wangara' experiment, Q. J. R. Meteorolog. Soc., 102, 535-551, doi:10.1002/qj.49710243304, 1976.

Hollinger, D. Y., and Richardson, A. D.: Uncertainty in eddy covariance measurements and its application to physiological models, Tree Physiol., 25, 873-885, doi:10.1093/treephys/25.7.873, 2005.

Kaimal, J. C., and Finnigan, J. J.: Atmospheric boundary layer flows: Their structure and measurement, Oxford University Press, New York, USA, 289 pp., 1994.

Kljun, N., Calanca, P., Rotach, M. W., and Schmid, H. P.: A simple parameterisation for flux footprint predictions, Boundary Layer Meteorol., 112, 503-523, doi:10.1023/B:BOUN.0000030653.71031.96, 2004.

Kljun, N., Calanca, P., Rotach, M. W., and Schmid, H. P.: A simple two-dimensional parameterisation for flux footprint prediction (FFP), Geosci. Model Dev., 8, 3695-3713, doi:10.5194/gmd-8-3695-2015, 2015.

Kondo, J., and Sate, T.: The Determination of the von Kármán Constant, Journal of the Meteorological Society of Japan, 60, 461-471, 1982.

Kormann, R., and Meixner, F. X.: An analytical footprint model for non-neutral stratification, Boundary Layer Meteorol., 99, 207-224, doi:10.1023/A:1018991015119, 2001.

Leclerc, M. Y., and Foken, T.: Footprints in micrometeorology and ecology, 1st ed., Springer, Berlin, Heidelberg, Germany, 239 pp., 2014.

Lee, X.: On micrometeorological observations of surface-air exchange over tall vegetation, Agric. For. Meteorol., 91, 39-49, doi:10.1016/s0168-1923(98)00071-9, 1998.

Lee, X., Finnigan, J., and Paw U, K. T.: Coordinate systems and flux bias error, in: Handbook of micrometeorology: A guide for surface flux measurement and analysis, 1 ed., edited by: Lee, X., Law, B., and Massman, W., Springer, Dordrecht, 33-66, 2004.

Lemon, E. R.: Photosynthesis under field conditions. II. An aerodynamic method for determining the turbulent carbon dioxide exchange between the atmosphere and a corn field, Agron. J., 52, 697-703, doi:10.2134/agronj1960.00021962005200120009x, 1960.

Lenschow, D. H., and Stankov, B. B.: Length scales in the convective boundary layer, Journal Of The Atmospheric Sciences, 43, 1198-1209, doi:10.1175/1520-0469(1986)043<1198:LSITCB>2.0.CO;2, 1986.

Lenschow, D. H., Mann, J., and Kristensen, L.: How long is long enough when measuring fluxes and other turbulence statistics?, J. Atmos. Oceanic Technol., 11, 661-673, doi:10.1175/1520-0426(1994)011<0661:HLILEW>2.0.CO;2, 1994.

Liu, H. P., Peters, G., and Foken, T.: New equations for sonic temperature variance and buoyancy heat flux with an omnidirectional sonic anemometer, Boundary Layer Meteorol., 100, 459-468, doi:10.1023/A:1019207031397, 2001.

Loescher, H. W., Law, B. E., Mahrt, L., Hollinger, D. Y., Campbell, J., and Wofsy, S. C.: Uncertainties in, and interpretation of, carbon flux estimates using the eddy covariance technique, J. Geophys. Res. Atmos., 111, D21S90-, 2006.

Lumley, J. L., and Panofsky, H. A.: The structure of atmospheric turbulence, Interscience Publishers, New York, 1964.

Mahrt, L., Vickers, D., Howell, J., Højstrup, J., Wilczak, J. M., Edson, J., and Hare, J.: Sea surface drag coefficients in the Risø Air Sea Experiment, J. Geophys. Res., 101, 14327-14335, doi:10.1029/96jc00748, 1996.

Mann, J., and Lenschow, D. H.: Errors in airborne flux measurements, J. Geophys. Res. Atmos., 99, 14519-14526, 1994.

Mauder, M., and Foken, T.: Documentation and instruction manual of the eddy-covariance software package TK3, Universität Bayreuth, Arbeitsergenisse Abteilung Mikrometeorologie, 46, Bayreuth, Germany, 60 pp., ISSN 1614-8924, 2011.

McMillen, R. T.: An eddy correlation technique with extended applicability to non-simple terrain, Boundary Layer Meteorol., 43, 231-245, doi:10.1007/bf00128405, 1988.

Metzger, S., Junkermann, W., Mauder, M., Beyrich, F., Butterbach-Bahl, K., Schmid, H. P., and Foken, T.: Eddy-covariance flux measurements with a weight-shift microlight aircraft, Atmos. Meas. Tech., 5, 1699-1717, doi:10.5194/amt-5-1699-2012, 2012.

Metzger, S., Burba, G., Burns, S. P., Blanken, P. D., Li, J., Luo, H., and Zulueta, R. C.: Optimization of an enclosed gas analyzer sampling system for measuring eddy covariance fluxes of H_2O and CO_2, Atmos. Meas. Tech., 9, 1341-1359, doi:10.5194/amt-9-1341-2016, 2016.

Metzger, S., Durden, D., Sturtevant, C., Luo, H., Pingintha-Durden, N., Sachs, T., Serafimovich, A., Hartmann, J., Li, J., Xu, K., and Desai, A. R.: eddy4R 0.2.0: a DevOps model for community-extensible processing and analysis of eddy-covariance data based on R, Git, Docker, and HDF5, Geosci. Model Dev., 10, 3189-3206, doi:10.5194/gmd-10-3189-2017, 2017.

Monin, A. S., and Obukhov, A. M.: Basic laws of turbulent mixing in the surface layer of the atmosphere, Tr. Akad. Nauk SSSR Geofiz. Inst., 24, 163-187; English translation by John Miller, 1959, 1954.

Monteith, J. L., and Unsworth, M. H.: Principles of environmental physics, 3 ed., Elsevier, Amsterdam, Boston, 418 pp., 2008.

Nordbo, A., and Katul, G.: A wavelet-based correction method for eddy-covariance high-frequency losses in scalar concentration measurements, Boundary Layer Meteorol., 146, 81-102, doi:10.1007/s10546-012-9759-9, 2012.

Paw U, K. T., Baldocchi, D. D., Meyers, T. P., and Wilson, K. B.: Correction of eddy-covariance measurements incorporating both advective effects and density fluxes, Boundary Layer Meteorol., 97, 487-511, doi:10.1023/a:1002786702909, 2000.

R Core Team: R: A language and environment for statistical computing, R Foundation for Statistical Computing, Vienna, Austria, 2016.

Rastetter, E. B., Williams, M., Griffin, K. L., Kwiatkowski, B. L., Tomasky, G., Potosnak, M. J., Stoy, P. C., Shaver, G. R., Stieglitz, M., Hobbie, J. E., and Kling, G. W.: Processing arctic eddy-flux data using a simple carbon-exchange model embedded in the ensemble Kalman filter, Ecological Applications, 20, 1285-1301, doi:10.1890/09-0876.1, 2010.

Rebmann, C., Kolle, O., Heinesch, B., Queck, R., Ibrom, A., and Aubinet, M.: Data acquisition and flux calculations, in: Eddy covariance: A practical guide to measurement and data analysis, edited by: Aubinet, M., Vesala, T., and Papale, D., Springer, Dordrecht, Heidelberg, London, New York, 59-83, 2012.

Salesky, S., Chamecki, M., and Dias, N.: Estimating the random error in eddy-covariance based fluxes and other turbulence statistics: The filtering method, Boundary Layer Meteorol., 144, 113-135, doi:10.1007/s10546-012-9710-0, 2012.

Schmid, H. P.: Source areas for scalars and scalar fluxes, Boundary Layer Meteorol., 67, 293-318, doi:10.1007/bf00713146, 1994.

Schotanus, P., Nieuwstadt, F. T. M., and Bruin, H. A. R.: Temperature measurement with a sonic anemometer and its application to heat and moisture fluxes, Boundary Layer Meteorol., 26, 81-93, doi:10.1007/BF00164332, 1983.

Smith, D., and Metzger, S.: Algorithm theoretical basis document: Time series automatic despiking for TIS level 1 data products, National Ecological Observatory Network, NEON.DOC.000783, Boulder, U.S.A., 15 pp., 2013.

Starkenburg, D., Metzger, S., Fochesatto, G. J., Alfieri, J. G., Gens, R., Prakash, A., and Cristóbal, J.: Assessment of de-spiking methods for turbulence data in micrometeorology, J. Atmos. Oceanic Technol., 33, 2001 - 2013, doi:10.1175/jtech-d-15-0154.1, 2016.

Stull, R. B.: An Introduction to Boundary Layer Meteorology, Kluwer Academic Publishers, Dordrecht, The Netherlands, 670 pp., 1988.

Tanner, C. B., and Thurtell, G. W.: Anemoclinometer Measurements of Reynolds Stress and Heat Transport in the Atmospheric Surface Layer, University of Wisconsin, Madison, 200, 1969.

Turnipseed, A. A., Anderson, D. E., Blanken, P. D., Baugh, W. M., and Monson, R. K.: Airflows and turbulent flux measurements in mountainous terrain: Part 1. Canopy and local effects, Agric. For. Meteorol., 119, 1-21, doi:10.1016/s0168-1923(03)00136-9, 2003.

Vesala, T., Kljun, N., Rannik, U., Rinne, J., Sogachev, A., Markkanen, T., Sabelfeld, K., Foken, T., and Leclerc, M. Y.: Flux and concentration footprint modelling: State of the art, Environ. Pollut., 152, 653-666, doi:10.1016/j.envpol.2007.06.070, 2008.

Wilczak, J. M., Oncley, S. P., and Stage, S. A.: Sonic anemometer tilt correction algorithms, Boundary Layer Meteorol., 99, 127-150, doi:10.1023/A:1018966204465, 2001.

Xu, K., Metzger, S., and Desai, A. R.: Upscaling tower-observed turbulent exchange at fine spatio-temporal resolution using environmental response functions, Agric. For. Meteorol., 232, 10-22, doi:10.1016/j.agrformet.2016.07.019, 2017.

Yuan, R., Kang, M., Park, S.-B., Hong, J., Lee, D., and Kim, J.: Expansion of the planar-fit method to estimate flux over complex terrain, Meteorol. Atmos. Phys., 110, 123-133, doi:10.1007/s00703-010-0113-9, 2011.

Appendix A NEON observatory design

The NEON observatory design is based on multivariate geographic clustering (Hargrove and Hoffman, 1999, 2004). Using national data sets for eco-climatic variables, the continental US, including Hawaii, Alaska, and Puerto were partitioned into 20 eco-climatic domains (Figure 11). These domains capture the full range of US ecological and climatic diversity as well as distinct regions of vegetation, landforms, and ecosystem, dynamics. In each domain, a core (30-year) site that represents the predominant "wildlands" ecosystem is accompanied by additional research sites designed to address specific scientific questions (e.g. land use, management, disturbance, or recovery). A detailed, interactive map is available from the NEON website.

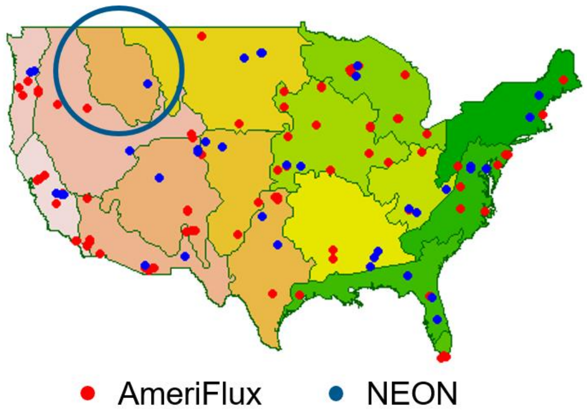


Figure 11. Eco-climatic zones across the contiguous United States after Hargrove and Hoffman (1999, 2004). Superimposed are AmeriFlux sites (prior to NEON site registration) and the NEON terrestrial instrumented site network. The NEON design adds previously underrepresented eco-climatic zones to the joint site distribution (blue circle).

Appendix B NEON site design

At each NEON TIS site, tower-based EC-flux measurements are performed in coordination with a wide range of contextual observations. These include meteorological, atmospheric composition, and soil measurements, alongside airborne remote sensing and characterization of soils, plants, insects, birds, mammals and phenology, as well as lakes and streams (Figure 12). Each NEON flux tower is placed and oriented with the design goal to represent a target ecosystem during 90% of the time, based on wind statistics from temporary deployments and source area modeling (Kormann and Meixner, 2001). The detailed spatial configuration of each site is available from the NEON website.

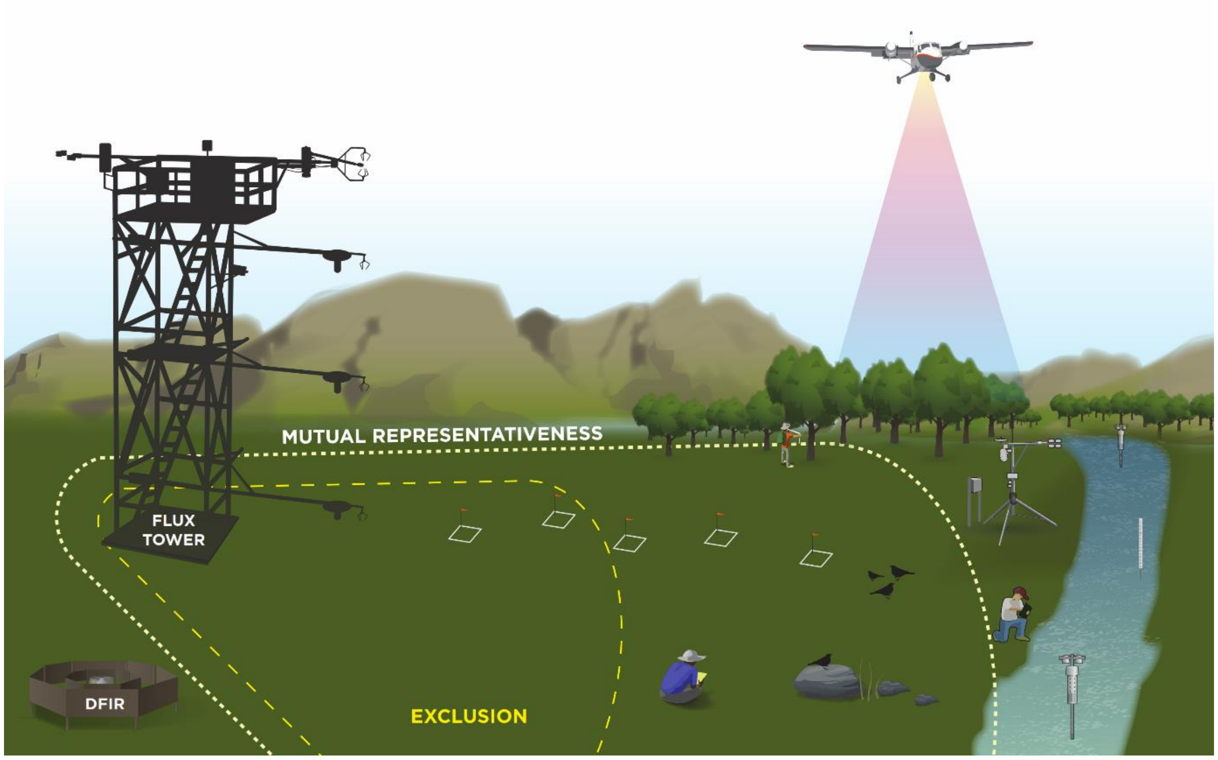
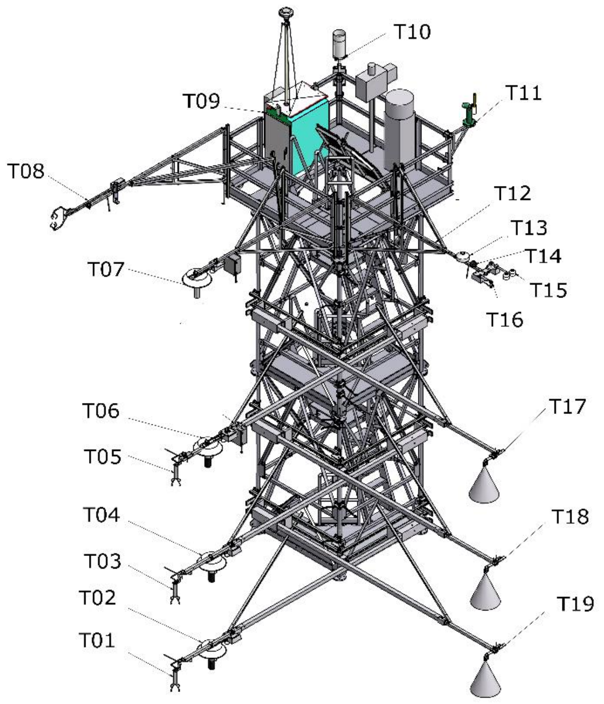


Figure 12. NEON TIS site design with instrument and observation systems covering a wide range of scales.

Appendix C NEON flux tower design

The projected base area of a NEON flux tower is 2 m x 2 m to mimic existing natural ecosystem structures and openings for most forest ecosystem found across NEON sites. Two criteria are applied to determine the height of the tower to ensure that the tower-top extends beyond the roughness sublayer: (i) A fixed tower-measurement height (h_m) of 8 m is used above all short stature ecosystems (e.g., grasslands, shrublands, or agricultural crops) when the mean canopy height is below 3 m. (ii) Over forested or more structurally complex ecosystems, the tower height is determined as $h_m \approx d + 4(h_c - d)$, where h_c is the mean canopy height and d is the zero plane displacement height (Dyer and Hicks, 1970; Hicks, 1976; Lemon, 1960; Monin and Obukhov, 1954; Monteith and Unsworth, 2008). The number of vertical measurement levels on a tower is a function of the ecosystem structure at a specific site. The number of levels varies from four to eight across NEON sites in order to capture ecological meaningful observations across vertical strata. All EC instruments are deployed on booms that extend 4 m from the tower, which is two times the face-width of the tower to reduce the impact of radiation load and flow distortion caused by the tower on the measurements (Figure 13).

The command, control and configuration of the eddy-covariance turbulent exchange (ECTE) subsystem is described in detail in AD[01]; in short: it consists of a suite of sensors that record wind speed, temperature, CO_2 and H_2O concentration on the tower top (Figure 13, location T08), which are used to calculate turbulent fluxes (Eq. (1) term IV). Wind components are measured in three dimensions by a sonic anemometer (Campbell Scientific Inc., model CSAT-3 firmware: 3.0f; Logan, Utah, USA) operating at 20 Hz. An attitude and heading reference system (Xsens North America Inc., model MTI-300-2A5G4; Culver City, California, USA) is attached to the sonic anemometer collecting data from a gyroscope, accelerometer, and magnetometer at 40 Hz to quantify and correct boom motions and allow rotated sensor deployment. H_2O and CO_2 concentration data are measured at 20 Hz by an enclosed-path infrared gas analyzer (IRGA; Li-Cor Inc., model LI-7200, firmware: 7.3.1; Lincoln, Nebraska, USA). Lastly, a validation system supplies reference gas concentrations to the IRGA for periodic validation enabling thorough uncertainty quantification.



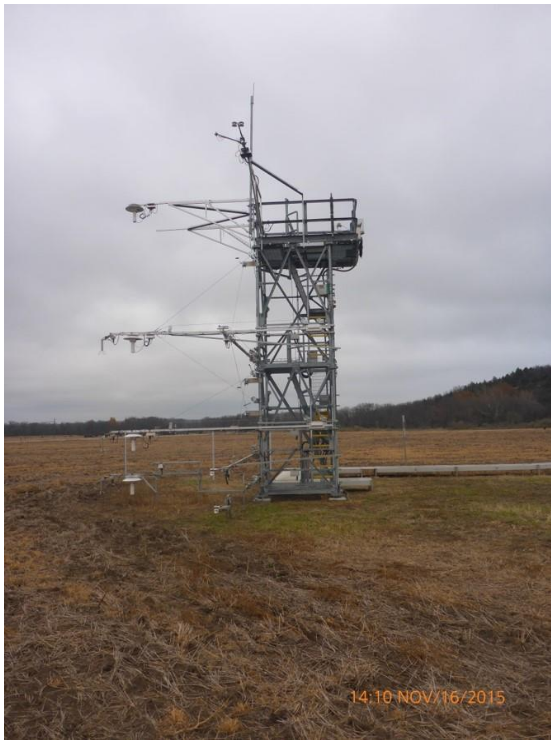


Figure 13. The NEON tower design. Left panel: conceptual design and location of individual instrument assemblies for a 4-level tower: 2D sonic anemometer (T01, T03, T05), air temperature sensor (T02, T04, T06, T07), eddy covariance boom (T08; including 3-D sonic anemometer, infrared gas analyzer, attitude and motion reference sensor), environmental enclosure (T09), secondary precipitation gauge (T10), spectral photometer (T11), radiation boom (T12), pyranometer (T13), sunshine pyranometer (T14), net radiometer (T15), up-facing and down-facing PAR sensors (T16), mid-level radiation boom (T17, T18, T19; including an up-facing PAR sensor and an infrared temperature sensor). Right panel: example of a 4-level tower at NEON CPER site.

The command, control and configuration of the eddy-covariance storage exchange (ECSE) subsystem is described in detail in AD[01]; in short: it consists of a suite of sensors that record vertical atmospheric profiles of temperature, CO_2 and H_2O concentration (Figure 13, locations T02, T04, T06, T07), which are used to calculate storage fluxes (Eq. (1) term I). The air temperature profile is measured at 1 Hz with aspirated temperature sensors (MetOne Instruments, Inc., model 076B-7388; Grant Pass, Oregon, USA). CO_2 and H_2O concentrations are measured at 1 Hz with a closed-path IRGA (Li-Cor, Inc., model LI-840A; Lincoln, Nebraska, USA). The analyzer is located in the instrument hut, and is programmed to operate in two modes, sampling and field validation. During sampling mode, the analyzer will measure air samples from different measurement levels on the tower. During field validation, the analyzer will cease measuring the air samples from the tower levels, and measure known CO_2 gas transfer standards instead. Using a similar strategy, gaseous phase stable carbon and water isotopes are measured at 1 Hz along the tower profile with cavity ring-down spectrometers (CRDS; Picarro Inc., model G2131-I and model L2130-i, firmware 1.5.0-N; Santa Clara, California, USA).

