Package 'SeaSondeR'

May 7, 2026

Title Radial Metrics from SeaSonde HF-Radar Data

Version 0.2.8

Description

Read CODAR's SeaSonde High-Frequency Radar spectra files, compute radial metrics, and generate plots for spectra and antenna pattern data. Implementation is based in technical manuals, publications and patents, please refer to the following documents for more information: Barrick and Lipa (1999) <https://codar.com/images/about/patents/05990834.PDF>; CODAR Ocean Sensors (2002) <http://support.codar.com/Technicians_Information_Page_for_SeaSondes/Docs/Informative/FirstOrder_Settings.pdf>; Lipa et al. (2006) <doi:10.1109/joe.2006.886104>; Paolo et al. (2007) <doi:10.1109/oceans.2007.4449265>; CODAR Ocean Sensors (2009a) <http://support.codar.com/Technicians_Information_Page_for_SeaSondes/Docs/GuidesToFileFormats/File_AntennaPattern.pdf>; CODAR Ocean Sensors (2009b) <http://support.codar.com/Technicians_Information_Page_for_SeaSondes/Docs/GuidesToFileFormats/File_CrossSpectraReduced.pdf>; CODAR Ocean Sensors (2016a) <http://support.codar.com/Technicians_Information_Page_for_SeaSondes/Manuals_Documentation_Release_8/File_Formats/File_Cross_Spectra_V6.pdf>; CODAR Ocean Sensors (2016b) <http://support.codar.com/Technicians_Information_Page_for_SeaSondes/Manuals_Documentation_Release_8/File_Formats/FIle_Reduced_Spectra.pdf>; CODAR Ocean Sensors (2016c) <http://support.codar.com/Technicians_Information_Page_for_SeaSondes/Manuals_Documentation_Release_8/Application_Guides/Guide_SpectraPlotterMap.pdf>; Bushnell and Worthington (2022) <doi:10.25923/4c5x-g538>.

License GPL (>= 3)

Encoding UTF-8

RoxygenNote 7.3.2

Depends R (>= 4.1.0)

Suggests here (>= 1.0.1), mockthat (>= 0.2.8), testthat (>= 3.0.0), openssl (>= 2.1.0), jsonlite (>= 1.8.7), knitr, rmarkdown

Config/testthat/edition 3

URL https://github.com/GOFUVI/SeaSondeR, https://gofuvi.github.io/SeaSondeR/

BugReports https://github.com/GOFUVI/SeaSondeR/issues

Imports bit64 (>= 4.0.5), bitops (>= 1.0.7), constants (>= 1.0.1), data.table (>= 1.15.4), dplyr (>= 1.1.3), geosphere (>= 1.5.18), ggplot2 (>= 3.5.1), glue (>= 1.6.2), lubridate (>= 1.9.3), magrittr (>= 2.0.3), pracma (>= 2.4.4), purrr (>= 1.0.2), rlang (>= 1.1.1), slider (>= 0.3.1), stringr (>= 1.5.0), tibble (>= 3.2.1), tidyr (>= 1.3.1), uuid (>= 1.2.1), whisker (>= 0.4.1), yaml (>= 2.3.7), zoo (>= 1.8.12)

VignetteBuilder knitr

NeedsCompilation no

Author Juan Luis Herrera Cortijo [aut, cre] (ORCID: <https://orcid.org/0000-0002-4206-2459>), Ramiro A. Varela Benvenuto [aut] (ORCID: <https://orcid.org/0000-0002-9212-577X>), Adrián Fernández Baladrón [aut] (ORCID: <https://orcid.org/0000-0001-6795-4261>)

Maintainer Juan Luis Herrera Cortijo <juan.luis.herrera.cortijo@gmail.com>

Repository CRAN

Date/Publication 2025-04-30 11:20:06 UTC

Contents

`dB_to_self_spectra` *Convert dB Values to Self-Spectra Power*

Description

This function converts power values expressed in decibels (dB) to linear self-spectra power values. The conversion is based on the given receiver gain, which accounts for the radar system's amplification effects.

Usage

```
dB_to_self_spectra(dB_values, receiver_gain)
```

Arguments

`dB_values`	A numeric vector. The power values in decibels (dB).
`receiver_gain`	A numeric scalar. The receiver gain in decibels (dB).

Details

The conversion from decibels to linear power follows the equation:

$$P = 10^{(dB+G)/10}$$

where:

- \(P \) is the self-spectra power in linear scale,
- \(dB \) represents the power values in decibels,
- \(G \) is the receiver gain in decibels.

Value

A numeric vector of self-spectra power values in linear scale.

See Also

`self_spectra_to_dB` for the inverse operation.

`new_SeaSondeRCS` *Create a New SeaSondeRCS Object*

Description

This function constructs a new SeaSondeRCS object with the provided header and data information, initializing default values for various attributes including processing steps, FOR and MUSIC data, noise level, APM, and reference noise normalized limits estimation interval.