Appendix D Acronyms

Acronym	Description
AD	Applicable Documents
ATBD	Algorithm Theoretical Basis Document
C3	Command control and configuration
CI	NEON Cyberinfrastructure project team
CRDS	Cavity ring-down spectrometer
DOM	DOMAIN, e.g. D10
DP	Data product
dp00	sensor readings in engineering units; e.g. concentration as infrared absorptance
dp01	descriptive statistics
dp02	time-interpolated data
dp03	space-interpolated data
dp04	flux data
dp0p	pre-conditioned data in scientific units; e.g. concentration as mole fraction
DPL	DATA PRODUCT LEVEL, e.g. DP1
DQU	Data/qfqm/uncertainty
EC	Eddy covariance
ECSE	Eddy covariance storage exchange
ECTE	Eddy covariance turbulent exchange
HDF	Hierarchical data format
HOR	HORIZONTAL INDEX. Semi-controlled. Examples: Tower=000, HUT=700.
IRGA	Infrared gas analyzer
ISR	Inertial subrange
LST	Land surface temperature
max	Maximum
mfc	Mass flow controller
mfm	Mass flow meter
ML	Measurement level

Acronym	Description
NA	Not available/not applicable
NaN	Not a number
NEON	National Ecological Observatory Network
NK12	Nordbo and Katul (2012)
NSAE	Net surface-atmosphere exchange
PAR	Photosynthetically active radiation
PRNUM	PRODUCT NUMBER =>5 digit number. Set in data products catalog.TIS = 00000-09999
QA/QC	Quality Assurance/Quality Control
QF	Quality flag
QM	Quality metric
REV	REVISION, e.g. 001
SAE	surface atmosphere exchange
SITE	SITE, e.g. STER
SOM	Science operation management
SONIC	Ultrasonic anemometer/thermometer
TERMS	From NEON's controlled list of terms. Index is unique across products
TIS	Terrestrial Instrument System
TMI	TEMPORAL INDEX. Examples: 001=1 minute, 030=30 minute, 999=irregular intervals
VER	VERTICAL INDEX. Semi-controlled. Examples: Ground level=000, second tower level=020

Appendix E Functions

Function	Description
$\sum$	Sum operator
$\int$	Integral operator
∂	Partial differential operator
σ	Standard deviation
abs()	Absolute value
CO	Cospectrum
S	Power spectrum
$\bar{X}$	Short-term (e.g., 30 min) arithmetic mean of atmospheric quantity *X*
$\hat{X}$	Longer-term (e.g., 1 week) arithmetic mean of atmospheric quantity *X*
X'	Immediate deviation from the arithmetic mean of atmospheric quantity *X*
$\overline{X'X'}$, $\overline{X'^2}$	Short-term (e.g., 30 min) sample variance of atmospheric quantity *X*
$\widehat{X'X'}$, $\widehat{X'^2}$	Longer-term (e.g., 1 week) sample variance of atmospheric quantity *X*
$\overline{X'Y'}$	Short-term (e.g., 30 min) sample covariance of atmospheric quantities *X* and *Y*
$\widehat{X'Y'}$	Longer-term (e.g., 1 week) sample covariance of atmospheric quantities *X* and *Y*

Appendix F Parameters, variables and subscripts

Parameter subscript or variable	Description	Unit (if applicable)
0	Potential quantity, unless otherwise specified (i.e., under NIST (National Institute of Standards and Technology) standard conditions T_0 = 293.15 K, p_0 = 101.325 kPa	
1…N	Numeric identifier	
1 Hz	1 s temporal resolution	
20 Hz	0.05 s temporal resolution	
40 Hz	0.025 s temporal resolution	
d	Distance/length/height	m
$d_{\mathrm{z,m}}$	Measurement height	m
f	Measurement frequency	Hz
F	Flux into or out of an ecosystem	Depending on unit of scalar
FW_{mass}	Wet mass fraction	kg kg^{-1}
i	Running index	
l	Lag time	s
n	Normalized frequency	Dimensionless
N	Sample size	Dimensionless (count)
QF_{Cal} (qfCal)	Quality flag for the Invalid Calibration test	1 = quality test failed 0 = quality test passed -1 = NA
QF_{FINAL} (qfFinal)	Final quality flag	1 = quality test failed 0 = quality test passed -1 = NA
qfIrgaVali	IRGA validation flag, generated to indicate when the sensor is operating under a validation period	1 = validation period 0 = normal operating conditions -1 = NA
QF_{Pers} (qfPers)	Quality flag for the Persistence test	1 = quality test failed 0 = quality test passed -1 = NA
QF_{Rng} (qfRng)	Quality flag for the Range test	1 = quality test failed 0 = quality test passed -1 = NA
$QF_{sciRevw}$ (qfsciRevw)	Flag set during science operation management review	1 = quality test failed 0 = quality test passed

Parameter subscript or variable	Description	Unit (if applicable)
		-1 = NA
QF_{spec} (qfSpec)	EC specific tests (i.e. detection limit, homogeneity and stationarity, development of turbulence tests)	1 = quality test failed 0 = quality test passed -1 = NA
QF_{Step} (qfStep)	Quality flag for the Step test	1 = quality test failed 0 = quality test passed -1 = NA
QM_{α}	Alpha quality metric	%
QM_{β}	Beta quality metric	%
t	Time/duration/period	s
T	Absolute temperature	K
T_{air}	Air temperature	K
t_b	beginning time of the first minute in the 30 minute block when calculating time rate of change	minute
t_e	the last minute of the 30 minute block when calculating time rate of change	minute
T_{SONIC}	SONIC temperature measurement	K
T_{v}	Virtual temperature	K
u, v, w	Along-, cross- and vertical wind speed	m s^{-1}
x, y, z	Along-, cross- and vertical axes of a Cartesian coordinate system	Dimensionless
X, Y	Placeholder for atmospheric quantities	Depending on unit of atmospheric quantity

Appendix G eddy4R functions

eddy4R Function	Description
eddy4R.qaqc::def.dspk.br86()	Median filter de-spiking after Brock (1986), Starkenburg et al. (2014)
eddy4R.turb::PFIT_det()	Regression of the planar-fit coefficients over a moving, centered window
eddy4R.base::wrap.derv.prd.day()	Reads the list inpList in the format provided by function eddy4R.base::wrap.neon.read.hdf5.eddy(). For the list entries in inpList the following derived quantities are calculated, each through the call to a separate definition function: inpList$data$time: fractional UTC time, fractional day of year, local standard time; inpList$data$irgaTurb: average signal strength, delta signal strength, total pressure,

eddy4R Function	**Description**
	average temperature, water vapor partial pressure, water vapor saturation pressure, relative humidity, molar density of air (dry air and water vapor), molar density of dry air, wet mass fraction (specific humidity); inpList$data$soni: sonic temperature
eddy4R.base::def.lag()	Lag two datasets, so as to maximize their cross-correlation
eddy4R.base::wrap.neon.dp01()	Compute NEON Level 1 data product descriptive statistics (mean, minimum, maximum, variance, number of non-NA points) across list elements.
eddy4R.turb::REYNflux_FD_mole_dry()	Calculate turbulent vertical flux and auxiliary variables
Waves::cwt()	Morlet mother Wavelet to perform the continuous Wavelet transform of the 3-D wind components, air temperature, as well as H_2O and CO_2 concentration
eddy4R.turb::wrap.wave()	Calculate Wavelet spectrum/cospectrum using the Waves package. The frequency response correction using Wavelet techniques described in Norbo and Katul, 2012 (NK12)
eddy4R.qaqc::def.qf.irga.vali	Definition function to generate the validation flags for IRGA from the IRGA sampling mass flow controller flow rate set point or from the IRGA validation solenoid valves.
eddy4R.qaqc::def.qf.irga.agc	Definition function to generate the signal strength flags for the IRGA from the diagnostic output quality metric
eddy4R.qaqc::wrap.neon.dp01.qfqm	Pre-processing and calculating the random sampling error for the NEON eddy-covariance storage exchange (ECSE) Level 1 data products
eddy4R.ucrt:: def.ucrt.samp.filt()	Calculates the random sampling error via the Salesky et al. (2012) method. Can be used for turbulent moments of any order. If the provided value of \code{NumFilt} is too small and would fail with an error, \code{NumFilt} is incrementally increased in steps of one order of magnitude.
eddy4R.turb::def.vari.wave()	funtion to determine the temporally resolved variance/covariance from continuous wavelet transform
eddy4R.turb::footK04()	Flux footprint after Kljun et a. (2004), Metzger et al. (2012)
eddy4R.turb::def.dist.rgh()	Aerodynamic roughness length
eddy4R.qaqc::def.qf.irga.vali	Definition function to generate the validation flags for IRGA from the IRGA sampling mass flow controller flow rate set point or from the IRGA validation soleniod valves
eddy4R.qaqc::def.qf.irga.agc	Definition function to generate the signal strength flags for the IRGA from the diagnostic output quality metric
eddy4R.ucrt:: def.ucrt.samp.filt()	Calculates the random sampling error via the Salesky et al. (2012) method
eddy4R.base::wrap.neon.dp01()	Compute NEON Level 1 data product descriptive statistics (mean, minimum, maximum, variance, number of non-NA points) across list elements
eddy4R.base::def.idx.agr()	Definition function to produce a dataframe of indices and corresponding times for aggregation periods

eddy4R Function	Description
eddy4R.qaqc::def.neon.dp01.qf.grp	Grouping the quality flags of each NEON ECTE and ECSE L1 data product into a single dataframe for further use in the calculation of Alpha, Beta, and Final flag

Package 'eddy4R.base'

March 22, 2018

Title Eddy-covariance calculation for R: Base package

Version 0.2.19

Description Contains basic commonalities for the eddy4R family of R-packages

Depends R (>= 3.4.0)

Imports BiocInstaller (>= 1.26.0), downloader (>= 0.4), EMD (>= 1.5.7), ff (>= 2.2-13), ffbase (>= 0.12.3), polynom (>= 1.3-9), rhdf5 (>= 2.20.0), robfilter (>= 4.1), signal (>= 0.7-6), zoo (>= 1.8-1)

License GNU AFFERO GENERAL PUBLIC LICENSE Version 3, 19 November 2007

LazyData true

RoxygenNote 6.0.1

Author Stefan Metzger [aut, cre],
David Durden [aut],
Natchaya Pingintha-Durden [aut],
Cove Sturtevant [aut],
Ke Xu [aut]

Maintainer Stefan Metzger <eddy4R.info@gmail.com>

RemoteType local

RemoteUrl /home/smetzger/eddy/Github/NEON-FIU-algorithm/NEONScience/0-NEONScience-DEFAULT/ext/eddy4R/pack/eddy4R.base

RemoteSha 0.2.19

R topics documented:

`def.agr.ecte.dp01` *Definition function: aggregation of ecte dp01 outputs*

Description

Definition function to produce a dataframe of indices and corresponding times for aggregation periods.

Usage

```
def.agr.ecte.dp01(inpList, MethSubAgr = FALSE, MethUcrt = FALSE,
  RptExpd = FALSE)
```

Arguments

`inpList`	a list of dp01 computed output statistics and dp01 quality flags and quality metrics over multiple aggregations periods that need to be combined.
`MethSubAgr`	a logical to indicate if the subset aggregated periods should be included in the data to be combined.
`RptExpd`	A logical parameter that determines if the full quality metric `qm` is output in the returned list (defaults to FALSE).

Value

A dataframe of indices and corresponding times for aggregation periods.

Author(s)

Dave Durden `<ddurden@battelleecology.org>`

References

License: GNU AFFERO GENERAL PUBLIC LICENSE Version 3, 19 November 2007.

See Also

Currently none

Examples

```
Currently none
```

`def.agr.vari.seSq` *Definition function: Determining mean, external, internal and total variance, and squared standard error*

Description

Function defintion. Calculates the mean, external, internal and total variance, and the squared standard error over all supplied input values. Useful e.g. to aggregate from daily to monthly resolution by using all provided values, or similar.

Usage

```
def.agr.vari.seSq(data = data.frame(mean, vari))
```

Arguments

`data`	Dataframe of type numeric containing column vectors `mean` and `vari` of equal length.
`mean`	Vector of type numeric. Means of the variable of interest at finer resolution, e.g. minutely [user-defined].
`vari`	Vector of type numeric. Variances of the variable of interest at finer resolution, e.g. minutely [user-defined^2].

Value

Returns a list containing mean `mean`, external variance `variExt`, internal variance `variIntl`, total variance `variTota`, and squared standard error `seSq` at coarser resolution, e.g. hourly when `data` contains 60 minutely values.

Author(s)

Ke Xu <`xuke2012abroad@gmail.com`>
Stefan Metzger <`eddy4R.info@gmail.com`>

References

License: GNU AFFERO GENERAL PUBLIC LICENSE Version 3, 19 November 2007

See Also

Currently none

Examples

```
def.vari.seSq.agr(data = data.frame(mean = rnorm(10), vari = rnorm(10)^2))
```

`def.base.ec` *Definition function: Base state for eddy-covariance calculation*

Description

Function definition. Determine base state for eddy-covariance calculation.

Usage

```
def.base.ec(pos, var, AlgBase)
```

Arguments

`pos`	time index used for interpolation [-]
`var`	variable of interest []
`AlgBase`	c("mean", "trnd", "ord03") algorithm used to determine base state, where "mean" is the simple algorithmic mean, "trnd" is the least squares linear (1st order) trend, and "ord03" is the least squares 3rd order polynomial fit

Value

Vector that contains the chosen base-signal for `var`.

Author(s)

Stefan Metzger <`eddy4R.info@gmail.com`>
Cove Sturtevant <`eddy4R.info@gmail.com`>
Natchaya Pingintha-Durden <`ndurden@battelleecology.org`>

References

License: GNU AFFERO GENERAL PUBLIC LICENSE Version 3, 19 November 2007

See Also

Currently none

Examples

```
Currently none
```

`def.bin` *Definition function: Binning data*

Description

Function definition. Smooth data using Binning method.

Usage

```
def.bin(idep, depe, RngMinMax = NULL, NumBin, widtBin = c("lin", "log10",
  "exp10", "logExp", "expLog"), meanFunc = c("mean", "median"))
```

Arguments

`idep`	Either a vector or matrix of class numeric or integer containing the independent variable and of the same length as `depe`. []
`depe`	Either a vector or matrix of class numeric or integer containing the dependent variable and of the same length as `idep`. []
`RngMinMax`	An object of class numeric or integer containing the minimum and maximum values of the independent variable. Defaults to NULL. []
`NumBin`	An object of class numeric or integer containing the number of bins. []
`widtBin`	An object of class string containing the functions ("lin", "log10", "exp10", "logExp", "expLog") to determine bin width distribution of the independent variable. []
`meanFunc`	An object of class string containing the arithmetic "mean" and "median". []

Value

`idep` A list object of class "numeric" containing the resulted binning of independent variable and of the same length as widtBin and `depe` a matrix containing the the resulted binning of dependent variable and of the same length as widtBin.

Author(s)

Stefan Metzger <eddy4R.info@gmail.com>
Natchaya Pingintha-Durden <ndurden@battelleecology.org>

References

License: GNU AFFERO GENERAL PUBLIC LICENSE Version 3, 19 November 2007

See Also

Currently none

Examples

```
def.bin(idep = rnorm(5000), depe = rnorm(5000), RngMinMax = NULL, NumBin = 23, widtBin = "log10", meanFunc = "mean"
def.bin(idep = rnorm(500), depe = rnorm(500), RngMinMax = c(0.1, 0.4), NumBin = 12, widtBin = "lin", meanFunc = "med
```

---

def.cart.pol *Definition function: Decomposing azimuth angles to cartesian vectors*

Description

Decompose azimuth angles from polar to cartesian for angular averaging.

Usage

```
def.cart.pol(az)
```

Arguments

az	vector of type numeric, clockwise azimuth angle with 0 / 360 degree discontinuity in north [decimal degrees]

Value

matrix consisting of unit vectors X (1st column, -1 ... 1, positive to north) and Y (2nd column, -1 ... 1, positive to east)

Author(s)

Stefan Metzger <eddy4R.info@gmail.com>
Natchaya Pingintha-Durden <ndurden@battelleecology.org>

References

License: GNU AFFERO GENERAL PUBLIC LICENSE Version 3, 19 November 2007

See Also

Currently none

Examples

```
def.cart.pol(20)
```

`def.coef.corl` *Definition function: Coriolis coefficient*

Description

Function definition. Determine coriolis coefficient.

Usage

```
def.coef.corl(lat)
```

Arguments

`lat`	Required. Latitude [degrees North]

Value

Coriolis coefficient for `lat` [rad s-1].

Author(s)

Stefan Metzger <`eddy4R.info@gmail.com`>
Cove Sturtevant <`eddy4R.info@gmail.com`>

References

License: GNU AFFERO GENERAL PUBLIC LICENSE Version 3, 19 November 2007

See Also

Currently none

Examples

```
Currently none
```

`def.conv.poly` *Definition function: Apply polynomial conversion*

Description

Function definition. Apply a polynomial equation to data given the polynomial coefficients.

Usage

```
def.conv.poly(data, coefPoly = c(0, 1), MethGc = TRUE)
```

Arguments

`data`	Required. A vector or matrix of type numeric, containing the data to be converted
`coefPoly`	Optional. A vector of type numeric, containing the polynomial coefficients (in increasing order) to apply to `data`. This is a numerical vector of [a0 a1 a2 a3 ...] signifying the coeficients of the equation a0 + a1*x + a2*x^2 + a3*x^3 ... , where x is the input data. Default is c(0,1).
`MethGc`	is set to `TRUE` (default), to run garbage collection function.

Value

A vector or matrix, matching the format of `data` containing the data with polynomial conversion applied.

Author(s)

Cove Sturtevant `<eddy4R.info@gmail.com>`

References

License: GNU AFFERO GENERAL PUBLIC LICENSE Version 3, 19 November 2007

See Also

Currently none

Examples

```
Currently none
```

def.dens.mass.h2o.press.h2o.temp *Definition function: Calculate absolute humidity from water vapor pressure and ambient temperature*

Description

Calculate absolute humidity from water vapor pressure and ambient temperature.

Usage

```
def.dens.mass.h2o.press.h2o.temp(presH2o, temp)
```

Arguments

`presH2o`	Either a vector or an object of class numeric of measured water vapor pressure and of the same length as `temp`. [Pa]
`temp`	Either a vector or an object of class numeric of measured air temperature and of the same length as `presH2o`. [K]

Value

Absolute humidity and of the same length as `presH2o` and `temp`. [kg m-3]

Author(s)

Stefan Metzger <`eddy4R.info@gmail.com`>
Natchaya Pingintha-Durden <`ndurden@battelleecology.org`>

References

License: GNU AFFERO GENERAL PUBLIC LICENSE Version 3, 19 November 2007

See Also

Currently none

Examples

```
def.dens.mass.h2o.press.h2o.temp(presH2o = 2054.04, temp = 298.15)
def.dens.mass.h2o.press.h2o.temp(presH2o = c(42.22, 348.70, 2054.04), temp = c(265.15, 278.15, 298.15))
```

def.dens.mole.air — *Definition function: Calculation of the molar density of the mixture of dry air and water vapor*

Description

Function definition. Calculation of the molar density of the misxture of dry air and water vapor

Usage

```
def.dens.mole.air(presSum, tempMean)
```

Arguments

`presSum`	A vector containing the total pressure of the air mixture, of class "numeric". [Pa]
`tempMean`	A vector containing the mean temperatrue of the air mixture, of class "numeric". [K]

Value

The returned object is the the molar density of the mixture of dry air and water vapor

Author(s)

Stefan Metzger <`eddy4R.info@gmail.com`>
Hongyan Luo <`hluo@battelleecology.org`>

References

License: GNU AFFERO GENERAL PUBLIC LICENSE Version 3, 19 November 2007

See Also

Currently none.

Examples

```
Example 1, this will cause an error message due to tempIn and tempOut have no units:
def.dens.mole.air(presSum = 86000, tempMean = 286)
Example 2, assign values and units to variables first, the function should run ok.
presSum = 86000
tempMean = 286
attributes(presSum)$unit = "Pa"
attributes(tempMean)$unit = "K"
def.dens.mole.air(presSum, tempMean)
```

`def.dens.mole.air.dry` *Definition function: Calculation of the molar density of the dry air alone*

Description

Function definition. Calculation of the molar density of the mdry air alone

Usage

```
def.dens.mole.air.dry(densMoleAir, densMoleH2o)
```

Arguments

`densMoleAir`	A vector containing the mole density of the air mixture (includes dry air and water vapor), of class "numeric". [mol m-3]
`densMoleH2o`	A vector containing the water vapor mole density of the air mixture, of class "numeric". [molH2o m-3]

Value

The returned object is the the molar density of the dry air alone

Author(s)

Stefan Metzger <`eddy4R.info@gmail.com`>
Hongyan Luo <`hluo@battelleecology.org`>

References

License: GNU AFFERO GENERAL PUBLIC LICENSE Version 3, 19 November 2007

See Also

Currently none.

Examples

```
Example 1, this will cause an error message due to densMoleAir and densMoleH2o have no units:
def.dens.mole.air.dry(densMoleAir = 37.9, densMoleH2o = 0.3)
Example 2, assign values and units to variables first, the function should run ok.
densMoleAir = 37.9
densMoleH2o = 0.3
attributes(densMoleAir)$unit = "mol m-3"
attributes(densMoleH2o)$unit = "molH2o m-3"
def.dens.mole.air.dry(densMoleAir, densMoleH2o)
```

`def.dens.pres.pois` *Definition function: Poisson's equation (adiabatic change) - density as function of pressure change*

Description

Poisson's equation (adiabatic change) - density as function of pressure change.

Usage

```
def.dens.pres.pois(dens01, pres01, pres02,
  Kppa = eddy4R.base::IntlNatu$KppaDry)
```

Arguments

`dens01`	Measured density, Amount per volume [same unit as returned density, e.g. kg/m3 or kmol/m3].
`pres01`	Measured air pressure [same unit as reference pressure]
`pres02`	Reference pressure [same unit as measured air pressure]
`Kppa`	Ratio of specific gas constant to specific heat at constant pressure. Default as KppaDry [-]

Value

Densities at reference pressure [same unit as measured density]

Author(s)

Stefan Metzger <`eddy4R.info@gmail.com`>
Natchaya Pingintha-Durden <`ndurden@battelleecology.org`>

References

License: GNU AFFERO GENERAL PUBLIC LICENSE Version 3, 19 November 2007

See Also

Currently none

Examples

```
dens02 <- def.dens.pres.pois(dens01 = 1.056, pres01 = 845, pres02 = 1000, Kppa = eddy4R.base::IntlNatu$KppaDry)
```

`def.dens.temp.pois` *Definition function: Poisson's equation (adiabatic change) - density as function of temperature change*

Description

Poisson's equation (adiabatic change) - density as function of temperature change.

Usage

```
def.dens.temp.pois(dens01, temp01, temp02,
  Kppa = eddy4R.base::IntlNatu$KppaDry)
```

Arguments

`dens01`	Measured density, Amount per volume [same unit as returned density, e.g. kg/m3 or kmol/m3].
`temp01`	Measured air temperature [K]
`temp02`	Reference temperature [K]
`Kppa`	Ratio of specific gas constant to specific heat at constant pressure. Default as KppaDry [-]

Value

Densities at reference temperature [same unit as measured density]

Author(s)

Stefan Metzger <`eddy4R.info@gmail.com`>
Natchaya Pingintha-Durden <`ndurden@battelleecology.org`>

References

License: GNU AFFERO GENERAL PUBLIC LICENSE Version 3, 19 November 2007

See Also

Currently none

Examples

```
dens02 <- def.dens.temp.pois(dens01 = 1.056, temp01 = 298.15, temp02 = 288.15, Kppa = eddy4R.base::IntlNatu$KppaDry
```

def.dens.temp.pres.pois

Definition function: Poisson's equation (adiabatic change) - density as function of pressure and temperature change

Description

Poisson's equation (adiabatic change) - density as function of pressure and temperature change.

Usage

```
def.dens.temp.pres.pois(dens01, temp01, temp02, pres01, pres02)
```

Arguments

`dens01`	Measured density, Amount per volume [same unit as returned density, e.g. kg/m3 or kmol/m3].
`pres01`	Measured air pressure [same unit as reference pressure]
`pres02`	Reference pressure [same unit as measured air pressure]
`temp01`	Measured air temperature [K]
`temp02`	Reference temperature [K]

Value

Densities at reference pressure and temperature [same unit as measured density]

Author(s)

Stefan Metzger <eddy4R.info@gmail.com>
Natchaya Pingintha-Durden <ndurden@battelleecology.org>

References

License: GNU AFFERO GENERAL PUBLIC LICENSE Version 3, 19 November 2007

See Also

Currently none

Examples

```
dens02 <- def.dens.temp.pres.pois(dens01 = 1.056, temp01 = 298.15,temp02 = 288.15, pres01 = 845, pres02 = 1000)
```

def.dir.wind *Definition function: Wind direction mean and variance calculation using vector averaging*

Description

Function defintion. Calculate the mean and variance of the wind direction.

Usage

```
def.dir.wind(inp, MethVari = c("02StepRad", "DistAngMin", "Yama")[3])
```

Arguments

`inp`	numeric vector containing instantaneous wind directions
`MethVari`	This is a character string to determine the method to calculate the wind direction variance. The methods include "02StepRad", "DistAngMin", and "Yama"

Value

Mean and variance of the wind direction in meteorological coordinate system [rad]

Author(s)

David Durden <ddurden@battelleecology.org>

References

License: GNU AFFERO GENERAL PUBLIC LICENSE Version 3, 19 November 2007.
Yamartino, R. J. (1984) A Comparison of Several "Single-Pass" Estimators of the Standard Deviation of Wind Direction. Journal of Applied Meteorology and Climatology, 23, 1362-1366.

See Also

Currently none

Examples

```
#Instantaneous wind direction
windDirInst <- rnorm(3600, 180, 10)
def.dir.wind(windDirInst, MethVari = "DistAngMin")
```

def.dld.zip — *Definition function: Download and extract .zip archives from a web address*

Description

Function defintion. Downloads .zip archives from a web `Url`, and saves and extracts the archive into the directory `Dir`.

Usage

```
def.dld.zip(Inp = list(Url, Dir = tempdir()))
```

Arguments

`Inp`	The input parameter list containing `Url` and `Dir`.
`Url`	The internet address, of class "character". For use with Dropbox, Google Drive, OneNote... the direct download link needs to be used. For example in case of Dropbox the web address needs to end on ...dl=1.
`Dir`	The target directory, of class "character".

Value

The function saves the extracted .zip archive into the directory `Dir`. Currently no values are returned to the environment it is called from.

Author(s)

Stefan Metzger <`eddy4R.info@gmail.com`>

References

License: GNU AFFERO GENERAL PUBLIC LICENSE Version 3, 19 November 2007.
`http://hydroecology.net/downloading-extracting-and-reading-files-in-r/`.

See Also

Currently none.

Examples

```
def.dld.zip(Inp = list(
Url = "https://www.dropbox.com/s/1yv1ais1wdvoq1l/outRefe_20160410.zip?dl=1",
Dir = tempdir()))
```

def.env.glob *Definition function: Global R-environment settings for use with the eddy4R family of R-packages*

Description

Function defintion. To avoid user-location-specific dependencies, the system locale is set to the C language standard. For R in Windows and Mac the 'times' font is defined. Under Unix OS, it is not neccesary to assign font "times".

Usage

```
def.env.glob()
```

Arguments

`Currently`	none

Value

Sys.setlocale('LC_ALL','C')
if(.Platform$OS.type == "windows") windowsFonts(times=windowsFont("TT Times New Roman"))
if(Sys.info()["sysname"] == "Darwin") quartzFonts(times = quartzFont(rep("Times-Roman", 4)))

Author(s)

Stefan Metzger <`eddy4R.info@gmail.com`>

References

License: GNU AFFERO GENERAL PUBLIC LICENSE Version 3, 19 November 2007.
`http://www.inside-r.org/r-doc/base/Sys.setlocale`

See Also

Currently none

Examples

```
Currently none
```

`def.extr.hdf5` *Definition function: Extract data and attributes from HDF5 input file to another HDF5 output file*

Description

Definition function. Function extracts group sturcture, data, and metadata attributes from input, examples include ecte (turbulent/turb) and ecse (storage/stor) HDF5 files to a another HDF5 file, used for nsae (net surface atmosphere exchange), with the same group heirarchy structure from each file. Either `FileIn` or `rpt` must be specified for `eddy4R.base::def.extr.hdf5()` to work. If `FileIn` is specified, the contents of the specified file are read (and optionally written to an output file). If `rpt` is specified, its contents are being written to an output file. This enables to read hdf5 files (call `eddy4R.base::def.extr.hdf5()` and specify `FileIn`), then modify the resulting `rpt` object as needed, and lastly to write the modified `rpt` object to file (call `eddy4R.base::def.extr.hdf5()` and specify `rpt`).

Usage

```
def.extr.hdf5(FileIn = NULL, rpt = NULL, FileOut = NULL,
  MethExtrData = TRUE, MethExtrAttr = TRUE, dp01 = NULL,
  MethDp0p = FALSE)
```

Arguments

`FileIn`	is the input HDF5 file (turb or stor) the data and metadata are being read from. It is ignored if `rpt` is specified.
`rpt`	is the list returned from a previous call of `eddy4R.base::def.extr.hdf5()` with `FileIn` specified. At a minimum the entries `rpt$listGrpName`, `rpt$listData` and `rpt$listAttr` are required to write the contents to an output hdf5 file.
`FileOut`	is the output file nsae HDF5 written to.
`MethExtrData`	logical parameter that decides if data from the input file should be extracted and written to the output file.
`MethExtrAttr`	logical parameter that decides if attributes (metadata) from the input file should be extracted and written to the output file.
`dp01`	A vector of class "character" containing the name of NEON dp01 which data from the input file are not extracted and written to the output file
`MethDp0p`	A logical stating if NEON dp0p data from the input file are not extracted and written to the output file

Value

A NEON formatted HDF5 file that has parameters from the input file written to the output HDF5 file.

Author(s)

David Durden <eddy4R.info@gmail.com> Stefan Metzger <eddy4R.info@gmail.com>

References

License: GNU AFFERO GENERAL PUBLIC LICENSE Version 3, 19 November 2007.
NEON Algorithm Theoretical Basis Document:Eddy Covariance Turbulent Exchange Subsystem Level 0 to Level 0' data product conversions and calculations (NEON.DOC.000823)
NEON Algorithm Theoretical Basis Document:Eddy Covariance Turbulent Exchange Subsystem Level 1 data product calculations (NEON.DOC.000807)

See Also

Currently none

Examples

```
Currently none
```

---

def.hdf5.crte *Definition function: Create the ECTE HDF5 file structure*

Description

Definition function. Function creates the standard NEON HDF5 file structure for the ECTE data products.

Usage

```
def.hdf5.crte(Date, Site = "SERC", LevlTowr, DirOut, FileOutBase = NULL,
  MethExpd = TRUE, MethDp04 = FALSE, FileNameReadMe = NULL,
  FileNameObjDesc = NULL)
```

Arguments

`Date`	is the date for the output file being generated.
`Site`	is the site for which the output file is being generated.
`LevlTowr`	is the measurement level of the tower top to determine the VER number of the NEON DP naming convention.
`DirOut`	is the output directory where the file being generated is stored.
`FileOutBase`	character string indicating the base output filename, to which the date and package information will be appended.
`MethExpd`	logical indicating if the output should be expanded or basic.
`MethDp04`	logical indicating if ECTE dp04 HDF5 folder structure should be included.

`FileNameReadMe`	character indicating the filename incl. absolute path to the ReadMe file for inclusion in the output HDF5 file. Defaults to `NULL`, which downloads the readme file from a web location.
`FileNameObjDesc`	character indicating the filename incl. absolute path to the object description file for inclusion in the output HDF5 file. Defaults to `NULL`, which downloads the object description file from a web location.

Value

A NEON formatted HDF5 file that is output to /codeDirOut with a readme and object description included.

Author(s)

David Durden <`ddurden@battelleecology.org`>
Stefan Metzger <`eddy4R.info@gmail.com`>

References

License: GNU AFFERO GENERAL PUBLIC LICENSE Version 3, 19 November 2007.
NEON Algorithm Theoretical Basis Document:Eddy Covariance Turbulent Exchange Subsystem Level 0 to Level 0' data product conversions and calculations (NEON.DOC.000807)

See Also

Currently none

Examples

```
#Example run to create L0 gold files for 4/24/2016 at SERC
#Setting Site
Site <- "SERC"
LevlTowr <- "000_060"
FileOutBase <- "NEON.D02.SERC.DP4.00200.001.ec-flux"
MethExpd <- TRUE
#Setting Date to be processed
Date <- "2016-04-24"
#Set directory for example output to working directory
DirOut <- getwd()
#Running example
def.hdf5.crte(Date = Date, Site = Site, LevlTowr = LevlTowr, DirOut = DirOut, FileOutBase = FileOutBase, MethExpd =
```

def.hdf5.dp.pack *Definition function: to package NEON eddy-covariance data product outputs to be written to HDF5 files*

Description

Definition function to produce a list of dataframes corresponding times for aggregation periods for NEON eddy-covariance data product, and are packaged to be written to the HDF5 file.

Usage

```
def.hdf5.dp.pack(inpList, MethMeas = c("ecte", "ecse")[1], time, Dp)
```

Arguments

`inpList`	A list of NEON eddy-covariance computed output statistics or quality flags and quality metrics over multiple aggregations periods that need to be combined and formatted for output to HDF5.
`MethMeas`	A vector of class "character" containing the name of measurement method (eddy-covariance turbulent exchange or storage exchange), MethMeas = c("ecte", "ecse"). Defaults to "ecte".
`time`	Optional. If MethMeas = "ecte", a dataframe including the timeBgn and timeEnd for the aggregated periods should be included in the data to be combined.
`Dp`	If MethMeas = "ecte", Dp is which data product is being packaged to be written to the HDF5 file. If MethMeas = "ecse", Dp is which level of data product is being packaged to be written to the HDF5 file, Dp = c("Dp01", "Dp02", "Dp03", "Dp04").

Value

A list of dataframes of for aggregation periods.

Author(s)

Natchaya Pingintha-Durden <ndurden@battelleecology.org>
Dave Durden <ddurden@battelleecology.org>

References

License: GNU AFFERO GENERAL PUBLIC LICENSE Version 3, 19 November 2007.

See Also

Currently none

Examples

```
Currently none
```

---

def.hdf5.dp01.pack *Definition function: to package dp01 outputs to be written to HDF5 files*

Description

Definition function to produce a list of dataframes corresponding times for aggregation periods for dp01, and are packaged to be written to the HDF5 file.

Usage

```
def.hdf5.dp01.pack(inpList, time, Dp01)
```

Arguments

`inpList`	a list of dp01 computed output statistics or dp01 quality flags and quality metrics over multiple aggregations periods that need to be combined and formatted for output to HDF5.
`time`	a dataframe including the timeBgn and timeEnd for the aggregated periods should be included in the data to be combined.
`Dp01`	which data product is being packaged to be written to the HDF5 file.

Value

A list of dataframes of for aggregation periods.

Author(s)

Dave Durden <`ddurden@battelleecology.org`>

References

License: GNU AFFERO GENERAL PUBLIC LICENSE Version 3, 19 November 2007.

See Also

Currently none

Examples

```
Currently none
```

def.hdf5.wrte.dp01 *Definition function: Write NEON Level 1 data, qfqm, and uncertainty to output HDF5*

Description

Definition function. To write NEON Level 1 data product descriptive statistics (mean, minimum, maximum, variance, number of non-NA points), quality flags and quality metrics, and uncertainty quantification to an output HDF5 file.

Usage

```
def.hdf5.wrte.dp01(inpList, FileOut, SiteLoca, LevlTowr, Dp01,
  MethUcrt = TRUE, MethSubAgr = TRUE)
```

Arguments

`inpList`	A list of including dp01 data, quality flags and quality metrics, and uncertainty calculations to package and write to an output HDF5 file.
`FileOut`	Character: The file name for the output HDF5 file
`SiteLoca`	Character: Site location.
`LevlTowr`	Character: The tower level that the sensor data is being collected in NEON data product convention (HOR_VER).
`MethUcrt`	Logical: Determines if uncertainty information is available for output.
`MethSubAgr`	Logical: Determines if 1-minute data is available for output.

Value

An HDF5 file with dp01 data, qfqm, and uncertainty written

Author(s)

David Durden <`ddurden@battelleecology.org`>

References

License: GNU AFFERO GENERAL PUBLIC LICENSE Version 3, 19 November 2007.

See Also

Currently none.

Examples

```
Currently none.
```

def.hdf5.wrte.dp01.api

Definition function: Gather/Write reingested NEON Level 1 data, qfqm, and uncertainty to output HDF5

Description

Definition function. To write NEON Level 1 data product descriptive statistics (mean, minimum, maximum, variance, number of non-NA points), quality flags and quality metrics, and uncertainty values gathered from via the API to an output HDF5 file.

Usage

```
def.hdf5.wrte.dp01.api(date, FileOut, SiteLoca, DpName, LevlTowr, TimeAgr)
```

Arguments

`date`	Character: The date for the data to be gathered.
`FileOut`	Character: The file name for the output HDF5 file
`SiteLoca`	Character: Site location.
`DpName`	Character: The data product name for the data to be gathered.
`LevlTowr`	Character: The tower level that the sensor data is being collected in NEON data product convention (HOR_VER).
`TimeAgr`	Integer: The time aggregation index in minutes (i.e. 30)

Value

An updated dp0p HDF5 file with dp01 data, qfqm, and uncertainty written

Author(s)

David Durden <`ddurden@battelleecology.org`>

References

License: GNU AFFERO GENERAL PUBLIC LICENSE Version 3, 19 November 2007.

See Also

Currently none.

Examples

```
Currently none.
```

def.idx.agr *Definition function: indices for aggregation periods*

Description

Definition function to produce a dataframe of indices and corresponding times for aggregation periods.

Usage

```
def.idx.agr(time, PrdAgr, FreqLoca, MethIdx = c("rglr", "specBgn",
  "specEnd")[1], crdH2oVali = FALSE, data = NULL, CritTime = 0)
```

Arguments

`time`	a vector of timestamps
`PrdAgr`	the time period to aggregate to averaging in seconds (30 min = 1800 s) [s]
`FreqLoca`	the frequency of the measurements that are being aggregated in hertz [Hz]
`MethIdx`	a vector of class "character" containing the name of method used to determine the beginning and ending indicies. MethIdx = c("rglr", "specBgn", "specEnd"), where "rglr" is the regular method, e.g. for FreqLoca = 1 and PrdAgr = 1800, the first result of idxBgn and idxEnd are 1 and 1800, respectively. "specBgn" is the specific method to determine the beginning and ending indicies using the first indency when data is available. "specEnd" is the specific method to determine the beginning and ending indicies using the last indency when data is available. Defaults to "rglr". [-]
`crdH2oVali`	a logical parameter indicating indices are being produced for a crdH2o during validation period. (defaults to FALSE).
`data`	a vector of input data which will be used to determine when data is available when "specBgn" or "specEnd" is selected. Defaults to NULL. [User-defined]
`CritTime`	the critcal time to include before determine the beginning and ending indicies, e.g. CritTime = 60 for aggregation only the last 2 min from 3 min measurement time. Defaults to 0. [s]

Value

A dataframe of indices and corresponding times for aggregation periods.

Author(s)

Dave Durden <`ddurden@battelleecology.org`>
Natchaya Pingintha-Durden <`ndurden@battelleecology.org`>

References

See Also

Currently none

Examples

```
FreqLoca <- 20
timeMeas <- base::as.POSIXlt(seq.POSIXt(
  from = base::as.POSIXlt("2016-01-01 00:00:00.001", format="%Y-%m-%d %H:%M:%OS", tz="UTC"),
  to = base::as.POSIXlt("2016-01-01 04:59:59.952", format="%Y-%m-%d %H:%M:%OS", tz="UTC"),
   by = 1/FreqLoca), tz = "UTC")
PrdAgr <- 1800
def.idx.agr(time = timeMeas, PrdAgr = PrdAgr, FreqLoca = FreqLoca)
```

`def.idx.diff` *Definition function: indices for difference calculation, e.g. time rate of change of temperature*

Description

Definition function to produce a dataframe of indices and corresponding times for difference calculation.

Usage

```
def.idx.diff(PrdWndwAgr, PrdIncrAgr, numDate)
```

Arguments

`PrdWndwAgr`	the width of time period to aggregate to averaging in seconds (4 min = 240 s) [s]
`PrdIncrAgr`	the incremental time period to calculate the difference in seconds (30 min = 1800 s) [s]
`numDate`	the number of days in the dataset [-]

Value

A dataframe of indices for difference calculation.

Author(s)

Ke Xu <kxu@battelleecology.org>

References

See Also

Currently none

Examples

```
PrdWndwAgr <- 4 * 60
PrdIncrAgr <- resoTimeDp02[[idxDp]] * 60
numDate <- 1
wrk$whrData <- eddy4R.base::def.idx.diff(
 PrdWndwAgr=PrdWndwAgr,
 PrdIncrAgr=PrdIncrAgr,
 numDate=numDate
)
```

---

def.inst.depe *Definition function: Install dependent packages*

Description

Function defintion. Installs dependent packages from a source package DESCRIPTION file, without installing the source package itself. Useful e.g. for re-packing source packages with Roxygen, because devtools::document() requires all dependent packages to be installed. In case during development new dependent packages have been added to the DESCRIPTION file but are not yet installed, devtools::document() stops with an error. Calling eddy4R.base::def.inst.depe() prior to devtools::document() mitigates this error.

Usage

```
def.inst.depe(DirPack = ".", Depe = TRUE,
  Repo = base::getOption("repos")[1], Lib = .libPaths()[1])
```

Arguments

DirPack	Path to the package DESCRIPTION file, defaults to the current working directory. Character vector with single entry [-].
Depe	Defines what type of dependencies are installed. Defaults to TRUE which includes all of c("Depends", "Imports", "LinkingTo"). c("Depends", "Imports", "LinkingTo") can also be specified individually. Logical vector with single entry or character vector [-].
Repo	The base URL(s) of the repositories to use, e.g., the URL of a CRAN mirror such as "https://cloud.r-project.org". Defaults to base::getOption("repos")[1]. Character vector [-].
Lib	Path to the libraries in which the dependent packages are installed. Character vector [-].

Author(s)

Stefan Metzger <eddy4R.info@gmail.com>

References

License: GNU AFFERO GENERAL PUBLIC LICENSE Version 3, 19 November 2007.
https://gist.github.com/jtilly/ac1b028f3666ce1d82c2.

See Also

Currently none.

Examples

```
eddy4R.base::def.inst.depe()
```

`def.lag` *Definition function: Lag two datasets, so as to maximise their cross-correlation*

Description

Function definition. Lag two datasets, so as to maximise their cross-correlation.

Usage

```
def.lag(refe, meas, freq, dataRefe = refe, dataMeas = meas,
  measVar = NULL, lagMax = 2 * freq, lagCnst = TRUE,
  lagNgtvPstv = c("n", "p", "np")[3], lagAll = TRUE, hpf = TRUE,
  fracMin = 0.1)
```

Arguments

`refe`	A vector with variable in reference time frame. Of class numeric. [-]
`meas`	A vector with variable in time frame to be adjusted. Of class numeric. [-]
`freq`	Acquisition frequency of refe and meas. Of class ingeter. [Hz]
`dataRefe`	A matrix or data.frame with all data that carries the time frame of refe. Defaults to refe. Of any class. [-]
`dataMeas`	A matrix or data.frame with all data that carries the time frame of meas. Defaults to meas. Of any class. [-]
`measVar`	A vector specifying if only several columns in dataMeas shall be lagged. Defaults to NULL. Of class integer or character. [-]
`lagMax`	Maximum lag, by default 2 x freq. Of class integer. [-]
`lagCnst`	TRUE - interpret lagMax as maximum permissible lag; FALSE - start with lagMax as first estimate and increase iteratively. Defaults to TRUE. Of class logical. [-]

`lagNgtvPstv`	"n" - consider negative lag times only, i.e. meas is expected to lag behind refe; "p" - consider positive lag times only, i.e. refe is expected to lag behind meas; "np" - consider negative and positive lag times. Defaults to "np". Of class character. [-]
`lagAll`	TRUE - consider positive and negative correlations when finding lag time; FALSE - consider positive correlations only when finding lag time. Defaults to TRUE. Of class logical. [-]
`hpf`	TRUE - apply Butterworth high-pass filter; FALSE - use raw data. Defaults to TRUE. Of class logical. [-]
`fracMin`	Minimum fraction of data to attempt lag determination. Defaults to 0.1. Of class numeric. [-]

Value

Lagged input data and calculation results in a list consisting of:
`dataRefe` The reference data. `dataMeas` The data that was lagged to coincide with the reference data. `lag` The number of data points by which the lag correction was performed. `corrCros` The cross-correction coefficient between refe and meas for the determined lag time.

Author(s)

Stefan Metzger <`eddy4R.info@gmail.com`>

References

License: GNU AFFERO GENERAL PUBLIC LICENSE Version 3, 19 November 2007.

See Also

Currently none.

Examples

```
Currently none.
```

`def.mean.med.mode` *Definition function: Calculate descriptive statistics based on arithmetic mean, median and mode*

Description

Calculate descriptive statistics based on arithmetic mean, median and mode.

Usage

```
def.mean.med.mode(test, refe = 0, Perc = FALSE)
```

Arguments

`test`	A vector containing the test data. Of class "numeric" or "integer", and of the same length as `refe`. [same units as reference data]
`refe`	A vector containing the reference data. Of class "numeric" or "integer". Defaults to `refe = 0`. [same units as test data]
`Perc`	Report results in the same units as `refe` and `test` (`Perc = FALSE`) or as percentages (`Perc = TRUE`)? Of class "logical", defaults to `Perc = FALSE`.

Value

`statLoc` A list object of class "numeric" [1, 1:3] containing mean, median, and mode.

`statDis` A list object of class "numeric" [1, 1:3] containing standard deviation, median absolute deviation, and mode absolute deviation.

`NumSamp` Sample size which not included NaNs.

Author(s)

Stefan Metzger <`eddy4R.info@gmail.com`>
Natchaya Pingintha-Durden <`ndurden@battelleecology.org`>

References

License: GNU AFFERO GENERAL PUBLIC LICENSE Version 3, 19 November 2007
Croux, C., and Rousseeuw, P. J.: Time-efficient algorithms for two highly robust estimators of scale, Computational Statistics, 1, 411-428, 1992.
`http://stackoverflow.com/questions/2547402/standard-library-function-in-r-for-finding-the-mode`

See Also

`def.mode`

Examples

```
#works
def.mean.med.mode(test = rnorm(100))

#returns error message
def.mean.med.mode(test = rnorm(10), refe = 0, Perc = TRUE)
```

`def.med.mad` *Definition function: Median and median absolute deviation as robust measures of scale and dispersion*

Description

Function defintion. Calculates the median as robust measure of scale, and the sample-size corrected median absolute deviation as robust measure of dispersion. By default, the univariate statistics of the test data `test` are returned. If reference data `refe` is provided, the bivariate statistics of the residuals `test - refe` are returned.

Usage

```
def.med.mad(test, refe = 0, Perc = FALSE)
```

Arguments

`test`	A vector containing the test data. Of class "numeric" or "integer", and of the same length as `refe`. [same units as reference data]
`refe`	A vector containing the reference data. Of class "numeric" or "integer", and of the same length as `test`. Defaults to `refe = 0`. [same units as test data]
`Perc`	Report results in the same units as `refe` and `test` (`Perc = FALSE`) or as percentages (`Perc = TRUE`)? Of class "logical", defaults to `Perc = FALSE`.

Value

Returns object of class "numeric" [1, 1:3] containing median, median absolute deviation, and sample size (NaNs not included).

Author(s)

Stefan Metzger <`eddy4R.info@gmail.com`>

References

License: GNU AFFERO GENERAL PUBLIC LICENSE Version 3, 19 November 2007.
Croux, C., and Rousseeuw, P. J.: Time-efficient algorithms for two highly robust estimators of scale, Computational Statistics, 1, 411-428, 1992.

See Also

Function eddy4R.base:::def.rmsd.bias.prec.cdet() for Gaussian statistics-based measures of scale and dispersion.

Examples

```
#works
def.med.mad(test = rnorm(500))
def.med.mad(test = rnorm(10), refe = rnorm(10))

#returns error message
def.med.mad(test = rnorm(10), refe = rnorm(5000))
```

`def.met.body` *Definition function: Coordinate transformation from CSAT3 body coordinate system to meteorological coordinate system*

Description

Function defintion. Coordinate transformation from CSAT3 body coordinate system to meteorological coordiante system.

Usage

```
def.met.body(AngZaxsSoniInst,
  AngZaxsSoniOfst = eddy4R.base::def.unit.conv(data = 90, unitFrom = "deg",
  unitTo = "rad"), veloBody)
```

Arguments

`AngZaxsSoniInst`	Parameter of class numeric. Azimuth (angle around z axis) direction against true north in which sonic anemometer installation (transducer array) is pointing [rad]
`AngZaxsSoniOfst`	Parameter of class integer or numeric. Azimuth Offset of meteorological x-axis against true north. That is, angle by which sonic data has to be clockwise azimuth-rotated when sonic anemometer body x-axis points perfectly north (az-Sonic = 0) [rad]
`veloBody`	Variable of class numeric. Data frame containing wind speeds along x-axis (veloXaxs), y-axis (veloYaxs),and z-axis (veloZaxs) in sonic anemometer body coordinate system. For example: `veloBody <- data.frame(veloXaxs=rnorm(20), veloYaxs=rnorm(` [m s-1]

Value

Wind speed in meteorological coordinate system [m s-1]

Author(s)

Stefan Metzger <`eddy4R.info@gmail.com`>
Hongyan Luo <`hluo@battelleecology.org`>
David Durden <`ddurden@battelleecology.org`>

References

License: GNU AFFERO GENERAL PUBLIC LICENSE Version 3, 19 November 2007.

See Also

Currently none

Examples

```
veloIn01 <- data.frame(veloXaxs=rnorm(20), veloYaxs=rnorm(20), veloZaxs=rnorm(20))
def.met.body(AzSoniInst=60, veloBody=veloIn01)
```

`def.mode` *Definition function: Calculate mode based on a continuous distribution*

Description

Calculate mode based on a continuous distribution.

Usage

```
def.mode(x)
```

Arguments

`x`	Either a vector or an object of class numeric of the data from which the estimate is to be computed.

Value

The estimated arithmetic mode of the values in `x`.

Author(s)

Stefan Metzger <eddy4R.info@gmail.com>
Natchaya Pingintha-Durden <ndurden@battelleecology.org>

References

License: GNU AFFERO GENERAL PUBLIC LICENSE Version 3, 19 November 2007
http://stackoverflow.com/questions/2547402/standard-library-function-in-r-for-finding-the-mode

See Also

Currently none

Examples

```
def.mode(x = rnorm(100))
```

`def.mtch.out.refe` *Definition function: Compare output against reference*

Description

Function definition. A function to compare the file output from a workflow to a reference output file to ensure that changes during development to functions called by the workflow have not impacted the results.
The function reads in the new data produced in the tmp directory and compares the first 10 lines to the reference output data.

Usage

```
def.mtch.out.refe(fileOut, fileRefe, Head = FALSE, NumLine = 10)
```

Arguments

`fileOut`	String. The relative or absolute path and filename of the numerical data output.
`fileRefe`	String. The relative or absolute path and filename of the numerical reference data.
`Head`	Logical. TRUE if there is 1 header row in both the output and reference files. Defaults to FALSE.
`NumLine`	The number of output rows to compare (starting from the first row). Defaults to 10.

Value

Currently none

Author(s)

David Durden <`ddurden@battelleecology.org`>
Cove Sturtevant <`eddy4R.info@gmail.com`>

References

License: GNU AFFERO GENERAL PUBLIC LICENSE Version 3, 19 November 2007.

See Also

Currently none

Examples

```
Currently none
```

`def.neon.dp01` *Definition function: Create NEON Level 1 data product descriptive statistics*

Description

Function definition. Compute NEON Level 1 data product descriptive statistics (mean, minimum, maximum, variance, number of non-NA points) by aggregating the input data over its entire range.

Usage

```
def.neon.dp01(data, vrbs = FALSE)
```

Arguments

`data`	A numeric vector or data.frame containing the L0p (calibrated) input data at native resolution. Of class numeric". [-]
`vrbs`	Verbose output, one of either TRUE of FALSE, defaults to FALSE. If `vrbs = FALSE` is selected, the returned object is a data.frame. If `vrbs = TRUE` is selected, the returned object is a list supporting the propagation of unit attributes for individual variables in `data`. Of class logical". [-]

Value

Descriptive statistics, for `vrbs = FALSE` a data frame and for `vrbs = TRUE` a list:
`mean` The mean of non-NA values in `data` `min` The minimum value of non-NA values in `data` `max` The maximum value of non-NA values in `data` `vari` The variance of non-NA values in `data` `num` The number of non-NA values in `data` `se` The standard error of the mean of non-NA values in `data`

Author(s)

Stefan Metzger <`eddy4R.info@gmail.com`>
Cove Sturtevant <`eddy4R.info@gmail.com`>

References

License: GNU AFFERO GENERAL PUBLIC LICENSE Version 3, 19 November 2007.
NEON.DOC.003311 - NEON DATA PRODUCTS DEVELOPMENT PLAN

See Also

Currently none.

Examples

```
# argument vrbs changes format of reported object

   # Calibrated raw data
     data <- c(1,2,3,NA,5,6,7,NaN,9,10)

   # Level 1 descriptive statistics
     dp01 <- def.neon.dp01(data = data)

   # Level 1 descriptive statistics in verbose output
     dp01Vrbs <- def.neon.dp01(data = data, vrbs = TRUE)

# argument vrbs = TRUE is useful for preserving unit information on
# per-variable basis and use with lapply() and do.call()

   # data.frame which variables contain the unit attributes

     # create data.frame
     data <- data.frame(
       velo = rnorm(10),
       temp = rnorm(10),
       dist = rnorm(10)
     )

     # assign unit attribute
     attributes(data$velo)$unit <- "m s-1"
     attributes(data$temp)$unit <- "K"
     attributes(data$dist)$unit <- "m"

   # vrbs = FALSE does not propagate unit information
   attributes(def.neon.dp01(data = data, vrbs = FALSE)$se.velo)$unit
   # NULL

   # vrbs = TRUE propagates unit information
   attributes(def.neon.dp01(data = data, vrbs = TRUE)$se$velo)$unit
   # [1] "m s-1"
```

`def.neon.read.hdf5.para` *Definition function: Function to read in dp0p files*

Description

Definition function. A function to read in dp0p files and extact relevant attributes as parameters for future use.

Usage

```
def.neon.read.hdf5.para(DirFileParaLoca, GrpName, PosPara = NULL)
```

Arguments

DirFileParaLoca	Absolute path to h5 file from which workflow and scientific parameters are to be used, of class "character". [-]
GrpName	Absolute path to h5 file group and variable name from which parameters are to be read, of class "character". [-]
PosPara	Parameters can be subset by PosPara; defaults to NULL which imports all parameters. A vector of characters. [-]

Value

The returned object is the parameters extracted from dp0p file for future use.

Author(s)

Stefan Metzger <eddy4R.info@gmail.com>
David Durden <ddurden@battelleecology.org>
Hongyan Luo <hluo@battelleecology.org>

References

License: GNU AFFERO GENERAL PUBLIC LICENSE Version 3, 19 November 2007.

See Also

Currently none.

Examples

```
Currently none
```

`def.neon.read.hdf5.qfqm`
Wrapper function: Reading quality flags from NEON HDF5 files

Description

definition function. Reads an HDF5 input file in NEON standard format from `DirInpLoca`.