Usage

```
new_SeaSondeRCS(header, data, seasonder_apm_object = NULL)
```

Arguments

`header`	A list containing header information for the SeaSondeRCS object.
`data`	A list containing the data fields for the SeaSondeRCS object.
`seasonder_apm_object`	An optional object representing the APM (Antenna Pattern Matrix or similar metadata). If provided, it is assigned to the SeaSondeRCS object; otherwise, the APM attribute is set to NULL.

Details

The object is created with the following components:

- `header`: Initially set to an empty list, then populated by `seasonder_setSeaSondeRCS_header`.
- `data`: Initially set to an empty list, then populated by `seasonder_setSeaSondeRCS_data`.
- `version`: Set to `1`.
- `ProcessingSteps`: A character vector to log processing steps.
- `FOR_data` and `MUSIC_data`: Initialized as empty lists.
- `NoiseLevel`: Set using `seasonder_defaultCSNoiseLevel()`.
- `APM`: Set to `seasonder_apm_object` if provided.
- `interpolated_doppler_cells_index`: An integer vector initialized as empty.
- `reference_noise_normalized_limits_estimation_interval`: Set using `seasonder_defaultCSReference_nois`
- The object's class is set to `c("SeaSondeRCS", "list")`.

After constructing the base object, the function updates the header and data attributes, initializes FOR parameters, and sets up the FOR configuration by calling `seasonder_initSeaSondeRCS_FOR`. A processing step message is logged to indicate successful creation.

Value

A SeaSondeRCS object with version 1 containing the specified header, data, and default-initialized attributes.

See Also

seasonder_setSeaSondeRCS_header, seasonder_setSeaSondeRCS_data, seasonder_setFOR_parameters, seasonder_setSeaSondeRCS_FOR

`parse_metadata_line` *Parse a Metadata Line from a SeaSonde APM File*

Description

This function takes a single line from a SeaSonde APM file and parses it into a named attribute and its corresponding value.

Usage

```
parse_metadata_line(line)
```

Arguments

`line`	The line of text to parse.

Value

A list containing the attribute name and its value.

`print.SeaSondeRAPM` *Print a SeaSondeRAPM Object*

Description

This function prints the details of a SeaSondeRAPM object, including the station code, original file name, site origin (latitude and longitude), and antenna bearing. It is primarily used for displaying the object's metadata in a human-readable format.

Usage

```
## S3 method for class 'SeaSondeRAPM'
print(x, ...)
```

Arguments

`x`	A SeaSondeRAPM object. This object should be created using the seasonder_createSeaSondeRAPM() function and must include a calibration matrix, a quality matrix, the BEAR attribute, and a StationCode.
`...`	Additional arguments that might be passed to other methods; currently not used.

Value

The SeaSondeRAPM object itself, invisibly.

Examples

```
# Print metadata of a test SeaSondeRAPM object
apm_file <- system.file("css_data/MeasPattern.txt", package = "SeaSondeR")
obj <- seasonder_readSeaSondeRAPMFile(apm_file)
print(obj)
```

---

print.SeaSondeRCS *Print Method for SeaSondeRCS Object*

Description

This method provides a formatted printout of the SeaSondeRCS object, displaying the station code, date/time, number of Doppler cells, and number of range cells. It is designed for interactive use, allowing users to quickly inspect the object.

Usage

```
## S3 method for class 'SeaSondeRCS'
print(x, ...)
```

Arguments

x	An object of class "SeaSondeRCS". This object should contain at least a header list with metadata (such as station name, date/time, and cell counts).
...	Additional arguments. Currently not used, but supplied for compatibility with generic print methods.

Details

The function uses the `whisker` package to render a template string with the header information.

Value

Invisibly returns the original SeaSondeRCS object.

Examples

```
obj <- list(header = list(nSiteCodeName = "Station1",
                          nDateTime = Sys.time(),
                          nDopplerCells = 256,
                          nRangeCells = 100))
class(obj) <- "SeaSondeRCS"
print(obj)
```

process_version_header *Process a Specific Version of the SeaSonde File Header*

Description

This function processes a specified version of the SeaSonde file header. It identifies the appropriate header function for the given version, processes the header, and then updates the accumulating pool of header data. Specifically:

Usage

```
process_version_header(
  pool,
  version,
  specs,
  connection,
  endian = "big",
  prev_data = NULL
)
```

Arguments

pool	List. An accumulating list of processed headers from prior versions.
version	Integer. The specific version of the header to be processed. E.g., for version 3, the function seasonder_readSeaSondeCSFileHeaderV3 should be present.
specs	List. Header specifications for each version. Each entry should correspond to a version number and contain the required information to process that version's header.
connection	Connection object. The file connection pointing to the SeaSonde file.
endian	Character string. Specifies the byte order for reading data. Can be "big" (default)
prev_data	previous header data or "little". Use the appropriate value depending on the system architecture and the file's source.