Usage

```
def.neon.read.hdf5.qfqm(DirInpLoca, SiteLoca, DateLoca, VarLoca,
  LevlTowr = c("000_040", "000_050", "000_060")[3], FreqLoca,
  MethMeas = c("ecte", "ecse")[1])
```

Arguments

`DirInpLoca`	Character: Input directory.
`SiteLoca`	Character: Site location.
`DateLoca`	Character: Date in ISO format "(2016-05-26").
`VarLoca`	Character: Which instrument to read data from.
`LevlTowr`	The tower level that the sensor data is being collected in NEON data product convention (HOR_VER)
`FreqLoca`	Integer: Measurement frequency.
`MethMeas`	A vector of class "character" containing the name of measurement method (eddy-covariance turbulent exchange or storage exchange), MethMeas = c("ecte", "ecse"). Defaults to "ecte".

Value

Named list `qfqm` containing time-series of quality flags.

Author(s)

David Durden `<ddurden@battelleecology.org>`

References

License: GNU AFFERO GENERAL PUBLIC LICENSE Version 3, 19 November 2007.

See Also

Currently none.

Examples

```
Currently none.
```

`def.para.flow` *Definition function: Determine the workflow variables*

Description

Definition function. Function to determine the workflow variables by either reading them in from the environmental variables, defining them explicitly, or defining them by default values

Usage

```
def.para.flow(Deve = TRUE, DirFilePara = NULL, DirInp = NA, DirMnt = NA,
  DirOut = NA, DirTmp = NA, DirWrk = NA, DateOut = NULL,
  FileOutBase = NULL, Read = "hdf5", VersDp = c("001", "004")[1],
  VersEddy = "latest", MethParaFlow = c("DfltInp", "EnvVar")[1], UrlInpRefe,
  UrlOutRefe, ...)
```

Arguments

`Deve`	is logical that determines if only a subset of the data should be read in to reduce testing time during development (`Deve = TRUE`) or all the input data should be read in (`Deve = FALSE`)
`DirFilePara`	is file path for the hdf5 dp0p input data file
`DirInp`	is directory path for the hdf5 dp0p input data file
`DirMnt`	is the base directory path for where the docker is mounted
`DirOut`	is directory path for the output data
`DirTmp`	is directory path for temporary storage during processing
`DirWrk`	is directory path for working storage during processing
`DateOut`	is a character string that lists the dates to produce output data
`FileOutBase`	is a character string that denotes the base file name for output
`Read`	determine if the data are read from hdf5 dp0p input data file or other input files
`VersDp`	is the data product level that will be output
`VersEddy`	is the version of the eddy4R docker that is being used to perform the processing
`MethParaFlow`	is the method used to specify workflow parameters, "EnvVar" will grab ParaFlow parameters from environmental variable and "DfltInp" will use whatever is specified in the function call.
`UrlInpRefe`	A single-entry vector of class "character" containing the web address of the reference input data zip file to be downloaded.
`UrlOutRefe`	A single-entry vector of class "character" containing the web address of the reference output data zip file to be downloaded.

Value

`ParaFlow` is a list returned that indicates the workflow control parameters, including `ParaFlow$DirFilePara`,`ParaFlow$Dir`
`ParaFlow$DirMnt`, `ParaFlow$DirOut`, `ParaFlow$DirTmp`, `ParaFlow$DirWrk`, `ParaFlow$DateOut`, `ParaFlow$FileOutBase`, `ParaFlow$Read`, `ParaFlow$VersDp`, `ParaFlow$VersEddy`.

Author(s)

David Durden <`ddurden@battelleecology.org`>

References

License: GNU AFFERO GENERAL PUBLIC LICENSE Version 3, 19 November 2007.
NEON Algorithm Theoretical Basis Document:Eddy Covariance Turbulent Exchange Subsystem Level 0 to Level 0’ data product conversions and calculations (NEON.DOC.000823)
NEON Algorithm Theoretical Basis Document:Eddy Covariance Turbulent Exchange Subsystem Level 1 data product calculations (NEON.DOC.000807)

See Also

Currently none

Examples

```
def.para.flow(DirFilePara = "test.h5", MethParaFlow = "DfltInp")
```

`def.para.hdf5.dp01` *Definition function: Copy metadata from HDF5 input file to another HDF5 output file*

Description

Definition function. Function recreates the metadata from an HDF5 file to another HDF5 file with the same group heirarchy structure.

Usage

```
def.para.hdf5.dp01(FileIn, FileOut)
```

Arguments

`FileIn`	is the input file where the parameters are being read from.
`FileOut`	is the output file where the parameters are being written to.

Value

A NEON formatted HDF5 file that has parameters from the input file written to the output HDF5 file.

Author(s)

David Durden `<ddurden@battelleecology.org>`

References

License: GNU AFFERO GENERAL PUBLIC LICENSE Version 3, 19 November 2007.
NEON Algorithm Theoretical Basis Document:Eddy Covariance Turbulent Exchange Subsystem Level 0 to Level 0' data product conversions and calculations (NEON.DOC.000823)
NEON Algorithm Theoretical Basis Document:Eddy Covariance Turbulent Exchange Subsystem Level 1 data product calculations (NEON.DOC.000807)

See Also

Currently none

Examples

```
Currently none
```

`def.para.site` *Definition function: Determine site location and tower top measurement level from input dp0p ECTE HDF5 file*

Description

Definition function. Function determines the site location and tower top measurement level from input dp0p ECTE HDF5 file structure

Usage

```
def.para.site(FileInp)
```

Arguments

`FileInp`	is the input dp0p ECTE HDF5 file where the parameters are being read from.

Value

`rpt` is list returned that indicates NEON specific site four letter code (`rpt$Loc`) and the Horizontal and Vertical indices of the tower top measurement level (`rpt$LevlTowr`).

Author(s)

David Durden <`ddurden@battelleecology.org`>

References

License: GNU AFFERO GENERAL PUBLIC LICENSE Version 3, 19 November 2007.
NEON Algorithm Theoretical Basis Document:Eddy Covariance Turbulent Exchange Subsystem Level 0 to Level 0' data product conversions and calculations (NEON.DOC.000823)
NEON Algorithm Theoretical Basis Document:Eddy Covariance Turbulent Exchange Subsystem Level 1 data product calculations (NEON.DOC.000807)

See Also

Currently none

Examples

```
Currently none
```

`def.para.thsh`	*Definition function to read threshold table from CI-Parameter-Repo*

Description

Definition function to read threshold table from CI-Parameter-Repo

Usage

```
def.para.thsh(dataIn, site)
```

Arguments

`dataIn`	data.frame containing the threshold data from the CI-Parameter-Repo
`site`	The site for which the parameters are being pulled

Value

A data.frame consisting of the threshold values to be used for the provided site.

Author(s)

Natchaya Pingintha-Durden <`ndurden@battelleecology.org`> David Durden <`ddurden@battelleecology.org`>

See Also

Currently none

`def.pol.cart`	*Definition function: Composing azimuth angle from cartesian vector data*

Description

Composing azimuth angle from cartesian vector data.

Usage

```
def.pol.cart(cart)
```

Arguments

`cart`	variable of type numeric, matrix consisting of unit vectors X (1st column, -1 ... 1, positive to north) and Y (2nd column, -1 ... 1, positive to east)

Value

clockwise azimuth angle with 0 / 360 degree discontinuity in north [decimal degrees]

Author(s)

Stefan Metzger <`eddy4R.info@gmail.com`>
Natchaya Pingintha-Durden <`ndurden@battelleecology.org`>

References

License: GNU AFFERO GENERAL PUBLIC LICENSE Version 3, 19 November 2007

See Also

Currently none

Examples

```
Currently none
```

---

def.pres.h2o.dens.mass.h2o.temp
Definition function: Calculate water vapor pressure from absolute humidity and ambient temperature

---

Description

Calculate water pressure from absolute humidity and ambient temperature.

Usage

```
def.pres.h2o.dens.mass.h2o.temp(densMassH2o, temp)
```

Arguments

`densMassH2o`	Either a vector or an object of class numeric of absolute humidity and of the same length as `temp`. [kg m-3]
`temp`	Either a vector or an object of class numeric of measured air temperature and of the same length as `densMassH2o`. [K]

Value

Water vapor pressure and of the same length as `densMassH2o` and `temp`. [Pa]

Author(s)

Stefan Metzger <`eddy4R.info@gmail.com`>
Natchaya Pingintha-Durden <`ndurden@battelleecology.org`>

References

License: GNU AFFERO GENERAL PUBLIC LICENSE Version 3, 19 November 2007

See Also

Currently none

Examples

```
def.pres.h2o.dens.mass.h2o.temp(densMassH2o = 1.493*1e-2, temp = 298.15)
def.pres.h2o.dens.mass.h2o.temp(densMassH2o = c(3.451*1e-4, 2.717*1e-3, 1.493*1e-2), temp = c(265.15, 278.15, 298
```

---

def.pres.h2o.pres.h2o.sat.rh

Definition function: Calculate water vapor pressure from saturated water vapor pressure and relative humidity

---

Description

Calculate water vapor pressure from saturated water vapor pressure and relative humidity.

Usage

```
def.pres.h2o.pres.h2o.sat.rh(presH2oSat, rh)
```

Arguments

`presH2oSat`	Either a vector or an object of class numeric of saturated water vapor pressure and of the same length as `rh`. [Pa]
`rh`	Either a vector or an object of class numeric of relative humidity and of the same length as `presH2oSat`. [-]

Value

Water vapor pressure and of the same length as `presH2oSat` and `rh`. [Pa]

Author(s)

Stefan Metzger <`eddy4R.info@gmail.com`>
Natchaya Pingintha-Durden <`ndurden@battelleecology.org`>

References

License: GNU AFFERO GENERAL PUBLIC LICENSE Version 3, 19 November 2007

See Also

Currently none

Examples

```
def.pres.h2o.pres.h2o.sat.rh(presH2oSat = 3160.057, rh = 0.65)
def.pres.h2o.pres.h2o.sat.rh(presH2oSat = c(422.19, 1701.67, 6265.31), rh = c(0.10, 0.45, 0.80))
```

---

def.pres.h2o.rtio.mass.h2o.dry.pres

Definition function: Calculate water vapor pressure from dry mass fraction and static pressure

---

Description

Calculate water vapor pressure from dry mass fraction and static pressure.

Usage

```
def.pres.h2o.rtio.mass.h2o.dry.pres(rtioMassDryH2o, pres)
```

Arguments

`rtioMassDryH2o`	Either a vector or an object of class numeric of water dry mass fraction and of the same length as `pres`. [kg kg-1]
`pres`	Either a vector or an object of class numeric of static pressure and of the same length as `rtioMassDryH2o`. [Pa]

Value

Water vapor pressure and of the same length as `rtioMassDryH2o` and `pres`. [Pa]

Author(s)

Stefan Metzger <`eddy4R.info@gmail.com`>
Natchaya Pingintha-Durden <`ndurden@battelleecology.org`>

References

License: GNU AFFERO GENERAL PUBLIC LICENSE Version 3, 19 November 2007

See Also

Currently none

Examples

```
def.pres.h2o.rtio.mass.h2o.dry.pres(rtioMassDryH2o = 2.144*1e-2, pres = 65000)
def.pres.h2o.rtio.mass.h2o.dry.pres(rtioMassDryH2o = c(2.144*1e-2, 2.617*1e-3, 4.931*1e-4), pres = c(65000, 83000
```

def.pres.h2o.rtio.mole.h2o.dry.pres

Definition function: Calculate water vapor pressure from dry mole fraction and static pressure

Description

Calculate water vapor pressure from dry mole fraction and static pressure.

Usage

```
def.pres.h2o.rtio.mole.h2o.dry.pres(rtioMoleDryH2o, pres)
```

Arguments

`rtioMoleDryH2o`	Either a vector or an object of class numeric of water dry mole fraction and of the same length as `pres`. [kmol kmol-1]
`pres`	Either a vector or an object of class numeric of static pressure and of the same length as `rtioMoleDryH2o`. [Pa]

Value

Water vapor pressure and of the same length as `rtioMoleDryH2o` and `pres`. [Pa]

Author(s)

Stefan Metzger <`eddy4R.info@gmail.com`>
Natchaya Pingintha-Durden <`ndurden@battelleecology.org`>

References

License: GNU AFFERO GENERAL PUBLIC LICENSE Version 3, 19 November 2007

See Also

Currently none

Examples

```
def.pres.h2o.rtio.mole.h2o.dry.pres(rtioMoleDryH2o = 6.52*1e-6, pres = 83000)
def.pres.h2o.rtio.mole.h2o.dry.pres(rtioMoleDryH2o = c(6.52*1e-6, 11.617*1e-6, 4.931*1e-6), pres = c(83000, 65000
```

def.pres.h2o.sat.temp.mag

Definition function: Calculate saturated water vapor pressure from temperature using Magnus equation

Description

Calculate saturated water vapor pressure from temperature using Magnus equation.

Usage

```
def.pres.h2o.sat.temp.mag(temp)
```

Arguments

`temp`	Either a vector or an object of class numeric of measured air temperature. [K]

Value

Saturated water vapor pressure and of the same length as `temp`. [Pa]

Author(s)

Stefan Metzger <`eddy4R.info@gmail.com`>
Natchaya Pingintha-Durden <`ndurden@battelleecology.org`>

References

License: GNU AFFERO GENERAL PUBLIC LICENSE Version 3, 19 November 2007.
Gueymard, C.: Assessment of the accuracy and computing speed of simplified saturation vapor equations using a new reference dataset, J. Appl. Meteorol., 32, 1294-1300, doi:10.1175/1520-0450(1993)0321294:aotaac2.0.co;2, 1993.

See Also

Currently none

Examples

```
def.pres.h2o.sat.temp.mag(temp = 265.15)
def.pres.h2o.sat.temp.mag(temp = c(265.15, 285.0, 290.2))
```

`def.pres.sum` *Definition function: Calculation of total pressure in LI-7200 IRGA cell*

Description

Function definition. Calculation of total pressure from static pressure and differential pressure

Usage

```
def.pres.sum(presAtm, presDiff)
```

Arguments

`presAtm`	A vector containing the atmospheric pressure (or static pressure), of class "numeric". [Pa]
`presDiff`	A vector containing the differential pressure, of class "numeric". [Pa]

Value

The returned object is the total pressure calculated by summing the static pressure (presAtm) and the differential pressure (presDiff).

Author(s)

Stefan Metzger <`eddy4R.info@gmail.com`>
Hongyan Luo <`hluo@battelleecology.org`>

References

License: GNU AFFERO GENERAL PUBLIC LICENSE Version 3, 19 November 2007.

See Also

Currently none.

Examples

```
Example 1, this will cause an error message due to presAtm and presDiff have no units:
def.pres.sum(presAtm = 99380, presDiff = -1109)
Example 2, assign valuea and units to variables first, the function should run ok.
presAtm = 99380
presDiff = -1109
attributes(presAtm)$unit = "Pa"
attributes(presDiff)$unit = "Pa"
def.pres.sum(presAtm, presDiff)
```

def.repl.char	*Definition function: Replace character vector elements with entries from a lookup table*

Description

Function defintion. If any of the values in ReplFrom exist in data, these values are replaced with the corresponding values in ReplTo. If none of the values in ReplFrom exist in data, then data is returned unchanged.

Usage

```
def.repl.char(data, ReplFrom, ReplTo)
```

Arguments

data	Named character vector, can contain values that are to be replaced as specified via ReplFrom and ReplTo.
ReplFrom	Character vector of the same length as ReplTo, specifying the values in data that are to be replaced with the corrresponding values in ReplTo.
ReplTo	Character vector of the same length as ReplFrom, specifying the replacement values for ReplFrom in data.

Value

Named character vector of the same length as data, with values specified in ReplFrom replaced by the corresponding values in ReplTo. If none of the entries in ReplFrom exist in data, then data is returned unchanged.

Author(s)

Stefan Metzger <eddy4R.info@gmail.com>

References

License: GNU AFFERO GENERAL PUBLIC LICENSE Version 3, 19 November 2007.

See Also

Currently none.

Examples

```
def.repl.char(
data = structure(.Data = c("Celsius", "Pa", "percent"), names = c("temperature", "pressure", "humidity")),
ReplFrom = c("Celsius", "percent"),
ReplTo = c("C", "%")
)
```

def.repl.na — *Defination Function to combine two daily files into one file and replacing NA values with previous value.*

Description

Defination Function to two combine daily files into one file, replacing NA with previous value, and divide file back into daily.
Specific for pump and soleniod valves.

Usage

```
def.repl.na(dataProcDate, dataBgnDate, Date, numCol)
```

Arguments

`dataProcDate`	A dataframe containing data and time in the processing date. [User-defined]
`dataBgnDate`	A dataframe containing data and time in the day before processing date. [User-defined]
`Date`	Character: Processing date e.g. "20170521". [-]
`numCol`	Numeric: Define which column number in `dataProcDate` and `dataBgnDate` are being process. [-]

Value

A dataframe including the data and time in the processing date which NA values have been replaced by previous value. [User-defined]

Author(s)

Natchaya Pingintha-Durden <`ndurden@battelleecology.org`>

References

License: Terms of use of the NEON FIU algorithm repository dated 2015-01-16.

See Also

Currently none

Examples

```
Currently none
```

`def.rglr` *Definition function: Regularizing irregular data to strictly regular / equidistant data*

Description

Function defintion. Takes a (potentially) irregularly spaced timeseries `timeMeas` of data `dataMeas` and returns a strictuly regularly spaced timeseries `timeRglr` of data `dataRglr`. **ATTENTION**: `MethRglr = "zoo"` uses the zoo:na.approx() function, which does not currently abide by its `maxgap` argument version 1.7-13. In result, where gaps exist currently the last known value is repeated instead of NAs being inserted. An Email with a request for bugfixing has been sent to `<Achim.Zeileis@R-project.org>` (2016-05-08).

Usage

```
def.rglr(timeMeas, dataMeas, unitMeas = base::attributes(dataMeas)$unit,
  BgnRglr = NULL, EndRglr = NULL,
  TzRglr = base::attributes(BgnRglr)$tzone, FreqRglr, MethRglr = c("CybiEc",
  "cybiDflt", "zoo")[1], WndwRglr = c("Cntr", "Lead", "Trlg")[1],
  PosWndw = c("Clst", "PosWndwMin", "PosWndwMax")[1], PrcsSec = 6)
```

Arguments

`timeMeas`	A vector containing the observation times. Of class "POSIXlt" including timezone attribute, and of the same length as `dataMeas`. [-]
`dataMeas`	A named data.frame containing the observations. Columns may be of class "numeric" or "integer", and of the same length as `timeMeas`. Columns of classes other than "numeric" or "integer" are removed and not included in the returned `dataRegl`. [user-defined]
`unitMeas`	A vector containing the unit of each column in `dataMeas`. Of class "character". It is recommended to conform to the "unit representation" guidelines documented in the eddy4R.base package.
`BgnRglr`	Desired begin time for the regularized dataset. Of class "POSIXlt" including timezone attribute, and `length(BgnRglr) = 1`. This input is not used in the "cybiDflt" method. [-]
`EndRglr`	Desired end time for the regularized dataset. Of class "POSIXlt" including timezone attribute, and `length(EndRglr) = 1`. This input is not used in the "cybiDflt" method. [-]
`TzRglr`	Desired timezone for the regularized dataset. Of class "character" and `length(TzRglr) = 1`, defaults to the same timezone as `BgnRglr`. For the "cybiDflt" method, the same time zone as timeMeas is used. [-]
`FreqRglr`	Desired frequency of the regularized dataset. Of class "numeric" or "integer" and `length(FreqRglr) = 1`. [Hz]

`MethRglr`	Switch for different regularization methods. Of class "character", currently defaults to "CybiEc". [-] Method "cybiDflt" implements the default for metereological variable regularization performed by NEON CI. Namely, a new time series is created from the first measurement time, rounded toward zero, using the expected data frequency. The first measurement falling in between one time stamp and the next is assigned to the first of these, and all other measurements falling in this range are ignored. Method "CybiEc" implements the default regularization method for eddy-covariance processing utilized CI. The procedure is documented in NEON.DOC.001069. Method "zoo" implements the regularization method using the zoo::na.approx function. This method can only handle up to millisecond precision (PrcsSec=3)
`WndwRglr`	Position of the window for binning in the "CybiEc" method. `WndwRglr` can be centered [Cntr], leading [Lead], or trailing [Trlg] (defaults to centered).
`PosWndw`	Determines which observation to allocate to a bin if multiple observations fall into a single bin when using the "CybiEc" method.. `PosWndw` can be set to closest [Clst], first [PosWndwMin], or last [PosWndwMax] (defaults to closest).
`PrcsSec`	A single numeric (integer) value indicating the operational precision of the seconds field of time vectors. Defaults to 6 (microsecond-precision). Values higher than 6 cannot be guaranteed to produce desired results.

Value

Returns a list with elements `TzRglr`, `FreqRglr`, `MethRglr`, `timeRglr`, and `dataRglr`.

Author(s)

Stefan Metzger <`eddy4R.info@gmail.com`>
Cove Sturtevant <`eddy4R.info@gmail.com`>
David Durden <`ddurden@battelleecology.org`>

References

License: GNU AFFERO GENERAL PUBLIC LICENSE Version 3, 19 November 2007.
NEON.DOC.001069 Preprocessing ATBD: The ATBD that describes the CybiEc and cybiDflt regularization methods.

See Also

?zoo:na.approx, ?stats::approx

Examples

```
# make sure that fractional seconds can be seen from the console
options(digits.secs=3)
# assign measured time vector
timeMeas <- base::as.POSIXlt(seq.POSIXt(
  from = base::as.POSIXlt("2016-01-01 00:00:00.001", format="%Y-%m-%d %H:%M:%OS", tz="UTC"),
  to = base::as.POSIXlt("2016-01-01 00:00:01.002", format="%Y-%m-%d %H:%M:%OS", tz="UTC"),
   by = 1/10), tz = "UTC")[-c(5,6,8)]
# assign fake observations
dataMeas <- base::data.frame("wind01" = rnorm(base::length(timeMeas)), "wind02" = rnorm(base::length(timeMeas)))
# regularize
def.rglr(
  timeMeas = timeMeas,
  dataMeas = dataMeas,
  unitMeas = c("metersPerSecond", "metersPerSecond"),
 BgnRglr = base::as.POSIXlt("2016-01-01 00:00:00.000", format="%Y-%m-%d %H:%M:%OS", tz="UTC"),
 EndRglr = base::as.POSIXlt("2016-01-01 00:00:01.000", format="%Y-%m-%d %H:%M:%OS", tz="UTC"),
  FreqRglr = 10,
  MethRglr = "zoo"
)

#"CybiEc" example with multiple observations in a single bin
timeMeas[3] <- timeMeas[3] - 0.03
timeMeas[2] <- timeMeas[2] - 0.03
timeMeas[1] <- timeMeas[1] + 0.05
timeMeas[7] <- timeMeas[7] - 0.031
timeMeas[8] <- timeMeas[8] - 0.081

#Regularize with a centered window and chosing the closest value to the regularized timestamp.
def.rglr(
  timeMeas = timeMeas,
  dataMeas = dataMeas,
  unitMeas = c("metersPerSecond", "metersPerSecond"),
 BgnRglr = base::as.POSIXlt("2016-01-01 00:00:00.000", format="%Y-%m-%d %H:%M:%OS", tz="UTC"),
 EndRglr = base::as.POSIXlt("2016-01-01 00:00:01.000", format="%Y-%m-%d %H:%M:%OS", tz="UTC"),
  FreqRglr = 10,
  MethRglr = "CybiEc",
  WndwRglr = "Cntr",
  PosWndw = "Clst"
)
```

---

def.rh.pres.h2o.pres.sat.h2o

Definition function: Calculation of RH from water vapor partial pressure and saturation pressure

---

Description

Function definition. Calculation of RH from water vapor partial pressure and saturation pressure

Usage

```
def.rh.pres.h2o.pres.sat.h2o(presH2o, presH2oSat)
```

Arguments

presH2o	A vector containing the water vapor partial pressure, of class "numeric". [Pa]
presH2oSat	A vector containing the water vapor saturation pressure, of class "numeric". [Pa]

Value

The returned object is the RH calculated from water vapor partial pressure and saturation pressure

Author(s)

Stefan Metzger <eddy4R.info@gmail.com>
Hongyan Luo <hluo@battelleecology.org>

References

License: GNU AFFERO GENERAL PUBLIC LICENSE Version 3, 19 November 2007.

See Also

Currently none.

Examples

```
Example 1, this will cause an error message due to presH2o and presH2oSat have no units:
def.rh.pres.h2o.pres.sat.h2o(presH2o = 1020, presH2oSat = 2500)
Example 2, assign values and units to variables first, the function should run ok.
presH2o = 1020
presH2oSat = 2500
attributes(presH2o)$unit = "Pa"
attributes(presH2oSat)$unit = "Pa"
def.rh.pres.h2o.pres.sat.h2o(presH2o, presH2oSat)
```

---

def.rmsd.diff.prcs.rsq

Definition function: RMSD, bias, precision and coefficient of determination - incl. deadband

---

Description

Function defintion.RMSD, bias, precision and coefficient of determination - incl. deadband.

Usage

```
def.rmsd.diff.prcs.rsq(refe, test, Perc = FALSE, Deba = NULL,
  DebaRltv = FALSE)
```

Arguments

`refe`	Variable of class numeric. Reference data.Same unit as test data.
`test`	Variable of class numeric. Test data.Same unit as reference data.
`Perc`	Variable of class logical. It describe if the output is in percentage.
`Deba`	Variable of class numeric. Numbers of the deadband around zero denominator
`DebaRltv`	Variable of class logical.Absolute or relative (percentage) deadband around zero. True indicates Deba in relative (percentage), False indicates Deba in absolute numbers.

Value

Currently none

Author(s)

Stefan Metzger <`eddy4R.info@gmail.com`>
Hongyan Luo <`hluo@battelleecology.org`>

References

License: GNU AFFERO GENERAL PUBLIC LICENSE Version 3, 19 November 2007.

See Also

Currently none

Examples

```
#refe dataset
df1 <- c(runif(200, min=-100, max=400))
#test dataset
df2 <- c(runif(200, min=-100, max=400))
#run function
def.rmsd.diff.prcs.rsq(refe=df1,test=df2,Perc = FALSE,Deba=NULL, DebaRltv=FALSE)
```

`def.rot.enu.ned` *Definition function: Coordinate transformation from the east-north-up (Enu) coordiante system to the north-east-down (Ned) coordiante system*

Description

Function defintion. Coordinate transformation from the east-north-up (Enu) coordiante system to the north-east-down (Ned) coordiante system, and vice versa.

Usage

```
def.rot.enu.ned(angEnu, Meth = c("vec", "ang"))
```

Value

Azimuth angle in the North-east-down local coordinate system [rad]

Author(s)

David Durden <ddurden@battelleecology.org>

References

License: GNU AFFERO GENERAL PUBLIC LICENSE Version 3, 19 November 2007.

See Also

Currently none.

Examples

```
Currently none.
```

`def.rsmp` *Definition function: Resampling function*

Description

Function definition. Resampling data by changing the sample rate from high frequency to low frequency.

Usage

```
def.rsmp(data, FreqInp, FreqOut, MethRsmp, ColDisc = NULL)
```

Arguments

`data`	A vector or matrix of type numeric, containing the data to be resampled. [user-defined]
`FreqInp`	Input sampling or measurements frequency. Of class "numeric" or "integer". [Hz]
`FreqOut`	Desired out put frequency. Of class "numeric" or "integer". [Hz]
`MethRsmp`	An object of class "character" containing the resampling method ("zoo","filt"). [-]
`ColDisc`	Specific column in `data` which containing the discontinuity variable e.g. azimuth angle of wind direction. Of class "numeric" (column number) or "character" (column name). Defaults to NULL. [-]

Value

Data frame of resampling data and of the same number of variables as `data`. [user-defined]

Author(s)

Stefan Metzger <`eddy4R.info@gmail.com`>
Natchaya Pingintha-Durden <`ndurden@battelleecology.org`>

References

Currently none License: GNU AFFERO GENERAL PUBLIC LICENSE Version 3, 19 November 2007.

See Also

Currently none

Examples

```
#Generate data frame
df <- data.frame("A"= runif(200, min=0, max=360),"B" = runif(200, min=0.1, max=10), "C" = rnorm(200))
df1 <- def.rsmp(data = df,FreqInp = 10,FreqOut = 5, MethRsmp = "filt",ColDisc = c(1))
df2 <- def.rsmp(data = df,FreqInp = 10,FreqOut = 0.5, MethRsmp = "zoo",ColDisc = c("A"))
```

`def.rtio.mass.h2o.dens.mole` *Definition function: Calculation of the wet mass fraction (specifc humidity) from mole density of water vapor and mole density of dry air*

Description

Function definition. Calculate the wet mass fraction (specific humidity) of air mixture from mole density of water vapor and mole density of dry air

Usage

```
def.rtio.mass.h2o.dens.mole(densMoleH2o, densMoleAirDry)
```

Arguments

`densMoleH2o`	A vector containing mole density of water vapor, of class "numeric". [molH2o m-3]
`densMoleAirDry`	A vector containing the mole density of dry air, of class "numeric". [mol m-3]

Value

The returned object is wet mass fraction (specific humidity)

Author(s)

Stefan Metzger <`eddy4R.info@gmail.com`>
Hongyan Luo <`hluo@battelleecology.org`>

References

License: GNU AFFERO GENERAL PUBLIC LICENSE Version 3, 19 November 2007.

See Also

Currently none.

Examples

```
Example 1, this will cause an error message due to densMoleH2o and densMoleAirDry have no units:
def.rtio.mass.h2o.dens.mole(densMoleH2o = 0.3, densMoleAirDry = 41.1)
Example 2, assign values and units to variables first, the function should run ok.
densMoleH2o = 0.3
densMoleAirDry = 41.1
attributes(densMoleH2o)$unit = "molH2o m-3"
attributes(densMoleAirDry)$unit = "mol m-3"
def.rtio.mass.h2o.dens.mole (densMoleH2o, densMoleAirDry)
```

`def.rtio.mass.h2o.wet.pres.h2o.pres` *Definition function: Calculate water vapor wet mass fraction (specific humidity) from water vapor pressure and static pressure*

Description

Calculate specific humidity from water vapor pressure and static pressure.

Usage

```
def.rtio.mass.h2o.wet.pres.h2o.pres(presH2o, pres)
```

Arguments

`presH2o`	Either a vector or an object of class numeric of measured water vapor pressure and of the same length as `pres`. [Pa]
`pres`	Either a vector or an object of class numeric of static pressure and of the same length as `presH2o`. [Pa]

Value

Specific humidity and of the same length as `presH2o` and `pres`. [kg kg-1]

Author(s)

Stefan Metzger <eddy4R.info@gmail.com>
Natchaya Pingintha-Durden <ndurden@battelleecology.org>

References

License: GNU AFFERO GENERAL PUBLIC LICENSE Version 3, 19 November 2007.

See Also

Currently none

Examples

```
def.rtio.mass.h2o.wet.pres.h2o.pres(presH2o = 2212.04 , pres = 65000)
def.rtio.mass.h2o.wet.pres.h2o.pres(presH2o = c(2212.04, 348.69, 87.17) , pres = c(65000, 83000, 110000))
```

`def.site.info` *Definition function: Load pre-defined site specific information into the global environment*

Description

Load pre-defined site specific information into the global environment.

Usage

```
def.site.info(loc = c("IM", "LOS", "NR", "NS", "PF", "SERC", "CPER")[6])
```

Arguments

`loc`	Parameter of class character. A parameter to choose the site for which site specific information is pulled.

Value

Site specific information provided as a list `SiteInfo`

Author(s)

Stefan Metzger <eddy4R.info@gmail.com>
David Durden <ddurden@battelleecology.org>

References

License: GNU AFFERO GENERAL PUBLIC LICENSE Version 3, 19 November 2007.

See Also

Currently none

Examples

```
def.site.info("SERC")
```

---

def.smth *Definition function: Data smoothing*

Description

Function definition.Data smoothing method.

Usage

```
def.smth(data, span, smthFunc = c("smthKern"))
```

Arguments

`data`	A vector containing the input data. Of class "numeric" or "integer". []
`span`	A Vector containing the span values. Of class "numeric" or "integer". The option span controls the degree of smoothing. For example; using span = c(15) to apply a long smoothing filter, or using spans = c(3,3) to apply two short filters in succession. [-]
`smthFunc`	An object of class string containing the smooth functions ("smthKern"). []

Value

Estimated smooth values and of the same length as `data`. []

Author(s)

Stefan Metzger <`eddy4R.info@gmail.com`>
Natchaya Pingintha-Durden <`ndurden@battelleecology.org`>

References

License: GNU AFFERO GENERAL PUBLIC LICENSE Version 3, 19 November 2007.
`http://www.stat.rutgers.edu/home/rebecka/Stat565/lab42007.pdf`

See Also

`kernel`, `kernapply`

Examples

```
def.smth(data = rnorm(100), span = c(15), smthFunc = c("smthKern"))
def.smth(data = rnorm(100), span = c(3,3))
```

`def.ssi.diff` *Definition function: Calculate delta(or difference) of signal strength for LI-7200 IRGA*

Description

Function definition. Calculation of signal strength difference for LI-7200 IRGA.

Usage

```
def.ssi.diff(ssiCo2, ssiH2o)
```

Arguments

`ssiCo2`	A vector containing the CO2 signal strength, of class "numeric". [unitless]
`ssiH2o`	A vector containing the H2O signal strength, of class "numeric". [unitless]

Value

The returned object is the signal strength difference between the CO2 signal strength and the H2O signal strength.

Author(s)

Stefan Metzger <`eddy4R.info@gmail.com`>
Hongyan Luo <`hluo@battelleecology.org`>

References

License: GNU AFFERO GENERAL PUBLIC LICENSE Version 3, 19 November 2007.

See Also

Currently none.

Examples

```
example 1 (This will give error message becuase ssiCo2 and ssiH2o have no units):
def.ssi.diff(ssiCo2 = 0.550, ssiH2o =0.561)
example 2 (Assign the units and values to the variable before run function, which shoudl work fine.)
ssiCo2 =0.550
ssiH2o = 0.561
attributes(ssiCo2)$unit <- "-"
attributes(ssiH2o)$unit <- "-"
def.ssi.diff(ssiCo2, ssiH2o)
```

`def.ssi.mean` *Definition function: Calculation of average signal strength for LI-7200 IRGA*

Description

Function definition. Calculation of average signal strength for LI-7200 IRGA.

Usage

```
def.ssi.mean(ssiCo2, ssiH2o)
```

Arguments

`ssiCo2`	A vector containing the CO2 signal strength, of class "numeric". [percent]
`ssiH2o`	A vector containing the H2O signal strength, of class "numeric". [percent]

Value

The returned object is the average signal strength calculated from the CO2 signal strength and the H2O signal strength.

Author(s)

Stefan Metzger <`eddy4R.info@gmail.com`>
Hongyan Luo <`hluo@battelleecology.org`>

References

License: GNU AFFERO GENERAL PUBLIC LICENSE Version 3, 19 November 2007.

See Also

Currently none.

Examples

```
Currently none.
```

`def.temp.dew.pres.h2o.temp.mag` *Definition function: Calculate dew point temperature from water vapor pressure and ambient temperature using Magnus equation*

Description

Calculate dew point temperature from water vapor pressure and ambient temperature using Magnus equation.

Usage

```
def.temp.dew.pres.h2o.temp.mag(presH2o, temp)
```

Arguments

`presH2o`	Either a vector or an object of class numeric of measured water vapor pressure and of the same length as `temp`. [Pa]
`temp`	Either a vector or an object of class numeric of measured air temperature and of the same length as `presH2o`. [K]

Value

Dew point temperature and of the same length as `presH2o` and `temp`. [K]

Author(s)

Stefan Metzger <`eddy4R.info@gmail.com`>
Natchaya Pingintha-Durden <`ndurden@battelleecology.org`>

References

License: GNU AFFERO GENERAL PUBLIC LICENSE Version 3, 19 November 2007.
Gueymard, C.: Assessment of the accuracy and computing speed of simplified saturation vapor equations using a new reference dataset, J. Appl. Meteorol., 32, 1294-1300, doi:10.1175/1520-0450(1993)0321294:aotaac2.0.co;2, 1993.

See Also

Currently none

Examples

```
def.temp.dew.pres.h2o.temp.mag(presH2o = 2212.04, temp = 298.15)
def.temp.dew.pres.h2o.temp.mag(presH2o = c(42.22, 87.22, 699.78), temp = c(265.15, 278.15, 293.15))
```

`def.temp.mean.7200` *Definition function: Calculation of the average temperature in LI-7200 irga cell*

Description

Function definition. Calculation of the average temperature in LI-7200 IRGA cell

Usage

```
def.temp.mean.7200(tempIn, tempOut, Vali = FALSE)
```

Arguments

`tempIn`	A vector containing the tempertaure measured at IRGA cell inlet, of class "numeric". [K]
`tempOut`	A vector containing the tempertaure measured at IRGA cell outlet, of class "numeric". [K]

Value

The returned object is the the average temperature in LI-7200 IRGA cell calculated from the temperatrue at IRGA cell inlet and the temperature at IRGA cell outlet.

Author(s)

Stefan Metzger <`eddy4R.info@gmail.com`>
Hongyan Luo <`hluo@battelleecology.org`>

References

License: GNU AFFERO GENERAL PUBLIC LICENSE Version 3, 19 November 2007.

See Also

Currently none.

Examples

```
Example 1, this will cause an error message due to tempIn and tempOut have no units:
def.temp.mean.7200(tempIn = 293, tempOut = 294)
Example 2, assign values and units to variables first, the function should run ok.
tempIn = 293
tempOut = 294
attributes(tempIn)$unit = "K"
attributes(tempOut)$unit = "K"
def.temp.mean.7200(tempIn, tempOut)
```

def.temp.pres.pois *Definition function: Poisson's equation (adiabatic change) - temperature as function of pressure change*

Description

Poisson's equation (adiabatic change) - temperature as function of pressure change.

Usage

```
def.temp.pres.pois(temp01, pres01, pres02,
  Kppa = eddy4R.base::IntlNatu$KppaDry)
```

Arguments

`pres01`	Measured air pressure [same unit as reference pressure]
`pres02`	Reference pressure [same unit as reference pressure]
`temp01`	Measured air temperature [K]
`Kppa`	Ratio of specific gas constant to specific heat at constant pressure. Default as KppaDry [-]

Value

Temperature at refernce pressure [K]

Author(s)

Stefan Metzger <`eddy4R.info@gmail.com`>
Natchaya Pingintha-Durden <`ndurden@battelleecology.org`>

References

License: GNU AFFERO GENERAL PUBLIC LICENSE Version 3, 19 November 2007

See Also

Currently none

Examples

```
temp02 <- def.temp.pres.pois(temp01 = 298.15, pres01 = 845, pres02 = 1000, Kppa = eddy4R.base::IntlNatu$KppaDry)
```

def.temp.soni	*Definition function: Calculation of sonic temperature from speed of sound*

Description

Function definition. Calculation of sonic temperature from speed of sound using Eq. (9) in Appendix C of CSAT3 Three Dimensional Sonic Anemometer Instruction manual

Usage

```
def.temp.soni(veloSoni)
```

Arguments

`veloSoni`	A vector containing speed of sound, of class "numeric". [m s-1]

Value

The returned object is sonic temperature

Author(s)

Stefan Metzger <`eddy4R.info@gmail.com`>
Hongyan Luo <`hluo@battelleecology.org`>

References

License: GNU AFFERO GENERAL PUBLIC LICENSE Version 3, 19 November 2007.
CSAT3 Three Dimensional Sonic Anemometer Instruction Mannual (Revisoin 2/15). Logan, Utah, USA. Campbell Scientifc, Inc. (2015)

See Also

Currently none.

Examples

```
Example 1, this will cause an error message due to veloSoni has no units:
def.temp.soni(veloSoni = 344)
Example 2, assign values and units to variables first, the function should run ok.
veloSoni = 344
attributes(veloSoni)$unit = "m s-1"
def.temp.soni(veloSoni)
```

`def.unit.extr` *Definition function: Extract variable with unit attribute from data frame*

Description

Function definition. Extracts a variable from a data frame (with units attached as attributes at the data frame level), and attaches the variable's unit directly to the extracted variable.

Usage

```
def.unit.extr(data, nameVar, AllwPos = FALSE)
```

Arguments

`data`	Required. A data frame with units of each variables attached as a named attribute "unit" to the data frame.
`nameVar`	Required. A string containing the name of the variable within `data` to extract, along with its units
`AllwPos`	Optional. Logical, defaulting to FALSE. Allow positional determination of the unit within the unit attribute vector? If FALSE, an error will result if the variable name does not match an entry with the named unit vector. If TRUE, the unit attribute vector does not need to be named, and the unit will be pulled based on the same position of the variable within the data frame `data`.

Value

The desired variable with units attached as attribute.

Author(s)

Cove Sturtevant <`eddy4R.info@gmail.com`>

References

License: GNU AFFERO GENERAL PUBLIC LICENSE Version 3, 19 November 2007.

See Also

Currently none

Examples

```
Currently none
```

def.unit.info — *Definition function: Interpret unit string*

Description

Function definition. Interpret a compound unit string, identifying the matching entries within eddy4R internal data `eddy4R.base::IntlUnit` for the base unit symbol, unit prefix, chemical species, and unit suffix (raise to the power of)

Usage

```
def.unit.info(unit)
```

Arguments

`unit`	Required. A single character string providing the compount unit, constructed with the following rules. Unit character strings must be constructed from the base unit symbols (case-sensitive) listed in eddy4R.base internal data IntlUnit$Base$Symb (e.g. the symbol for meter is "m"). Unit base symbols can be directly preceded (no space) by the case-sensitive unit prefixes listed in eddy4R.base internal data IntlUnit$Prfx (e.g. kilometers = "km") . Unit base symbols can be directly followed (no spaces) by the suffix `n`, where `n` is an integer (...-2,-1,0,1,2...), indicating the unit is raised to the power of `n` (e.g. per square kilometer = "km-2"). In the case of chemical species (i.e. converting between mass and molar units), specify the full unit (including prefix and suffix) followed immediately (no spaces inbetween) by one of the chemical species characters listed in eddy4R.base internal data IntlUnit$Spcs (eg. per gram of carbon dioxide = "g-1CO2"): Compound units can be formed by inserting spaces between the individual unit components (ex.milligrams carbon per meter squared per day = "mgC m-2 d-1").

Value

A named list of vectors, each of length N, where N is the number of base units making up the compound unit. For example, umolCO2 m-2 s-1 has 3 base units (mol, m, s):
type: the abbreviated character string denoting unit type (e.g. "dist" for distance) base: the base unit character symbol as listed in eddy4R.base::IntlUnit$Base$Symb posBase: the integer list position in eddy4R.base::IntlUnit$Base$Symb corresponding to the base unit prfx: the unit prefix character(s) as listed in eddy4R.base::IntlUnit$Prfx posPrfx: the integer list position in eddy4R.base::IntlUnit$Prfx

corresponding to the unit prefix sufx: the numerical unit suffix (i.e. the power to which the unit is raised) spcs: a character string of the chemical species as listed in eddy4R.base::Unit$Spcs posSpcs: the integer list position in eddy4R.base::IntlUnit$Spcs corresponding to the chemical species

Author(s)

Cove Sturtevant <`eddy4R.info@gmail.com`>

References

License: GNU AFFERO GENERAL PUBLIC LICENSE Version 3, 19 November 2007.

See Also

`def.unit.conv`, `def.unit.intl`, `IntlUnit$Intl`

Examples

```
Currently none
```

`def.unit.var` *Definition function: Assign unit attribute to each variable in a new object from an existing object*

Description

Function defintion. eddy4R strives to provide a unit attribute individually for each physical parameter and variable in an object. However, most native R functions do not propagate an existing unit attribute in the returned object. The function `def.unit.var` fills this gap, by assigning the unit attribute of a parameter/variable that exists (by name) in `refe` to the corresponding parameter/variable in `samp`. `def.unit.var` currently only supports data.frames, but additional methods can be added for lists and ffdf objects.

Usage

```
def.unit.var(samp, refe)
```

Arguments

`samp`	A data.frame containing variables that are missing the unit attribute.
`refe`	A data.frame containing all variables (by name) of `samp` incl. their corresponding unit attribute.

Value

The function returns `samp` with the corresponding units from `refe` assigned to each of its variables.

Author(s)

Stefan Metzger <`eddy4R.info@gmail.com`>

References

License: GNU AFFERO GENERAL PUBLIC LICENSE Version 3, 19 November 2007.

See Also

Currently none.

Examples

```
# data.frame to which variables the unit attribute is to be assigned
samp <- data.frame(
  velo = rnorm(10),
  temp = rnorm(10)
)
attributes(samp$velo)$unit
NULL

# data.frame which variables contain the unit attributes

  # create data.frame
  refe <- data.frame(
    velo = rnorm(10),
    temp = rnorm(10),
    dist = rnorm(10)
  )

  # assign unit attribute
  attributes(refe$velo)$unit <- "m s-1"
  attributes(refe$temp)$unit <- "K"
  attributes(refe$dist)$unit <- "m"

  samp <- def.unit.var(samp = samp, refe = refe)
  attributes(samp$velo)$unit
  [1] "m s-1"
```

`IntlNatu` *Internal data: Parameter set: Natural constants*

Description

Natural constants, such as the ideal gas constant, for use across eddy4R package suite.

Usage

```
IntlNatu
```

Format

A list of named constants:

Grav acceleration of gravity = 9.81 [m s-2]

PrdErth rotation period of the Earth (one sidereal day) = 86164.1 [s]

AvelErth angular velocity of Earth = 7.2921159e-05 [rad s-1]

Pres00 NIST standard pressure = 101325 [Pa == kg m-1 s-2]

Temp00 NIST standard temperature = 293.15 [K]

Rg ideal gas constant = 8.314462175 [J mol-1 K-1 == kg m2 s-2 mol-1 K-1]

VonkFokn von-Karman constant accordig to Foken (2008) = 0.4 [-]

MolmDry molecular mass of dry air = 28.97e-3 [kg mol-1]

MolmH2o molecular mass of water vapor = 18.02e-3 [kg mol-1]

MolmCo2 molecular mass of carbon dioxide = 44.01e-3 [kg mol-1]

MolmCh4 molecular mass of methane = 16.04e-3 [kg mol-1]

MolmC molecular mass of carbon = 12e-3 [kg mol-1]

RtioMolmH2oDry molar mass ratio water vapour / dry air = 0.6220228 [-]

RtioMolmDryH2o molar mass ratio dry air / water vapour = 1.607658 [-]

CpDry dry air specific heat at constant pressure = 1004.64 [J kg-1 K-1]

CvDry dry air specific heat at constant volume = 717.6 [J kg-1 K-1]

RsDry specific gas constant for dry air = 287.04 [J kg-1 K-1]

GmmaDry CpDry / CvDry = 1.4 [-]

KppaDry Dry air Kappa exponent for ideal gas law (Poisson), RsDry / CpDry = 0.2857 [-]

CpH2o water vapour specific heat at constant pressure = 1846 [J kg-1 K-1]

CvH2o water vapour specific heat at constant volume = 1384.04 [J kg-1 K-1]

RsH2o specific gas constant for water vapour = 461.96 [J kg-1 K-1]

GmmaH2o CpH2o / CvH2o = 1.333776 [-]

KppaH2o water vapour Kappa exponent for ideal gas law (Poisson), RsH2o / CpH2o = 0.2502 [-]

Source

Natural constants are defined within flow.save.intl.cnst.R, available in the data-raw/ folder of the source version of the package

---

`IntlUnit` *Internal data: Parameter set: Unit representations*

Description

Unit base symbols, types, prefixes, associated chemical species, and definition of the internal units used in the eddy4R family of packages.

The unit representations below are used to construct character strings describing variable units. Unit character strings are constructed following these rules:

The (case-sensitive) unit base symbol is selected from those listed in IntlUnit$Base$Symb (e.g. the base symbol for meter is "m").

Unit base symbols can be directly preceded (no space) by the (case-sensitive) unit prefixes listed in IntlUnit$Prfx (e.g. kilometers = "km") .

Unit base symbols can be directly followed (no spaces) by the suffix `n`, where `n` is an integer (...-2,-1,0,1,2...), indicating the unit is raised to the power of `n` (e.g. per square kilometer = "km-2").

In the case of chemical species attached to a unit, specify the full unit (including prefix and suffix) followed immediately (no spaces) by one of the chemical species character strings in IntlUnit$Spcs (eg. per gram of carbon dioxide = "g-1Co2"):

Compound units can be formed by inserting spaces between the individual unit components (ex.milligrams carbon per meter squared per day = "mgC m-2 d-1").

Usage

```
IntlUnit
```

Format

A nested list of named unit categories:

Base Symb A named list of base unit (case-sensitive) character designations:

`Metr = "m"` (meter)
`Feet = "ft"` (foot)
`Inch = "inch"` (inch)
`Mile = "mi"` (statute mile)
`MileNaut = "NM"` (nautical mile)
`Gram = "g"` (gram)
`Pnd  = "lb"` (pound-mass)
`Ton  = "t"` (metric ton)
`TonUs = "ST"` (US/short ton)

`Scnd = "s"` (second)
`Mint = "min"` (minute)
`Hour = "h"` (hour)
`Day  = "d"` (day)
`Year = "y"` (year)
`Kelv = "K"` (Kelvin)
`Cels = "C"` (degrees Celcius)
`Frht = "F"` (degrees Fahrenheit)
`Mole = "mol"` (mole)
`Rad  = "rad"` (radian)
`Deg  = "deg"` (degree - geometry)
`Ampr = "A"` (ampere)
`Volt = "V"` (volt)
`Ohm  = "ohm"` (ohm)
`Pasc = "Pa"` (Pascal)
`Bar  = "bar"` (bar)
`Atm  = "atm"` (standard atmosphere)
`Torr = "Torr"` (torr)
`Psi  = "psi"` (pound-force per square inch)
`Watt = "W"` (watt)
`Joul = "J"` (joule)
`Newt = "N"` (newton)
`Pndf = "lbf"` (pound-force)
`Hrtz = "Hz"` (hertz)

Type A named list of the unit type corresponding to entries in IntlUnit$Base$Symb. Unit types are used to constrain conversions between variables (except for mol to/from mass units, conversion is only allowed within the same unit type). Unit types are also used to find the eddy4R internal units (listed in IntlUnit$Intl):

`Metr = "Dist"` (distance)
`Feet = "Dist"` (distance)
`Inch = "Dist"` (distance)
`Mile = "Dist"` (distance)
`MileNaut = "Dist"` (distance)
`Gram = "Mass"` (mass)
`Pnd  = "Mass"` (mass)
`Ton  = "Mass"` (mass)
`TonUs = "Mass"` (mass)
`Scnd = "Time"` (time)
`Mint = "Time"` (time)
`Hour = "Time"` (time)
`Day  = "Time"` (time)
`Year = "Time"` (time)
`Kelv = "Temp"` (temperature)
`Cels = "Temp"` (temperature)
`Frht = "Temp"` (temperature)
`Mole = "Num"` (number)

```
Rad  = "Ang" (angle)
Deg  = "Ang" (angle)
Ampr = "Curr" (electrical current)
Volt = "Epot" (electrical potential)
Ohm  = "Eres" (electrical resistance)
Pasc = "Pres" (pressure)
Bar  = "Pres" (pressure)
Atm  = "Pres" (pressure)
Torr = "Pres" (pressure)
Psi  = "Pres" (pressure)
Watt = "Powr" (power)
Joul = "Engy" (energy)
Newt = "Forc" (force)
Pndf = "Forc" (force)
Hrtz = "Freq" (frequency)
```

Prfx A named list of unit prefix (case-sensitive) character designations:

```
Exa  = "E" (1e18)
Peta = "P" (1e15)
Tera = "T" (1e12)
Giga = "G" (1e9)
Mega = "M" (1e6)
Kilo = "k" (1e3)
Hect = "h" (1e2)
Deca = "da" (1e1)
Deci = "d" (1e-1)
Cnti = "c" (1e-2)
Mili = "m" (1e-3)
Micr = "u" (1e-6)
Nano = "n" (1e-9)
Pico = "p" (1e-12)
Femt = "f" (1e-15)
Atto = "a" (1e-18)
```

Spcs A character vector of chemical species used in unit designations (not case-sensitive):

```
"C" (carbon)
"Co2" (carbon dioxide)
"H2o" (water vapor)
"Ch4" (methane)
"Dry" (dry air)
```

Intl A named list of the units (including prefixes) used internally in the eddy4R family of functions. List names are unit types selected from IntlUnit$Base$Type. List values are unit base symbol and prefix character strings selected from IntlUnit$Base$Symb and IntlUnit$Prfx, respectively. Only one unit per type is used internally:

```
Dist = "m"
Mass = "kg"
Time = "s"
Temp = "K"
Num  = "mol"
Ang  = "rad"
Curr = "A"
Epot = "V"
Eres = "ohm"
Pres = "Pa"
Powr = "W"
Engy = "J"
Forc = "N"
Freq = "Hz"
```

Source

Units are defined within flow.save.intl.cnst.R, available in the data-raw/ folder of the source version of the package

---

`wrap.agr.vari.seSq` *Wrapper function: Calculate aggregated variance and squared standard error from a small to large temporal scale.*

Description

Function wrapper. Calculate aggregated variance and squared standard error from a finer to a coarser temporal resolution: from minutely to hourly resolution, from hourly to diurnal cycle, and from diurnal cycle to monthly. The input data can be irregular, because data expansion is included in the current wrapper to generate unbiased mean, variance and squared standard error.

Usage

```
wrap.agr.vari.seSq(data = data.frame(timeDoyDecm, mean, vari), zone, MethAgr)
```

Arguments

`data`	Dataframe of type numeric containing column vectors `timeDoyDecm`, `mean`, and `vari` of equal length.
`timeDoyDecm`	input decimal doy time in UTC [float number].
`mean`	Vector of type numeric. Means of the variable of interest at finer resolution, e.g. minutely [user-defined].
`vari`	Vector of type numeric. Variances of the variable of interest at finer resolution, e.g. minutely [user-defined^2].

`zone`	The time zone of the location, which is the offset from Coordinated Universal Time (UTC) by a whole number of hours. If local time is ahead (behind) the Greenwich mean time, zone is a positive (negative) number. e.g. CST" [hour]. a time offset in Wisconsin, in the North American Central Time Zone, would be -6.
`MethAgr`	String type. One of three choices: "agrHour", "mdc", "agrMnth". "agrHour" represents aggregation from minutely to hourly resolution; "mdc" represents aggregation from hourly to diurnal cycle; "agrMnth" represents aggregation from diurnal cycle to monthly resolution.

Value

Returns object of class "dataframe": if MethAgr is "agrHour", the dataframe is [n, 1:5] containing timeDoyIntg, hour, mean, variance, square of standard error of the scalar at hourly resolution if MethAgr is "mdc", the dataframe is [n, 1:4] containing hour, mean, variance, square of standard error of the scalar at diurnal cycle if MethAgr is "agrMnth", the dataframe is [n, 1:3] containing mean, variance, square of standard error of the scalar at monthly scale

Author(s)

Ke Xu `<xuke2012abroad@gmail.com>`

References

License: GNU AFFERO GENERAL PUBLIC LICENSE Version 3, 19 November 2007.

See Also

Currently none

Examples

```
wrap.agr.vari.seSq(
data = data.frame(
  timeDoyDecm = c(1441:(5*1440))/24/60,
  mean = rnorm(4*1440),
  vari = rnorm(4*1440)^2
),
zone = -6,
MethAgr = "agrHour"
)
```

wrap.derv.prd.day *Wrapper function: Calculation of derived quantities (daily extent, native resolution)*

Description

Wrapper function. Reads the list `inpList` in the format provided by function `eddy4R.base::wrap.neon.read.hdf5.eddy(`
For the list entries in `inpList` the following derived quantities are calculated, each through the call to a separate definition function:
`inpList$data$time`: fractional UTC time, fractional day of year, local standard time;
`inpList$data$irgaTurb`: average signal strength, delta signal strength, total pressure, average temperature, water vapor partial pressure, water vapor saturation pressure, relative humidity, molar density of air (dry air and water vapor), molar density of dry air, wet mass fraction (specific humidity);
`inpList$data$soni`: sonic temperature.

Usage

```
wrap.derv.prd.day(inpList, ZoneTime, AngZaxsSoniInst)
```

Arguments

`inpList`	List consisting of `ff::ffdf` file-backed objects, in the format provided by function `eddy4R.base::wrap.neon.read.hdf5.eddy()`. Of types numeric and integer.
`AngZaxsSoniInst`	Parameter of class numeric. Azimuth (angle around z axis) direction against true north in which sonic anemometer installation (transducer array) is pointing [deg]
`SiteLoca`	List consisting of site-specific parameters. Of types numeric, integer and character.

Value

The returned object consistes of `inpList`, with the derived variables added to the respective list entry, and all list levels sorted alphabetically.

Author(s)

Stefan Metzger <`eddy4R.info@gmail.com`>

References

License: GNU AFFERO GENERAL PUBLIC LICENSE Version 3, 19 November 2007.

See Also

Currently none.

Examples

```
Currently none.
```

`wrap.hdf5.wrte.dp01` *Wrapper function: Write NEON Level 1 data, qfqm, and uncertainty to output HDF5*

Description

Wrapper function. To write NEON Level 1 data product descriptive statistics (mean, minimum, maximum, variance, number of non-NA points), quality flags and quality metrics, and uncertainty quantification to an output HDF5 file.

Usage

```
wrap.hdf5.wrte.dp01(inpList, FileIn, FileOut, SiteLoca, LevlTowr,
  MethUcrt = TRUE, MethDp04 = FALSE, MethSubAgr = TRUE)
```

Arguments

`inpList`	A list of including dp01 data, quality flags and quality metrics, and uncertainty calculations to package and write to an output HDF5 file
`FileIn`	The file name for the input dp0p HDF5 file to grab metadata
`FileOut`	The file name for the output HDF5 file
`SiteLoca`	Character: Site location.
`LevlTowr`	The tower level that the sensor data is being collected in NEON data product convention (HOR_VER)
`MethUcrt`	Logical: Determines if uncertainty information is available for output.
`MethDp04`	logical indicating if ECTE dp04 HDF5 data should be included.
`MethSubAgr`	Logical: Determines if 1-minute data is available for output.

Value

An HDF5 file with dp01 data, qfqm, and uncertainty written

Author(s)

David Durden <`ddurden@battelleecology.org`>

References

License: GNU AFFERO GENERAL PUBLIC LICENSE Version 3, 19 November 2007.

See Also

Currently none.

Examples

```
Currently none.
```

---

wrap.hdf5.wrte.dp01.api

Wrapper function: Gather/Write reingested NEON Level 1 data, qfqm, and uncertainty to output HDF5

Description

Definition function. To write NEON Level 1 data product descriptive statistics (mean, minimum, maximum, variance, number of non-NA points), quality flags and quality metrics, and uncertainty values gathered from via the API to an output HDF5 file.

Usage

```
wrap.hdf5.wrte.dp01.api(date, FileOut, SiteLoca, DpName = c("tempAirLvl",
  "tempAirTop"), LevlTowr, TimeAgr = c(1, 30))
```

Arguments

`date`	Character: The date for the data to be gathered in ISO format ("YYYYmmdd").
`FileOut`	Character: The file name for the output HDF5 file
`SiteLoca`	Character: Site location.
`LevlTowr`	Character: The tower level that the sensor data is being collected in NEON data product convention (HOR_VER).
`TimeAgr`	Integer: The time aggregation index in minutes (i.e. 30)
`Dp01`	Character: A vector of data product names for the data to be gathered.

Value

An updated dp0p HDF5 file with dp01 data, qfqm, and uncertainty written

Author(s)

David Durden <`ddurden@battelleecology.org`>

References

License: GNU AFFERO GENERAL PUBLIC LICENSE Version 3, 19 November 2007.

See Also

Currently none.

Examples

```
Currently none.
```

wrap.neon.dp01 — *Wrapper function: Create NEON Level 1 data product descriptive statistics across list elements*

Description

Wrapper function. Compute NEON Level 1 data product descriptive statistics (mean, minimum, maximum, variance, number of non-NA points) across list elements.

Usage

```
wrap.neon.dp01(data, idx = NULL)
```

Arguments

`data`	A data.frame or list containing the L0p (calibrated) input data at native resolution. Of type numeric or integer. [-]
`idx`	If data is a list, which list entries should be processed into Level 1 data products? Defaults to NULL which expects data to be a data.frame. Of type character. [-]

Value

Descriptive statistics, for `vrbs = FALSE` a data frame and for `vrbs = TRUE` a list:
`mean` The mean of non-NA values in `data` `min` The minimum value of non-NA values in `data` `max` The maximum value of non-NA values in `data` `vari` The variance of non-NA values in `data` `num` The number of non-NA values in `data` `se` The standard error of the mean of non-NA values in `data`

Author(s)

Stefan Metzger <`eddy4R.info@gmail.com`>

References

License: GNU AFFERO GENERAL PUBLIC LICENSE Version 3, 19 November 2007.

See Also

Currently none.

Examples

```
# data.frame which variables contain the unit attributes

  # create list with L0p data
  data <- list(
    sens01 = data.frame(
      velo = rnorm(10),
      temp = rnorm(10)),
    sens02 = data.frame(
      velo = rnorm(10),
      temp = rnorm(10)),
    sens03 = data.frame(
      velo = rnorm(10),
      temp = rnorm(10))
    )

  # assign unit attribute
  attributes(data$sens01$velo)$unit <- "m s-1"
  attributes(data$sens01$temp)$unit <- "K"
  attributes(data$sens02$velo)$unit <- "m s-1"
  attributes(data$sens02$temp)$unit <- "K"
  attributes(data$sens03$velo)$unit <- "m s-1"
  attributes(data$sens03$temp)$unit <- "K"

# calculate L1 data products only for sensors sens01 and sens02
rpt <- wrap.neon.dp01(data = data, idx = c("sens01", "sens02"))
# units are propagated
attributes(rpt$sens02$se$velo)$unit
# [1] "m s-1"
```

---

wrap.neon.dp01.agr.prd

Wrapper function: Create NEON Level 1 data products with quality flags and quality metrics for different aggregation periods

---

Description

Wrapper function. Compute NEON Level 1 data products with quality flags and quality metrics for different aggregation periods (e.g. 1 minute).

Usage

```
wrap.neon.dp01.agr.prd(inpList)
```

Arguments

`inpList`	A list of including dp0p data and quality flags to perform dp01 calculation and quality tests.

Value

A list containing dp01 data and quality flags and metrics for all sensors including the following: `data`: `mean` The mean of non-NA values in `data` `min` The minimum value of non-NA values in `data` `max` The maximum value of non-NA values in `data` `vari` The variance of non-NA values in `data` `num` The number of non-NA values in `data` `se` The standard error of the mean of non-NA values in `data` `qfqm`: `qm` A list of data frame's containing quality metrics (fractions) of failed, pass, and NA for each of the individual flag which related to L1 sub-data products if RptExpd = TRUE. [fraction]

`qmAlph` A dataframe containing metrics in a columns of class "numeric" containing the alpha quality metric for L1 sub-data products. [fraction]
`qmBeta` A dataframe containing metrics in a columns of class "numeric" containing the beta quality metric for L1 sub-data products. [fraction]
`qfFinl` A dataframe containing flags in a columns of class "numeric", [0,1], containing the final quality flag for L1 sub-data products. [-]
`qfSciRevw` A dataframe containing flags in a columns of class "numeric", [0,1], containing the scientific review quality flag for L1 sub-data products. [-]

Author(s)

David Durden <`ddurden@battelleecology.org`>

References

License: GNU AFFERO GENERAL PUBLIC LICENSE Version 3, 19 November 2007.

See Also

Currently none.

Examples

```
Currently none
```

`wrap.neon.dp01.ecse` *Wrapper function: Preprocessing and computing NEON eddy-covariance stroage exchange L1 data product descriptive statistics*

Description

Wrapper function. Preprocessing and computing NEON eddy-covariance stroage exchange (ECSE) Level 1 data product (dp01) descriptive statistics (mean, minimum, maximum, variance, number of non-NA points).

Usage

```
wrap.neon.dp01.ecse(dp01 = c("co2Stor", "h2oStor", "tempAirLvl", "tempAirTop",
  "isoCo2", "isoH2o")[1], lvl, lvlMfcSampStor = NULL, lvlEnvHut = NULL,
  lvlValv = NULL, lvlCrdH2oValvVali = NULL, data = list(),
  qfInput = list(), TypeMeas = c("samp", "vali")[1], PrdMeas, PrdAgr,
  idxTime = list())
```

Arguments

`dp01`	A vector of class "character" containing the name of NEON ECSE dp01 which descriptive statistics are being calculated, c("co2Stor", "h2oStor", "tempAirLvl", "tempAirTop", "isoCo2", "isoH2o"). Defaults to "co2Stor". [-]
`lvl`	Measurement level of dp01 which descriptive statistics are being calculated. Of type character. [-]
`lvlMfcSampStor`	Measurement level of mfcSampStor which apply to only dp01 equal to "co2Stor" or "h2oStor". Defaults to NULL. Of type character. [-]
`lvlEnvHut`	Measurement level of envHut. Defaults to NULL. Of type character. [-]
`lvlValv`	Measurement level of irgaValvLvl, crdCo2ValvLvl, or crdH2oValvLvl. Defaults to NULL. Of type character. [-]
`lvlCrdH2oValvVali`	Measurement level of crdH2oValvVali which apply to only dp01 equal to "isoH2o". Defaults to NULL. Of type character. [-]
`data`	A list of data frame containing the input dp0p data that related to dp01 which descriptive statistics are being calculated. Of class integer". [User defined]
`qfInput`	A list of data frame containing the input quality flag data that related to dp01 are being grouped. Of class integer". [-]
`TypeMeas`	A vector of class "character" containing the name of measurement type (sampling or validation), TypeMeas = c("samp", "vali"). Defaults to "samp". [-]
`PrdMeas`	The measurement time period in minute. [min]
`PrdAgr`	The time period to aggregate to averaging in minute. [min]
`idxTime`	A list of data frame containing the indices and corresponding times for aggregation periods. [-]

Value

A list of dp01 descriptive statistics.
`mean` The mean of non-NA values in `data` `min` The minimum value of non-NA values in `data` `max` The maximum value of non-NA values in `data` `vari` The variance of non-NA values in `data` `numSamp` The number of non-NA values in `data` `se` The standard error of the mean of non-NA values in `data` `timeBgn` The beginning time to determine descriptive statistics. `timeEnd` The ending time to determine descriptive statistics.

Author(s)

Natchaya Pingintha-Durden <`eddy4R.info@gmail.com`>

References

License: GNU AFFERO GENERAL PUBLIC LICENSE Version 3, 19 November 2007.

See Also

Currently none.

Examples

```
Currently none.
```

`wrap.neon.dp01.qfqm.ec`
Wrapper function: Create NEON Level 1 data product quality flags and quality metrics across list elements

---

Description

Wrapper function. Compute NEON Level 1 data product quality flags and quality metrics (qfFinl, qfSciRevw, qmAlph, qmBeta) across list elements.

Usage

```
wrap.neon.dp01.qfqm.ec(qfqm, idx = NULL, TypeMeas = "samp",
  MethMeas = c("ecte", "ecse")[1], RptExpd = FALSE)
```

Arguments

`qfqm`	A data.frame or list containing the L0p input data quality flags (sensor and plausibility flags) at native resolution. Of type numeric or integer. [-]
`idx`	If data is a list, which list entries should be processed into Level 1 data product quality metrics? Defaults to NULL which expects qfqm to be a data.frame. Of type character. [-]
`TypeMeas`	A vector of class "character" containing the name of measurement type (sampling or validation), TypeMeas = c("samp", "ecse"). Defaults to "samp". [-]
`MethMeas`	A vector of class "character" containing the name of measurement method (eddy-covariance turbulent exchange or storage exchange), MethMeas = c("ecte", "ecse"). Defaults to "ecse". [-]
`RptExpd`	A logical parameter that determines if the full quality metric `qm` is output in the returned list (defaults to FALSE).

Value

A list of:
`qm` A list of data frame's containing quality metrics (fractions) of failed, pass, and NA for each of the individual flag which related to L1 sub-data products if RptExpd = TRUE. [fraction]
`qmAlph` A dataframe containing metrics in a columns of class "numeric" containing the alpha quality metric for L1 sub-data products. [fraction]
`qmBeta` A dataframe containing metrics in a columns of class "numeric" containing the beta quality metric for L1 sub-data products. [fraction]
`qfFinl` A dataframe containing flags in a columns of class "numeric", [0,1], containing the final quality flag for L1 sub-data products. [-]
`qfSciRevw` A dataframe containing flags in a columns of class "numeric", [0,1], containing the scientific review quality flag for L1 sub-data products. [-]

Author(s)

David Durden <ddurden@battelleecology.org>

References

License: GNU AFFERO GENERAL PUBLIC LICENSE Version 3, 19 November 2007.

See Also

Currently none.

Examples

```
#generate the fake quality flags for each sensor
TimeBgn <- "2016-04-24 02:00:00.000"
TimeEnd <- "2016-04-24 02:29:59.950"
qf <- list()
qf$irgaTurb <- eddy4R.qaqc::def.qf.ecte(TimeBgn = TimeBgn, TimeEnd = TimeEnd, Freq = 20, Sens = "irgaTurb", PcntQf
qf$mfcSampTurb <- eddy4R.qaqc::def.qf.ecte(TimeBgn = TimeBgn, TimeEnd = TimeEnd, Freq = 20, Sens = "mfcSampTurb", P
qf$soni <- eddy4R.qaqc::def.qf.ecte(TimeBgn = TimeBgn, TimeEnd = TimeEnd, Freq = 20, Sens = "soni", PcntQf = 0.05)
qf$amrs <- eddy4R.qaqc::def.qf.ecte(TimeBgn = TimeBgn, TimeEnd = TimeEnd, Freq = 40, Sens = "amrs", PcntQf = 0.05)
#calculate quality metric, qmAlpha, qmBeta, qfFinl
qfqm <- list()
qfqm <- wrap.neon.dp01.qfqm.ec(qfqm = wrk$qfqm, idx = c("soni", "amrs", "co2Turb", "h2oTurb") )
```

---

wrap.neon.read.hdf5.eddy

Wrapper function: Reading NEON HDF5 files

Description

Wrapper function. Reads an HDF5 input file in NEON standard format from `DirInpLoca`. Subsequently, (i) name and unit attributes are converted to eddy4R standard terms, (ii) variable units are converted accordingly, (iii) the data is regularized, (iv) sensor diagnostic tests are performed, (v) a range test is performed, and (vi) the data is de-spiked.

Usage

```
wrap.neon.read.hdf5.eddy(DirInpLoca, SiteLoca, DateLoca, VarLoca,
  LevlTowr = c("000_040", "000_050", "000_060")[3], FreqLoca, Rglr = FALSE,
  Diag = FALSE, Rng = FALSE, RngLoca, DespLoca, MethMeas = c("ecte",
  "ecse")[1])
```

Arguments

`DirInpLoca`	Character: Input directory.
`SiteLoca`	Character: Site location.
`DateLoca`	Character: Date in ISO format "(2016-05-26").
`VarLoca`	Character: Which instrument to read data from.
`LevlTowr`	The tower level that the sensor data is being collected in NEON data product convention (HOR_VER)
`FreqLoca`	Integer: Measurement frequency.
`RngLoca`	List of named ingegers: Thresholds for range test.
`DespLoca`	List of integers: De-spiking parameters
`MethMeas`	A vector of class "character" containing the name of measurement method (eddy-covariance turbulent exchange or storage exchange), MethMeas = c("ecte", "ecse"). Defaults to "ecte".

Value

Named list containing pre-processed time-series $time and $data.

Author(s)

Stefan Metzger <`eddy4R.info@gmail.com`>

References

License: GNU AFFERO GENERAL PUBLIC LICENSE Version 3, 19 November 2007.

See Also

Currently none.

Examples

```
Currently none.
```

`wrap.para.thsh` *Wrapper Function to read threshold table from CI-Parameter-Repo*

Description

Workflow to read ECSE threshold table from CI-Parameter-Repo

Usage

```
wrap.para.thsh(DpName = c("IrgaTurb", "MfcSampTurb", "Soni", "Amrs"),
  DirInp = "~/eddy/data/Thresholds_EC/CI-Parameter-Repo/ParaSci/Ecte/Dp0p",
  DirOut = "~/eddy/data/Thresholds_EC/threshold_ecte", FileWrte = FALSE)
```

Value

A data.frame consisting of the threshold values to be used for the provided site.

Author(s)

Natchaya Pingintha-Durden <ndurden@battelleecology.org> David Durden <ddurden@battelleecology.org>

See Also

Currently none

Examples

```
Currently none
```

---

wrap.prd.day.ecse *Wrapper function: To perform daily ECSE processing in native resolution*

Description

Wrapper function. To perform daily ECSE processing in native resolution

Usage

```
wrap.prd.day.ecse(inpList, Desp)
```

Arguments

`inpList`	List consisting of input data in the format provided by function `eddy4R.base::wrap.neon.read.hdf5.e` Of types numeric, integer, character and POSIXct.
`Desp`	De-spiking parameters as mixed list of types integer and character with the following list entries: `widt` de-spiking median filter window width (single integer); `nbin` de-spiking histogram bins initial number/step size (single integer); `rest` de-spiking resolution threshold (single integer); `var` sub-list of sensors in `inpList`, with each list entry containing the variable names for which to perform de-spiking (character). See `?eddy4R.qaqc::def.dspk.br86` for details.

Value

The returned object consistes of `rpt` after applying the daily processing.

Author(s)

Natchaya Pingintha-Durden <eddy4R.info@gmail.com>
Stefan Metzger <eddy4R.info@gmail.com>

References

License: GNU AFFERO GENERAL PUBLIC LICENSE Version 3, 19 November 2007.

See Also

Currently none.

Examples

```
Currently none.
```

`wrap.time.rglr.dp00.ecse` *Wrapper function: Time regularization for ECSE dp00*

Description

Wrapper function. Time regularization for ECSE dp00

Usage

```
wrap.time.rglr.dp00.ecse(DirIn, Date, Site = "CPER", Dom = "D10", Freq,
  IdDp00, horVer)
```

Arguments

`DirIn`	Character: Input directory. [-]
`Date`	Character: Processing date e.g. "20170521". [-]
`Site`	Character: Site location. [-]
`Dom`	Character: Domain. [-]
`Freq`	Desired frequency of the regularized dataset. Of class "numeric" or "integer". [Hz]
`IdDp00`	Character: dp00 data product number. [-]
`horVer`	Character: Horizontal and vertical location of dp00. [-]

Value

A dataframe including the regularized dp00. [User-defined]

Author(s)

Natchaya Pingintha-Durden <`ndurden@battelleecology.org`>

References

License: Terms of use of the NEON FIU algorithm repository dated 2015-01-16.

See Also

Currently none.

Examples

```
Currently none.
```

`wrap.unit.conv.out.ec` *Wrapper function: Output unit conversion for ECTE*

Usage

```
wrap.unit.conv.out.ec(inpList, MethType = c("Data", "Ucrt", "Qfqm")[1])
```

Arguments

`inpList`	Required. A named list of data frames of type numeric, containing the data to be converted.
`MethType`	Required. A character string containing the type of data to be converted (Defauts to (MethType) = "data").} } { A list, (rpt), with data, qfqm, or uncertainty output with the correct output units Function wrapper. Convert a list of data to eddy4r output units using def.unit.conv function, with special attention to variable with the mean removed that are translated between units (i.e. variance for temperature when converting K to C). Currently none. License: GNU AFFERO GENERAL PUBLIC LICENSE Version 3, 19 November 2007. Currently none. David Durden <`eddy4R.info@gmail.com`> HDF5 output,

Index

Package ‘eddy4R.qaqc’

March 22, 2018

Title Eddy-covariance calculation for R: Quality assurance and quality control

Version 0.2.8

Description Eddy-covariance calculation for R: Quality assurance and quality. Currently removed from Imports because devtools::install_github() ignores and attempts to re-install previously installed packages, but without passing GITHUB_PAT (starting 2016-06-26, only affets packages hosted on private Github repos with access token, such as NEONInc/NEON-FIU-algorithm): eddy4R.base (>= 0.2.19).

Depends R (>= 3.4.0)

Imports ggplot2 (>= 2.2.1), grid(>= 3.4.0), gridExtra (>= 2.2.1), gtable (>= 0.2.0), lattice (>= 0.20-35), plyr (>= 1.8.4), reshape2 (>= 1.4.2), Rmisc (>= 1.5)

License GNU AFFERO GENERAL PUBLIC LICENSE Version 3, 19 November 2007

LazyData true

RoxygenNote 6.0.1

Author Cove Sturtevant [aut, cre],
Stefan Metzger [aut],
Natchaya Pingintha-Durden [aut],
David Durden [aut]

Maintainer Cove Sturtevant <eddy4R.info@gmail.com>

RemoteType local

RemoteUrl /home/eddy/NEON-FIU-algorithm/ext/eddy4R/pack/eddy4R.qaqc

RemoteSha 0.2.8

R topics documented:

`def.agr.file.dp00` *Definition function: Combine individual L0 data streams from file*

Description

Definition function. Combine (and regularize) L0 data streams downloaded in individual files from the L0 sandbox tool.

Usage

```
def.agr.file.dp00(dirFile = ".", nameFile, nameVar, unitVar, Freq,
  FmtTime = "%d-%b-%Y %I.%M.%OS %p", Tz = "GMT")
```

Arguments

`dirFile`	Optional. A character string indicating the directory in which the files to combine are stored. Default is the current working directory.
`nameFile`	Required. A character vector of file names holding the L0 data streams to combine. Must be in .csv format. Within these files, the first column must containing the measurement time, the second column must contain one L0 data stream
`nameVar`	Required. A character vector of the same length as nameFile with the variable names of the L0 data streams
`unitVar`	Required. A character vector of the same length as nameFile with the variable units
`Freq`	Required. A numeric value indicating the expected frequency [Hz] of L0 data within the files in nameFile
`FmtTime`	Optional. An character format string to interpret the time values in the first column of each file in nameFile. Default is %d-%b-%Y %I.%M.%OS %p
`Tz`	Optional. A character string specifying the time zone in with the time values in the first column of each file in nameFile are represented. Default is GMT

Value

A list of:
`time` a POSIXlt vector of regularized times corresponding to each row in data. Limits are the min and max of times found within nameFile
`data` a named data frame containing the regularized time series of L0 variables found within nameFile

Author(s)

Cove Sturtevant <`eddy4R.info@gmail.com`>

References

License: GNU AFFERO GENERAL PUBLIC LICENSE Version 3, 19 November 2007

See Also

Currently none

Examples

```
Currently none
```

`def.conv.qf.not.vrbs` *Definition function: Convert from verbose to non-verbose output of quality tests*

Description

Function definition. Convert from the verbose output of quality test algorithms (actual quality flag values) to the non-verbose option (vector positions of failed and na test results).

Usage

```
def.conv.qf.not.vrbs(qf)
```

Arguments

`qf`	A list of variables, each containing a data frame of quality flags for that variable.

Value

A named list of variables matching those in `qf`, each itself a named list of flags (standard or user-defined names), with nested lists of $fail and $na vector positions of failed and na quality test results for that variable and flag (eg. posQf$X$posQfStep$fail and posQf$Y$posQfStep$na).

Author(s)

Cove Sturtevant <`eddy4R.info@gmail.com`>

References

License: GNU AFFERO GENERAL PUBLIC LICENSE Version 3, 19 November 2007

See Also

def.plau def.conv.qf.vrbs

Examples

```
qf <- list(var01=data.frame(qfRng = c(0,1,-1,0,1,0,0,0,0,-1),qfStep = c(0,0,-1,-1,0,0,0,0,0,-1))) # Verbose outpu
posQf <- def.conv.qf.not.vrbs(qf=qf) # Convert to vector positions of failed and na quality tests
```

`def.conv.qf.vrbs` *Definition function: Convert from non-verbose to verbose output of quality tests*

Description

Function definition. Convert from the non-verbose option (vector positions of failed and na test results) of quality test algorithms to the verbose output (actual quality flag values).

Usage

```
def.conv.qf.vrbs(posQf, numRow)
```

Arguments

posQf	A named list of variables, each itself a named list of flags (standard or user-defined names), with nested lists of $fail and $na vector positions of failed and na quality tests for that variable and flag (eg. posQf$X$posQfStep$fail and posQf$Y$posQfStep$na).
numRow	A single integer indicating the number of rows in the original data from which the quality test results in `posQf` were derived

Value

A list of variables matching those in `posQf`, each containing a data frame of quality flags for that variable. Number of rows match that of `numRow`

Author(s)

Cove Sturtevant <`eddy4R.info@gmail.com`>

References

License: GNU AFFERO GENERAL PUBLIC LICENSE Version 3, 19 November 2007

See Also

def.plau def.conv.qf.not.vrbs

Examples

```
posQf <- list(var01=list(qfRng = list(fail=c(2,5),na=c(3,10)),qfStep = list(fail=numeric(0),na=c(3,4,10)))) # non
qf <- def.conv.qf.vrbs(posQf=posQf,numRow=10) # Convert to quality flag values
```

`def.dspk.br86` *Definition function: Median filter de-spiking*

Description

Function definition. Median filter de-spiking
After Brock (1986), Starkenburg et al. (2014);
Order N = 3 - 4 (i.e., a window of 7 - 9 points);
Such a filter will be sensitive to spikes of up to 3 - 4 consecutive points (Brock, 1986); this is the maximum number of points typically allowed to be considered spikes (Vickers and Mahrt, 1997; Mauder and Foken, 2011); Raw signal is normalized so that its mean is zero and standard deviation is 1;
Initial number of bins is 2, and doubles iteratively until minima are found;
Distribution of the differences (D1) between the filtered signal and the raw signal is assessed; good data points will be centered within the histogram near zero; spikes will lie farthest from the center, resulting in subpopulations that make the distribution multi-modal;
Determine the the range of accepted values, DT by searching the histogram for the first minima from the center; Points where |D1| > DT are considered spikes, provided the difference is >= 10 x the measurement resolution (smallest detected change);
Error escapes:
-1: no non-NAs in dataset
-2: singular or constant value
-3: measurement changes so slow that time-series and filter value are identical

Usage

```
def.dspk.br86(dataIn, WndwFilt = 9, NumBin = 2, ThshReso = 10,
  FracRealMin = 0.025)
```

Arguments

`dataIn`	Required. A univariate vector of integers or numerics of Input data
`WndwFilt`	Optional. A single integer value of filter width. Default = 9
`NumBin`	Optional. A single integer value of the initial number/step size of histogram bins. Default = 2
`ThshReso`	Optional. A single integer value of the resolution threshold for spike determination. Only considered spike only if difference larger than measurement resolution x ThshReso. Default = 10

`FracRealMin`	Optional. A single numeric value of the minimum fraction of non-NA values required to perform median filter despiking. Default = 0.025 (2.5%)

Value

A list of:

`dataIn` Same as input.

`WndwFilt` Same as input.

`NumBin` Same as input.

`ThshReso` Same as input.

`FracRealMin` Same as input.

`numBinFinl` Integer. The final number of histogram bins used.

`numSpk` Integer. The number of spikes identified.

`dataOut` Numeric vector of input data with spikes removed.

`dataNorm` Numeric vector of input data normalized to mean 0 and standard deviation of 1

`dataNormDiff` Numeric vector of differences between subsequent values of `dataNorm`. Length = 1-length(dataIn)

`resoDataNorm` Numeric value. Measurement resolution (assumed to be smallest recorded change in `dataNorm`)

`thshNumData` Numeric value. Minimum number of non-NAs in window to calculate median, otherwise NA is returned.

`dataNormFiltMed` Numeric vector. Median-filtered timeseries of `dataNorm`.

`histDiff` Numeric vector. Initial histogram of differences of `dataNormFiltMed`

`crit` Logical. Criteria for ending iteration over histogram bins

`locBin` Numeric vector. Bin edges for `histDiff`

`histDiffFinl` Numeric vector. Final histogram of differences of `dataNormFiltMed` using bin edges in `locBin`

`histDiffFinl$counts` Integer vector. Counts within each bin of `histDiffFinl`

`histDiffFinl$breaks` Numeric vector. Bin edges for `histDiffFinl`

`posBinMin` Integer. Index of histogram bin within `histDiffFinl` with minimum number of counts.

`posBinMax` Integer. Index of histogram bin within `histDiffFinl` with maximum number of counts.

`posThshBinMin` Integer. Current iteration of minimum threshold index of histogram bin within `histDiffFinl` for spike determination

`posThshBinMax` Integer. Current iteration of maximum threshold index of histogram bin within `histDiffFinl` for spike determination

`posThshBinMinFinl` Integer. Final minimum threshold index of histogram bin within `histDiffFinl` for spike determination

`posThshBinMaxFinl` Integer. Final maximum threshold index of histogram bin within `histDiffFinl` for spike determination

`posSpk` Indices of determined spikes within `dataIn`

`qfSpk` Integer. Quality flag values [-1,0,1] for despike test, matching same size as `dataIn`

Author(s)

Stefan Metzger <`eddy4R.info@gmail.com`>
Cove Sturtevant <`eddy4R.info@gmail.com`>

References

License: GNU AFFERO GENERAL PUBLIC LICENSE Version 3, 19 November 2007.
Brock, F. V. A Nonlinear Filter to Remove Impulse Noise from Meteorological Data. J. Atmos. Oceanic Technol. 3, 51–58 (1986).
Starkenburg, D. et al. Assessment of Despiking Methods for Turbulence Data in Micrometeorology. J. Atmos. Oceanic Technol. 33, 2001–2013 (2016).
Vickers, D. & Mahrt, L. Quality control and flux sampling problems for tower and aircraft data. in 512–526 (Amer Meteorological Soc, 1997).
Mauder, M. and Foken, T. Documentation and instruction manual of the edy covariance software package TK3. Arbeitsergebn. Univ Bayreuth. Abt Mikrometeorol. ISSN 1614-8916, 46:58 pp. (2011) NEON Algorithm Theoretical Basis Document: Quality Flags and Quality Metrics for TIS Data Products (NEON.DOC.001113)

See Also

Currently none

Examples

```
Currently none
```

`def.dspk.wndw` *Definition function: Determine spike locations using window-based statistics*

Description

Function definition. Determines spike locations based on window-based Gaussian statistics (arithmetic mean and standard deviation) and distribution statistics (median, median absolute deviation).

Usage

```
def.dspk.wndw(data, Trt = list(AlgClss = c("mean", "median")[2], NumPtsWndw =
  c(11, 101)[2], NumPtsSlid = 1, ThshStd = c(3.5, 20)[2], NaFracMax = 0.1, Infl
  = 0, IterMax = Inf, NumPtsGrp = c(4, 10)[2], NaTrt = c("approx", "omit")[2]),
  Cntl = list(NaOmit = c(TRUE, FALSE)[2], Prnt = c(TRUE, FALSE)[1], Plot =
  c(TRUE, FALSE)[1]), Vrbs = FALSE)
```

Arguments

`data`	Required input. A data frame or matrix containing the data to be evaluated.
`Trt`	Optional. A list of the following parameters specifying the details of the despike algorithm. `AlgClss`: string. de-spiking algorithm class ["mean" or "median"] `NumPtsWndw`: integer. window size [data points] (must be odd for median / mad) `NumPtsSlid`: integer. window sliding increment [data points] `ThshStd`: number. threshold for detecting data point as spike [sigma / MAD_sigma] `NaFracMax`: The maximum allowable proportion of NA values per window in which spikes can be reliably determined `Infl`: number. inflation per iteration [fraction of sigma / MAD_sigma] `IterMax`: number or Inf. maximum number of iterations `NumPtsGrp`. integer. minimum group size that is not considered as consecutive spikes [data points] `NaTrt`. string. spike handling among iterations ["approx" or "omit"]
`Cntl`	Optional. A list of the following parameters specifying other implementation details `NaOmit`: Logical. delete leading / trailing NAs from dataset? `Prnt`: Logical. print results? `Plot`: Logical. plot results?
`Vrbs`	Optional. Option to output the quality flags (same size as `data`) rather than vector positions of failed and na values. Default = FALSE

Value

A list of the following:
`data`: the despiked data matrix
`smmy`: a summary of the despike algorithm results, including iterations (`iter`), determined spikes (`news`), and total resultant NAs (`alls`)
And one of the following: `posSpk`: output if `Vrbs` is FALSE. A list of each input variable, with nested lists of $posQfSpk$fail and $posQfSpk$na vector positions of determined spikes and 'cannot evaluate' positions, respectively. This output format is identical to def.plau `qfSpk`: output if `Vrbs` is TRUE. A data frame the same size as data with the quality flag values [-1,0,1] for each input variable

Author(s)

Stefan Metzger <`eddy4R.info@gmail.com`>
Cove Sturtevant <`eddy4R.info@gmail.com`>

References

License: GNU AFFERO GENERAL PUBLIC LICENSE Version 3, 19 November 2007.
Hojstrup, J.: A statistical data screening procedure, Meas. Sci. Technol., 4, 153-157, doi:10.1088/0957-0233/4/2/003, 1993.
Metzger, S., Junkermann, W., Mauder, M., Beyrich, F., Butterbach-Bahl, K., Schmid, H. P., and Foken, T.: Eddy-covariance flux measurements with a weight-shift microlight aircraft, Atmos. Meas. Tech., 5, 1699-1717, doi:10.5194/amt-5-1699-2012, 2012.

Vickers, D., and Mahrt, L.: Quality control and flux sampling problems for tower and aircraft data, J. Atmos. Oceanic Technol., 14, 512-526, doi:10.1175/1520-0426(1997)014<0512:QCAFSP>2.0.CO;2, 1997.

See Also

Currently none

Examples

```
Currently none
```

`def.neon.dp01.qf.grp` *Definition function: Grouping the quality flags for each of NEON ECTE and ECSE L1 data product*

Description

Function definition. Grouping the quality flags of each NEON ECTE and ECSE L1 data product into a single dataframe for further use in the calculation of Alpha, Beta, and Final flag.

Usage

```
def.neon.dp01.qf.grp(qfInput = list(), MethMeas = c("ecte", "ecse")[1],
  TypeMeas = c("samp", "vali")[1], dp01 = c("envHut", "co2Turb", "h2oTurb",
  "co2Stor", "h2oStor", "isoCo2", "isoH2o", "soni", "amrs", "tempAirLvl",
  "tempAirTop")[1], idGas = NULL)
```

Arguments

`qfInput`	A list of data frame containing the input quality flag data that related to L1 data products are being grouped. Of class integer". [-]
`MethMeas`	A vector of class "character" containing the name of measurement method (eddy-covariance turbulent exchange or storage exchange), MethMeas = c("ecte", "ecse"). Defaults to "ecse". [-]
`TypeMeas`	A vector of class "character" containing the name of measurement type (sampling or validation), TypeMeas = c("samp", "ecse"). Defaults to "samp". [-]
`dp01`	A vector of class "character" containing the name of NEON ECTE and ECSE L1 data products which the flags are being grouped, c("envHut", "co2Turb", "h2oTurb", "isoCo2", "isoH2o", "soni", "amrs", "tempAirLvl", "tempAirTop"). Defaults to "co2Turb". [-]
`idGas`	A data frame contianing gas ID for isoCo2 measurement. Need to provide when dp01 = "isoCo2". Default to NULL. [-]

Value

A list of data frame of the quality flags related to that sub-data product.

Author(s)

Natchaya Pingintha-Durden <ndurden@battelleecology.org>

References

License: GNU AFFERO GENERAL PUBLIC LICENSE Version 3, 19 November 2007

See Also

Currently none

Examples

```
#generate the fake quality flags for each sensor
TimeBgn <- "2016-04-24 02:00:00.000"
TimeEnd <- "2016-04-24 02:29:59.950"
qf <- list()
qf$irgaTurb <- eddy4R.qaqc::def.qf.ecte(TimeBgn = TimeBgn, TimeEnd = TimeEnd, Freq = 20, Sens = "irgaTurb", PcntQf =
#add qfIrgaTurbvali
qf$irgaTurb$qfIrgaTurbVali <- rep(0, 86000)
qf$mfcSampTurb <- eddy4R.qaqc::def.qf.ecte(TimeBgn = TimeBgn, TimeEnd = TimeEnd, Freq = 20, Sens = "mfcSampTurb", P
qf$soni <- eddy4R.qaqc::def.qf.ecte(TimeBgn = TimeBgn, TimeEnd = TimeEnd, Freq = 20, Sens = "soni", PcntQf = 0.05)
qf$amrs <- eddy4R.qaqc::def.qf.ecte(TimeBgn = TimeBgn, TimeEnd = TimeEnd, Freq = 40, Sens = "amrs", PcntQf = 0.05)

#grouping the set of the flags
qfGrpCo2Turb <- eddy4R.qaqc::def.neon.dp01.qf.grp(qfInput = qf, MethMeas = "ecte", TypeMeas = "vali", dp01="co2Tur
qfGrpSoni <- eddy4R.qaqc::def.neon.dp01.qf.grp(qfInput = qf, MethMeas = "ecte", TypeMeas = "samp", dp01="soni")
```

`def.plau` *Definition function: Plausibility tests (Range, Step, Persistence, Null, Gap)*

Description

Function definition. Determines inplausible data indices based on user-specified limits for the data range, step between adjacent values, persistence (similarity of adjacent values), nulls, and gaps.

Usage

```
def.plau(data, time = as.POSIXlt(seq.POSIXt(from = Sys.time(), by = "sec",
  length.out = length(data[, 1]))), RngMin = apply(data, 2, min, na.rm =
  TRUE), RngMax = apply(data, 2, max, na.rm = TRUE),
  DiffStepMax = apply(abs(apply(data, 2, diff)), 2, max, na.rm = TRUE),
  DiffPersMin = rep.int(0, length(data)), WndwPers = 60 *
  median(abs(diff(time)), na.rm = TRUE) * rep.int(1, length(data)),
  TestNull = rep(FALSE, length(data)), NumGap = rep(length(data[, 1]) + 1,
  length(data)), Vrbs = FALSE)
```

Arguments

`data`	Required input. A data frame containing the data to be evaluated (do not include the time stamp vector here).
`time`	Optional. A time vector of class POSIXlt of times corresponding with each row in data. Defaults to an evenly spaced time vector starting from system time of execution by seconds.
`RngMin`	Optional. A numeric vector of length equal to number of variables in data containing the minimum acceptable value for each variable. Defaults to observed minimums (no flags will result)
`RngMax`	Optional. A numeric vector of length equal to number of variables in data containing the maximum acceptable value for each variable. Defaults to observed maximums (no flags will result)
`DiffStepMax`	Optional. A numeric vector of length equal to number of variables in data containing the maximum acceptable absolute difference between sequential data points for each variable. Defaults to observed maximum (no flags will result)
`DiffPersMin`	Optional. A numeric vector of length equal to number of variables in data containing the minimum absolute change in value over the interval specified in WndwPers to indicate the sensor is not "stuck". Defaults to a vector of zeros (no flags will result).
`WndwPers`	Optional. The time window for evaluting the persistence test. This must be a vector of length equal to number of variables in data. If the values are numeric (integer), then WndwPers specifies number of data points over which to test for the minimum absolute change in value specified in DiffPersMin. If the vector is a difftime object (e.g. as.difftime(5,units="secs")), it specifies the time interval over which to test for the minimum absolute change in value specified in DiffPersMin. The results are the same if the time-based window exactly corresponds to an integer number of data points. Defaults to a difftime object of 60 x median observed time difference.
`TestNull`	Optional. Apply the null test? A logical vector of [TRUE or FALSE] of length equal to number of variables in data. Defaults to FALSE (no null values are flagged)
`NumGap`	Optional. A numeric value >= 1, interpretable as an integer, specifying the numer of consecutive NA values constituting a gap. Default is the one more than the length of the data series (no gaps will be flagged)
`Vrbs`	Optional. A logical FALSE/TRUE value indicating whether to: `Vrbs = FALSE`: (Default) output the vector positions of the fail and na results for each test (default), or `Vrbs = TRUE`: output a data frame for each variable in data, with a column for each plausibility test outputting the actual quality flags [-1,0,1] for each data point

Value

If:
`Vrbs = FALSE` A list of variables in `data`, nested within each a list of qf test names: (`posQfRng`), Step (`posQfStep`), Persistence (`posQfPers`), Null (`posQfNull`), and Gap(`posQfGap`) tests. Each flag is itself a nested list of failed and na (unable to eval) flagged vector positions (e.g. `$X$posQfRng$fail` and `$X$posQfRng$na`
`Vrbs = TRUE` A list of variables matching those in `data`, each containing a data frame with a column for each plausibility test outputting the actual quality flags [-1,0,1] for each data point, where -1 indicates the test could not be evaluated, 0 indicates a pass, and 1 indicates a fail

Author(s)

Cove Sturtevant <`eddy4R.info@gmail.com`>

References

License: GNU AFFERO GENERAL PUBLIC LICENSE Version 3, 19 November 2007
NEON Algorithm Theoretical Basis Document QA/QC Plausibility Testing (NEON.DOC.011081)

See Also

Currently none

Examples

```
data <- data.frame(x=rnorm(1000,mean=0,sd=1)) # Start off with a vector of 1000 random values
data$x[c(20,50,500,90)] <- 50 # insert some spikes
data$x[600:699] <- rnorm(100,mean=0,sd=0.001) # Add some "stuck" data
data$x[800:810] <- NA
RngMin <- -4
RngMax <- 4
DiffStepMax <- 6
DiffPersMin <- 0.1
WndwPers <- as.difftime(10,units="secs") # We are using the default time variable, which generates a freq of 1 secon
TestNull <- TRUE
NumGap <- 11
Vrbs=TRUE
qf <- eddy4R.qaqc::def.plau(data=data,RngMin=RngMin,RngMax=RngMax,DiffStepMax=DiffStepMax,DiffPersMin=DiffPersM
```

`def.plot.dp01.qfqm` *Definition function: Plot quality flags and quality metrics (basic L1 data products)*

Description

Function definition. Plots the aggregated quality flags, quality metrics and final quality flag for basic L1 (time window averaged) data products as output from wrap.dp01.qfqm.R.

Usage

```
def.plot.dp01.qfqm(dataDp01, WndwTime = c(min(dataDp01$timeAgrBgn),
  max(dataDp01$timeAgrBgn)), NameQmPlot = sub("Pass", "",
  names(dataDp01$dataAgr[[1]][grep("Pass", names(dataDp01$dataAgr[[1]]))])))
```

Arguments

`dataDp01`	Required input. A list output from wrap.dp01.qfqm.R of: timeAgrBgn - the starting time stamp of aggregated L1 data and quality metrics timeAgrEnd - the ending time stamp (non-inclusive) of aggregated L1 data and quality metrics dataAgr - a list of named variables, each containing a data frame of the time-aggregated mean, minimum, maximum, variance, number of points going into the average, standard error of the mean, and quality metrics (pass, fail, NA) pertaining to that variable for each flag in flgs, as well as the alpha & beta quality metrics and final quality flag. It is important that the column names of this data frame are indistinguishable from those that would be created from wrap.dp01.qfqm.R
`WndwTime`	Optional. A 2-element POSIXlt vector of the minimum and maximum time range to plot. Default is the entire data range.
`NameQmPlot`	Optional. A character vector listing the individual quality metrics to plot. The strings in this vector can be partial names, i.e. a partial match will result in the quality metric being plotted (ex. NameQmIndiv <- c("Step") will result in the quality metrics "qmStepPass","qmStepFail" and "qmStepNa" to be plotted). Default is all QMs.

Value

Running this function will output 3 plots per data variable: 1) basic L1 statistics (mean, min, max, etc). 2) Pass, Fail, and NA quality metrics for ever flag, 3) the final Alpha and Beta quality metrics and the final quality flag.

Author(s)

Cove Sturtevant <`eddy4R.info@gmail.com` >

References

License: GNU AFFERO GENERAL PUBLIC LICENSE Version 3, 19 November 2007.
NEON Algorithm Theoretical Basis Document: Quality Flags and Quality Metrics for TIS Data Products (NEON.DOC.001113)

See Also

Currently none

Examples

```
Currently none
```

`def.qf.amrs` *Definition function: AMRS flags for XSens*

Description

Definition function to interpret the AMRS (Attitude and Reference Motion System) sensor flags from `diag32`. Flags output are a reduced set that were deemed important for the NEON QFQM framework and described in the NEON.DOC.000807.

Usage

```
def.qf.amrs(diag32, MethQf = c("qfqm", "xsen")[1])
```

Arguments

`diag32`	The 32-bit diagnostic stream that is output from the XSens AMRS.
`MethQf`	Switch for quality flag determination for the XSens AMRS, diag32 provides ones for passing quality by default the equals "xsen". The "qfqm" method changes the ones to zeros to match the NEON QFQM logic for certain flags that are set high when good, qfAmrsVal & qfAmrsFilt.

Value

A dataframe (`qfSoniAmrs`) of sensor specific AMRS quality flags as described in NEON.DOC.000807.

Author(s)

Dave Durden <`ddurden@battelleecology.org`>

References

License: GNU AFFERO GENERAL PUBLIC LICENSE Version 3, 19 November 2007.
NEON Algorithm Theoretical Basis Document:Eddy Covariance Turbulent Exchange Subsystem Level 0 to Level 0' data product conversions and calculations (NEON.DOC.000807) XSens AMRS reference manual

See Also

Currently none

Examples

```
diag32 <- as.integer(rep(135, 72000))
def.qf.amrs(diag32 = diag32)

pos <- runif(15,1, 72000) # inserting error positions for other flags
diag32[pos] <- as.integer(c(262279,524423, 1048710)) # filling with numbers that would indicate flags for qfAmrsVal
eddy4R.qaqc::def.qf.amrs(diag32 = diag32)
```

def.qf.ecte *Definition function: Create fake flags for ECTE sensors*

Description

Workflow. File to create fake flags for soni to test the QAQC test

Usage

```
def.qf.ecte(TimeBgn, TimeEnd, Freq = 20, Sens = c("soni", "irga",
  "irgaMfcSamp", "soniAmrs")[1], PcntQf = 0.05)
```

Arguments

`TimeBgn`	is the beginning time of the period to generate flags.
`TimeEnd`	is the end time of the period to generate flags.
`Freq`	is the measurement frequency used to generate flags.
`Sens`	is ECTE sensor assembly name for which the flags are being generated.
`PcntQf`	is percentage of observations that should be flagged.

Value

Currently none.

Author(s)

David Durden <`ddurden@battelleecology.org`>

References

License: GNU AFFERO GENERAL PUBLIC LICENSE Version 3, 19 November 2007.

See Also

Currently none

Examples

```
TimeBgn <- "2016-04-24 02:00:00.000"
TimeEnd <- "2016-04-24 02:29:59.950"
Freq <- 20
Sens <- "soni"
PcntQf <- 0.05
qfSens <- def.qf.ecte(TimeBgn = TimeBgn, TimeEnd = TimeEnd, Freq = Freq, Sens = Sens, PcntQf = PcntQf)
```

def.qf.finl *Definition function: Final Quality Flag*

Description

Function definition. Determine the final quality flag for individual level 1 data product following the method described in Smith et.al. (2014). The alpha and beta quality flags and metrics are also computed in this function. Performed for the entire set of input data.

Usage

```
def.qf.finl(qf, WghtAlphBeta = c(2, 1), Thsh = 0.2)
```

Arguments

qf	A data frame of quality flags, class integer, from which to compute the final quality flag. Each column contains the quality flag values [-1,0,1] for that flag. Note: This is the Vrbs output from def.plau, def.dspk.wndw, and def.dspk.br86. See def.conv.qf.vrbs for converting from non-verbose to verbose output.
WghtAlphBeta	A 2-element integer vector of weights to apply to the alpha and beta quality metrics, which will then be summed and evaluated against the threshold (Thsh) for determing the final quality flag. Default to WghtAlphBeta=c(2,1). [-]
Thsh	Threshold for determine the condition (pass = 0 or failed = 1) of final quality flag. Default to 0.2 (0.2 percent). [fraction]

Value

A list of:
qaqcRpt A dataframe containing alpha and beta quality flags at the same frequency as input qf. [-]
qfqm A dataframe containing alpha and beta quality metric and final qualiy flag. [-]

Author(s)

Cove Sturtevant <eddy4R.info@gmail.com>
Natchaya Pingintha-Durden <ndurden@battelleecology.org>

References

License: GNU AFFERO GENERAL PUBLIC LICENSE Version 3, 19 November 2007.
NEON Algorithm Theoretical Basis Document: Quality Flags and Quality Metrics for TIS Data Products (NEON.DOC.001113)
Smith, D.E., Metzger, S., and Taylor, J.R.: A transparent and transferable framework for tracking quality information in large datasets. PLoS ONE, 9(11), e112249.doi:10.1371/journal.pone.0112249, 2014.

See Also

`wrap.dp01.qfqm`
`def.qm`

Examples

```
qfA <- c(0,0,0,0,0,1,1,1,1,1,0,0,0,0,1)
qfB <- c(0,0,-1,-1,-1,1,1,0,0,0,0,0,0,0,0)
qfC <- c(0,1,1,0,0,0,0,0,0,0,0,0,-1,-1,-1)
test<-list()
test$qf <- data.frame(qfA,qfB,qfC)
out <- def.qf.finl(qf=test$qf, WghtAlphBeta=c(2,1), Thsh=0.2)
```

`def.qf.irga` *Definition function: irga flags for LI72000*

Description

Definition function to interpret the irga sensor flags from `diag01`.

Usage

```
def.qf.irga(diag01, MethQf = c("qfqm", "lico")[1])
```

Arguments

`diag01`	The 32-bit diagnostic stream that is output from the LI7200.
`MethQf`	Switch for quality flag determination for the LI7200, diag01 provides ones for passing quality by default the equals "lico". The "qfqm" method changes the ones to zeros to match the NEON QFQM logic.

Value

A dataframe (`qfIrgaTurb`) of sensor specific irga quality flags as described in NEON.DOC.000807.

Author(s)

Dave Durden <`ddurden@battelleecology.org`>

References

License: GNU AFFERO GENERAL PUBLIC LICENSE Version 3, 19 November 2007.
NEON Algorithm Theoretical Basis Document:Eddy Covariance Turbulent Exchange Subsystem Level 0 to Level 0' data product conversions and calculations (NEON.DOC.000807)
Licor LI7200 reference manual

See Also

Currently none

Examples

```
diag01 <- rep(8191, 72000)
def.qf.irga(diag01 = diag01)
```

---

def.qf.irga.agc *Definition function: Signal strength flag for IRGA*

Description

Definition function to generate the signal strength flags for the IRGA from the diagnostic output quality metric `qfIrgaAgc`. Flag indicating when the sensor is operating with low signal strength using 50 percent as the default threshold (1 = when qfIrgaAgc <= 0.50, 0 = when qfIrgaAgc >= 0.50, -1 = NA).

Usage

```
def.qf.irga.agc(qfIrgaAgc, critThsh = 0.5)
```

Arguments

`qfIrgaAgc`	The quality metric derived from the IRGA diagnostics to determine signal strength. Presented as a dimensionless fraction. [-]
`critThsh`	The critical threshold value for the `qfIrgaAgc` value to throw the flag for low signal strength (defaults to 0.50 or 50 percent).

Value

A vector class of integer (`qfIrgaAgcOut`) of IRGA AGC flags. Flag indicating when the sensor is operating with low signal strength using 50 percent as the default threshold (1 = when qfIrgaAgc <= 0.50, 0 = when qfIrgaAgc >= 0.50, -1 = NA). [-]

Author(s)

David Durden <`ddurden@battelleecology.org`>

References

License: GNU AFFERO GENERAL PUBLIC LICENSE Version 3, 19 November 2007.

See Also

Currently none

Examples

```
qfIrgaAgc <- rnorm(10,0.70,0.15)
qfIrgaAgc[c(2,8)] <- NA
attributes(qfIrgaAgc)$unit <- "-"

qfIrgaAgcOut <- def.qf.irga.agc(qfIrgaAgc = qfIrgaAgc)
```

`def.qf.irga.vali` *Definition function: Validation flag for IRGA*

Description

Definition function to generate the validation flags for IRGA from the IRGA sampling mass flow controller flow rate set point or from the IRGA validation soleniod valves. Flag indicating when the sensor is operated under validation period (1 = validation period, 0 = normal operating condition, -1 = NA).

Usage

```
def.qf.irga.vali(data, Sens = c("irgaMfcSamp", "irgaSndValiNema")[1])
```

Arguments

`data`	A data.frame containing the L0p input IRGA sampling mass flow controller data or the IRGA validation soleniod valves data at native resolution. Of type numeric or integer. [User-defined]
`Sens`	A vector of class "character" containing the name of sensor used to determine the flag (IRGA sampling mass flow controller data or the IRGA validation soleniod valves), Sens = c("irgaMfcSamp", "irgaSndValiNema"). Defaults to "irgaMfc-Samp". [-]

Value

A vector class of integer (`qfIrgaVali`) of IRGA validation flags. Flag indicating when the sensor is operated under validation period (1 = validation period, 0 = normal operating condition, -1 = NA) [-]

Author(s)

Natchaya Pingintha-Durden <`ndurden@battelleecology.org`>
David Durden <`ddurden@battelleecology.org`>

References

License: GNU AFFERO GENERAL PUBLIC LICENSE Version 3, 19 November 2007.

See Also

Currently none

Examples

```
data <- list()
#generate test data for irgaMfcSamp
data$irgaMfcSamp <- data.frame(frt = c(0.0003621, 0.00035066, NA, NA, 0, 0),
frt00 = c(0.00020, 0.0001983, NA, NA, 0, 0),
frtSet00 = c(0.00020, 0.00020, NA, NA, 0, 0),
presAtm = c(57200, 58500, NA, NA, 58600, 58600),
temp = c(300, 301, NA, NA, 299, 300))
#generate test data for irgaSndValiNema
data$irgaSndValiNema <- data.frame(qfGas01 = c(rep(0, 25)),
qfGas02 = c(rep(0, 5), rep(1, 5), rep(0, 15)),
qfGas03 = c(rep(0, 10), rep(1, 5), rep(0, 10)),
qfGas04 = c(rep(0, 15), rep(1, 5), rep(0, 5)),
qfGas05 = c(rep(0, 20), rep(NA, 5)))
#determine the flag using irgaMfcSamp
data$qfqm$irga$qfIrgaVali <- def.qf.irga.vali(data = data$irgaMfcSamp, Sens = "irgaMfcSamp")
#determine the flag using irgaSndValiNema
data$qfqm$irga$qfIrgaVali <- def.qf.irga.vali(data = data$irgaSndValiNema, Sens = "irgaSndValiNema")
```

`def.qf.rmv.data` *Definition function: to remove high frequency data points that have failed quality flags*

Description

Definition function to remove high frequency data points that have failed quality flags from a data.frame

Usage

```
def.qf.rmv.data(dfData, dfQf, Sens = NULL, Vrbs = FALSE,
  TypeData = c("integer", "real")[1])
```

Arguments

`dfData`	Input data.frame for data to be removed from based on quality flags
`dfQf`	Input data.frame of quality flags (must be of class integer to be included in the processing)
`Sens`	Character string indicating the sensor the high frequency data come from to check for sensor specific flags
`Vrbs`	Optional. A logical FALSE/TRUE value indicating whether to: `Vrbs = FALSE`: (Default) cleaned data set, list of variables assessed, list of quality flags for each variable assessed, and the total number of bad data per

	variable, or `Vrbs = TRUE`: cleaned data set, list of variables assessed, list of quality flags for each variable assessed, the number of each quality flag tripped for each variable and the total number of bad data per variable
`TypeData`	Type of input data, TypeData = c("integer", "real"). Defaults to "integer". [-].

Value

A list (`rpt`) containing a dataframe (`rpt$dfData`) of the data with bad data replaced by NaN's, a vector of data variables to assess (`rpt$listVar`), a list containing vectors of flag names used for each variable (`rpt$listQf`), a list containing vectors of the number of quality flags set high for each variable (`rpt$fracQfBad`), a list containing vectors of flagged data points for each variable (`rpt$posBad`), a list containing total number of flagged data points for each variable (`rpt$numBadSum`).

Author(s)

Dave Durden <`ddurden@battelleecology.org`>

References

License: GNU AFFERO GENERAL PUBLIC LICENSE Version 3, 19 November 2007.
NEON Algorithm Theoretical Basis Document:Eddy Covariance Turbulent Exchange Subsystem Level 0 to Level 0' data product conversions and calculations (NEON.DOC.000807)

See Also

Currently none

Examples

```
Currently none
```

`def.qf.soni` *Definition function: Soni flags for CSAT3*

Description

Definition function to interpret the Soni (Campbell Sci. CSAT3) sensor flags from `diag16`. Flags output are important for the NEON QFQM framework and described in the NEON.DOC.000807. This code was built for flags outlined in firmware version 3 of the CSAT3; however, flag detemination may be transferable to other firmware versions.

Usage

```
def.qf.soni(diag16)
```

Arguments

`diag16`	The 16-bit diagnostic stream that is output from the Soni (CSAT3).

Value

A dataframe (`qfSoni`) of sensor specific AMRS quality flags as described in NEON.DOC.000807.

Author(s)

Dave Durden <`ddurden@battelleecology.org`>

References

License: GNU AFFERO GENERAL PUBLIC LICENSE Version 3, 19 November 2007.
NEON Algorithm Theoretical Basis Document:Eddy Covariance Turbulent Exchange Subsystem Level 0 to Level 0' data product conversions and calculations (NEON.DOC.000807)
Campbell Scientific CSAT3 reference manual

See Also

Currently none

Examples

```
diag16 <- as.integer(rep(135, 36000))
pos <- runif(20,1, 36000) # inserting error positions for other flags
diag16[pos] <- as.integer(c(32768,16384,8192,4096,61442, 61441,61440,61503, -99999, NaN)) # filling with numbers t

eddy4R.qaqc::def.qf.soni(diag16 = diag16)
```

`def.qm` *Definition function: Quality Metrics*

Description

Function definition. Determine the quality metrics of failed, pass, and NA for each of the individual quality flag following the method described in Smith et.al. (2014). Performed for the entire set of input data.

Usage

```
def.qm(qf, nameQmOut = NULL)
```

Arguments

`qf`	A data frame of quality flags, class integer. Each column contains the quality flag values [-1,0,1] for that flag. Note: This is the Vrbs output from def.plau, def.dspk.wndw, and def.dspk.br86. See def.conv.qf.vrbs for converting from non-verbose to verbose output.
`nameQmOut`	Optional. A vector of class "character" containing the base name of the output quality metrics for each flag in `qf`. These names will be ammended with "Pass", "Fail", or "Na" at the end when outputting their respective quality metrics. Default behavoir is to autoassign names based on the column names of `qf` [-]

Value

A dataframe containing quality metrics (fractions) of failed, pass, and NA for each of the individual flag defined in `qf`.

Author(s)

Cove Sturtevant <`eddy4R.info@gmail.com`>
Natchaya Pingintha-Durden <`ndurden@battelleecology.org`>

References

License: GNU AFFERO GENERAL PUBLIC LICENSE Version 3, 19 November 2007.
NEON Algorithm Theoretical Basis Document: Quality Flags and Quality Metrics for TIS Data Products (NEON.DOC.001113)
Smith, D.E., Metzger, S., and Taylor, J.R.: A transparent and transferable framework for tracking quality information in large datasets. PLoS ONE, 9(11), e112249.doi:10.1371/journal.pone.0112249, 2014.

See Also

`wrap.qfqm.dp01` `def.plau` `def.dspk.wndw` `def.dspk.br86` `def.conv.qf.vrbs`

Examples

```
qfA <- c(0,0,0,0,0,1,1,1,1,1,0,0,0,0,1)
qfB <- c(0,0,-1,-1,-1,1,1,0,0,0,0,0,0,0,0)
qfC <- c(0,1,1,0,0,0,0,0,0,0,0,0,-1,-1,-1)
test<-list()
test$qf <- data.frame(qfA,qfB,qfC)
test$qm<- def.qm(qf=test$qf, nameQmOut=c("qmA","qmB","qmC"))
```

`wrap.dp01.qfqm`	*Wrapper function: Generate basic L1 data product, including descriptive statics, quality metrics, and final quality flag*

Description

Function wrapper. Aggregates Level 0' (calibrated raw) data and accompanying quality flags and metrics into a basic time-aggregated L1 data product, including descriptive statics, quality metrics, and final quality flag.

Usage

```
wrap.dp01.qfqm(data, time = base::as.POSIXlt(base::seq.POSIXt(from =
  base::Sys.time(), by = "sec", length.out = base::length(data[, 1]))),
  posQf = base::vector("list", base::length(data)),
  qf = base::vector("list", base::length(data)), WndwAgr = 1800 *
  stats::median(base::abs(base::diff(time)), na.rm = TRUE), TimeBgn = NULL,
  TimeEnd = NULL, NameQfExcl = base::as.list(base::character(length =
  base::length(data))))
```

Arguments

`data`	Required input. A data frame containing the L0' (calibrated raw) data evaluated (do not include the time stamp vector here).
`time`	Optional. A time vector of class POSIXlt of times corresponding with each row in data. Defaults to an evenly spaced time vector starting from system time of execution by seconds.
`posQf`	Optional. Only input ONE of either `posQf` or `qf`. # A named list of variables matching those in data, each itself a named list of flags (standard or user-defined names), with nested lists of $fail and $na vector positions of failed and na quality tests for that variable and flag (eg. posQf$X$posQfStep$fail and posQf$Y$posQfStep$na). Aggregated quality flags and metrics will be computed for all flags in this list, and all will be used to compute the final quality flag. This function directly accepts the output from def.plaus.R and def.dspk.wndw.R. Defaults to an empty flag list.
`qf`	Optional. Only input ONE of either `posQf` or `qf`. A list of variables matching those in data, each containing a data frame of quality flags for that variable. Number of rows must match that of `data`
`WndwAgr`	Optional. A difftime object of length 1 specifying the time interval for aggregating the data and flags of each variable. Defaults to 1800 x median observed time difference. Class difftime can be generated using as.difftime. If `WndwAgr` is < 1
`TimeBgn`	Optional. A POSIXlt vector of length 1 indicating the time to begin aggregating from, in windows of WndwAgr. If unspecified, defaults to the first value in time truncated to: the minute if `WndwAgr` is <= 1 min, the hour if 1 min < `WndwAgr`

	<= 1 hr, the day otherwise. Aggregation windows prior to the start of data will be removed.
`TimeEnd`	Optional. A POSIXlt vector of length 1 indicating the time to end aggregation (non-inclusive). If unspecified, defaults to the end of the last aggregation window containing data values as constructed from `WndwAgr` and `TimeBgn`
`NameQfExcl`	Optional. A list of length equal to number of variables, each itself a vector of character strings naming the flags (matching a subset of posQf or qf) for which to exclude the flagged indices of each variable from the L1 average. Eg. `NameQfExcl` <- list(x=c("posQfRng","posQfStep"),y=c("posQfPers","posQfNull","posQfGap")). Note that variables in the list must be in the same order as those in data. Defaults to an empty list (flagged points are not excluded from the L1 average).

Value

A list of:
timeAgrBgn - the starting time stamp of aggregated L1 data and quality metrics
timeAgrEnd - the ending time stamp (non-inclusive) of aggregated L1 data and quality metrics
dataAgr - a list of variables, each containing a data frame of the time-aggregated mean, minimum, maximum, variance, number of points going into the average, and quality metrics (pass, fail, NA) pertaining to that variable for each flag in posQf, as well as the alpha & beta quality metrics and final quality flag.

Author(s)

Cove Sturtevant <`eddy4R.info@gmail.com`>

References

License: GNU AFFERO GENERAL PUBLIC LICENSE Version 3, 19 November 2007.
NEON Algorithm Theoretical Basis Document: Quality Flags and Quality Metrics for TIS Data Products (NEON.DOC.001113)

See Also

Currently none

Examples

```
Currently none
```

`wrap.neon.dp01.qfqm` *Wrapper function: Calculate quality metrics, alpha and beta quality metrics, and final quality flag for the NEON eddy-covariance turbulent and stroage exchange L1 data products*

Description

Wrapper function. Calculate quality metrics, alpha and beta quality metrics, and final quality flag for the NEON eddy-covariance turbulent and storage exchange L1 data products.

Usage

```
wrap.neon.dp01.qfqm(qfInput = list(), MethMeas = c("ecte", "ecse")[1],
  TypeMeas = c("samp", "vali")[1], RptExpd = FALSE, dp01 = c("envHut",
  "co2Turb", "h2oTurb", "co2Stor", "h2oStor", "isoCo2", "isoH2o", "soni",
  "soniAmrs", "tempAirLvl", "tempAirTop")[1], idGas = NULL)
```

Arguments

`qfInput`	A list of data frame containing the input quality flag data that related to L1 data products are being grouped. Of class integer". [-]
`MethMeas`	A vector of class "character" containing the name of measurement method (eddy-covariance turbulent exchange or storage exchange), MethMeas = c("ecte", "ecse"). Defaults to "ecse". [-]
`TypeMeas`	A vector of class "character" containing the name of measurement type (sampling or validation), TypeMeas = c("samp", "ecse"). Defaults to "samp". [-]
`RptExpd`	A logical parameter that determines if the full quality metric `qm` is output in the returned list (defaults to FALSE).
`dp01`	A vector of class "character" containing the name of NEON ECTE and ECSE L1 data products which the flags are being grouped, c("envHut", "co2Turb", "h2oTurb", "isoCo2", "isoH2o", "soni", "soniAmrs", "tempAirLvl", "tempAirTop"). Defaults to "co2Turb". [-]
`idGas`	A data frame contianing gas ID for isoCo2 measurement. Need to provide when dp01 = "isoCo2". Default to NULL. [-]

Value

A list of:
`qm` A list of data frame's containing quality metrics (fractions) of failed, pass, and NA for each of the individual flag which related to L1 sub-data products if RptExpd = TRUE. [fraction]
`qmAlph` A dataframe containing metrics in a columns of class "numeric" containing the alpha quality metric for L1 sub-data products. [fraction]
`qmBeta` A dataframe containing metrics in a columns of class "numeric" containing the beta quality metric for L1 sub-data products. [fraction]
`qfFinl` A dataframe containing flags in a columns of class "numeric", [0,1], containing the final quality flag for L1 sub-data products. [-]
`qfSciRevw` A dataframe containing flags in a columns of class "numeric", [0,1], containing the scientific review quality flag for L1 sub-data products. [-]

Author(s)

Natchaya Pingintha-Durden <ndurden@battelleecology.org>
David Durden <ddurden@battelleecology.org>

References

License: GNU AFFERO GENERAL PUBLIC LICENSE Version 3, 19 November 2007.

See Also

Currently none

Examples

```
#generate the fake quality flags for each sensor
TimeBgn <- "2016-04-24 02:00:00.000"
TimeEnd <- "2016-04-24 02:29:59.950"
qf <- list()
qf$irgaTurb <- eddy4R.qaqc::def.qf.ecte(TimeBgn = TimeBgn, TimeEnd = TimeEnd, Freq = 20, Sens = "irgaTurb", PcntQf =
qf$mfcSampTurb <- eddy4R.qaqc::def.qf.ecte(TimeBgn = TimeBgn, TimeEnd = TimeEnd, Freq = 20, Sens = "mfcSampTurb", P
qf$soni <- eddy4R.qaqc::def.qf.ecte(TimeBgn = TimeBgn, TimeEnd = TimeEnd, Freq = 20, Sens = "soni", PcntQf = 0.05)
qf$soniAmrs <- eddy4R.qaqc::def.qf.ecte(TimeBgn = TimeBgn, TimeEnd = TimeEnd, Freq = 40, Sens = "soniAmrs", PcntQf =
#calculate quality metric, qmAlpha, qmBeta, qfFinl
qfqm <- list()
qfqm$co2Turb <- eddy4R.qaqc::wrap.neon.dp01.qfqm (qfInput = qf, MethMeas = "ecte", TypeMeas = "samp", dp01="co2Tur
qfqm$h2oTurb <- eddy4R.qaqc::wrap.neon.dp01.qfqm (qfInput = qf, MethMeas = "ecte", TypeMeas = "samp", dp01="h2oTur
qfqm$soni <- eddy4R.qaqc::wrap.neon.dp01.qfqm (qfInput = qf, MethMeas = "ecte", TypeMeas = "samp", dp01="soni")
# Example with expanded quality metrics included
qfqm$soniAmrs <- eddy4R.qaqc::wrap.neon.dp01.qfqm (qfInput = qf, MethMeas = "ecte", TypeMeas = "samp", RptExpd = TR
```

---

wrap.neon.dp01.qfqm.ecse

Wrapper function: Preprocessing and calculating quality metrics, alpha and beta quality metrics, and final quality flag for the NEON eddy-covariance stroage exchange L1 data products

---

Description

Wrapper function. Preprocessing and calculating quality metrics, alpha and beta quality metrics, and final quality flag for the NEON eddy-covariance stroage exchange (ECSE) Level 1 data products (dp01).

Usage

```
wrap.neon.dp01.qfqm.ecse(dp01 = c("co2Stor", "h2oStor", "tempAirLvl",
  "tempAirTop", "isoCo2", "isoH2o")[1], lvl, lvlMfcSampStor = NULL,
  lvlEnvHut = NULL, lvlValv = NULL, lvlValvAux = NULL,
  lvlCrdH2oValvVali = NULL, data = list(), qfInput = list(),
  TypeMeas = c("samp", "vali")[1], PrdMeas, PrdAgr, idxTime = list())
```

Arguments

`dp01`	A vector of class "character" containing the name of NEON ECSE dp01 which descriptive statistics are being calculated, c("co2Stor", "h2oStor", "tempAirLvl", "tempAirTop", "isoCo2", "isoH2o"). Defaults to "co2Stor". [-]
`lvl`	Measurement level of dp01 which descriptive statistics are being calculated. Of type character. [-]
`lvlMfcSampStor`	Measurement level of mfcSampStor which apply to only dp01 equal to "co2Stor" or "h2oStor". Defaults to NULL. Of type character. [-]
`lvlEnvHut`	Measurement level of envHut. Defaults to NULL. Of type character. [-]
`lvlValv`	Measurement level of irgaValvLvl, crdCo2ValvLvl, or crdH2oValvLvl. Defaults to NULL. Of type character. [-]
`lvlValvAux`	Location of valvAux which apply to only dp01 equal to "co2Stor" or "h2oStor". Defaults to NULL. Of type character. [-]
`lvlCrdH2oValvVali`	Measurement level of crdH2oValvVali which apply to only dp01 equal to "isoH2o". Defaults to NULL. Of type character. [-]
`data`	A list of data frame containing the input dp0p data that related to dp01 which qfqm are being calculated. Of class integer". [User defined]
`qfInput`	A list of data frame containing the input quality flag data that related to dp01 are being grouped. Of class integer". [-]
`TypeMeas`	A vector of class "character" containing the name of measurement type (sampling or validation), TypeMeas = c("samp", "vali"). Defaults to "samp". [-]
`PrdMeas`	The measurement time period in minute. [min]
`PrdAgr`	The time period to aggregate to averaging in minute. [min]
`idxTime`	A list of data frame containing the indices and corresponding times for aggregation periods. [-]

Value

A list of dp01 descriptive statistics.
`mean` The mean of non-NA values in `data` `min` The minimum value of non-NA values in `data` `max` The maximum value of non-NA values in `data` `vari` The variance of non-NA values in `data` `numSamp` The number of non-NA values in `data` `se` The standard error of the mean of non-NA values in `data` `timeBgn` The beginning time to determine descriptive statistics. `timeEnd` The ending time to determine descriptive statistics.

Author(s)

Natchaya Pingintha-Durden <`eddy4R.info@gmail.com`>

References

License: GNU AFFERO GENERAL PUBLIC LICENSE Version 3, 19 November 2007.

See Also

Currently none.

Examples

```
Currently none.
```

`wrap.qf.rmv.data` *Wrapper function: to remove high frequency data points that have failed quality flags*

Description

Wrapper function to remove high frequency data points that have failed quality flags from a data.frame

Usage

```
wrap.qf.rmv.data(inpList, Vrbs = FALSE, MethMeas = c("ecte", "ecse")[1])
```

Arguments

`inpList`	List consisting of `ff::ffdf` file-backed objects, in the format provided by function `eddy4R.base::wrap.neon.read.hdf5.eddy()`. Of types numeric and integer.
`Vrbs`	Optional. A logical FALSE/TRUE value indicating whether to: `Vrbs = FALSE`: (Default) cleaned data set with the bad high frequency quality flagged data replaced with NaN's as part of the `inpList` in the same format., or `Vrbs = TRUE`: cleaned data set with the bad high frequency quality flagged data replaced with NaN's as part of the `inpList` in the same format. In addition, a separate list `qfqmAnal` will be added to the output list `rpt` with a list of variables assessed, a list of quality flags for each variable assessed, the number of each quality flag tripped for each variable and the total number of bad data per variable.
`MethMeas`	A vector of class "character" containing the name of measurement method (eddy-covariance turbulent exchange or storage exchange), MethMeas = c("ecte", "ecse"). Defaults to "ecte". [-]

Value

The returned object consistes of `inpList`, with the bad high frequency quality flagged data replaced with NaN's. Optionally, (`Vrbs = TRUE`) a separate list `qfqmAnal` will be added to the output list `rpt` with a list of variables assessed, a list of quality flags for each variable assessed, the number of each quality flag tripped for each variable and the total number of bad data per variable.

Author(s)

Dave Durden <`ddurden@battelleecology.org`>

References

License: GNU AFFERO GENERAL PUBLIC LICENSE Version 3, 19 November 2007.
NEON Algorithm Theoretical Basis Document:Eddy Covariance Turbulent Exchange Subsystem Level 0 to Level 0' data product conversions and calculations (NEON.DOC.000807)
Licor LI7200 reference manual

See Also

Currently none

Examples

```
Currently none
```

Index