Details

1. For fields in the current header that overlap with the accumulated pool, the current header's values overwrite those in the pool.
2. Fields that are unique to the current header are appended to the pool.

Value

List. A combination of the initial pool and the processed header for the given version. Fields in the current header will overwrite or append to the pool as described above.

Assumptions

This function assumes that the desired version-specific `seasonder_readSeaSondeCSFileHeaderV*` functions are available in the global environment.

See Also

`seasonder_readSeaSondeCSFileHeaderV2` `seasonder_readSeaSondeCSFileHeaderV3` `seasonder_readSeaSondeCSFi`
`seasonder_readSeaSondeCSFileHeaderV5` `seasonder_readSeaSondeCSFileHeaderV6`

---

qc_check_range *Quality Control - Check Range and Type*

Description

This function verifies if a given value lies within a specified range and matches the expected type, if provided.

Usage

```
qc_check_range(field_value, min, max, expected_type = NULL)
```

Arguments

`field_value`	The value to be checked.
`min`	Minimum allowable value for field_value.
`max`	Maximum allowable value for field_value.
`expected_type`	(optional) The expected type of the field_value. Default is NULL.

Value

The original field_value if it's within range and matches the expected_type; otherwise, an error is raised.

---

qc_check_type *Quality Control - Check Type*

Description

This function verifies if a given value is of the expected type.

Usage

```
qc_check_type(field_value, expected_type)
```

Arguments

field_value	The value whose type needs to be checked.
expected_type	The expected type of the field_value.

Value

The original field_value if it matches the expected_type; otherwise, an error is raised.

---

qc_check_unsigned *Quality Control Check for Unsigned Values*

Description

This function performs a quality control check to ensure that a given field value is an unsigned number (i.e., a non-negative number). Optionally, it can also check if the field value matches a specified data type before performing the unsigned check.

Usage

```
qc_check_unsigned(field_value, expected_type = NULL)
```

Arguments

field_value	The value to be checked. The function verifies if this value is non-negative. It can be of any type but is typically expected to be a numeric value.
expected_type	An optional parameter specifying the expected data type of field_value. If provided, the function first checks if field_value matches the expected type before verifying if it is unsigned. Default is NULL, which means no type check is performed.

Value

Returns the field_value if it passes the checks: it is of the expected type (if expected_type is not NULL) and is non-negative. If any of the checks fail, the function logs an error message and aborts execution.

`readV6BlockData` *Read Version 6 Block Data*

Description

This function reads and processes regular and repeated blocks of data based on provided specifications. Regular blocks are read directly, while repeated blocks are processed recursively based on a set of loops provided in the specifications.

Usage

```
readV6BlockData(
  specs,
  connection,
  endian = "big",
  prev_data = NULL,
  remaining_loops = NULL
)
```

Arguments

`specs`	A list. Specifications detailing the structure and content of the data blocks. Contains variable names, types, quality check functions, and other related attributes. For repeated blocks, a 'repeat' key is added which details the loop structure and nested specifications.
`connection`	A connection object. Represents the connection to the data source. It's passed to the lower-level reading function.
`endian`	A character string. Specifies the byte order to be used. Default is "big". Passed to the lower-level reading function.
`prev_data`	A list. Previous data or metadata that might be required to inform the reading process, such as loop lengths for repeated blocks. Default is NULL.
`remaining_loops`	A character vector. Details the remaining loops to be processed for repeated blocks. Internally used for recursive processing. Default is NULL. If provided, it should always be in sync with the repeat specifications.

Value

A list. Contains the read and processed data based on the provided specifications. Regular variables are returned at the top level. Repeated blocks are nested lists with 'loop' and 'data' keys detailing the loop variable and corresponding data.

See Also

`readV6BlockData`

Examples

```
# Example: read a single UInt8 value using internal helper
specs <- list(
  field1 = list(
    type = "UInt8",
    qc_fun = "qc_check_unsigned",
    qc_params = list()
  )
)
con <- rawConnection(as.raw(c(10)), "rb")
result <- readV6BlockData(specs, con, endian = "big")
print(result)
close(con)
```

`read_and_qc_field` *Read and Quality Control a Single Field*

Description

This auxiliary function reads a field from a binary file using a provided specification and applies a quality control function on the retrieved data. The expectations and functioning of the quality control functions are described in detail in the documentation for `seasonder_readSeaSondeCSFileBlock`.

Usage

```
read_and_qc_field(field_spec, connection, endian = "big")
```

Arguments

`field_spec`	A list containing the specifications for the field to read. It should contain: • `type`: the type of data to read, passed to `seasonder_readCSField`. • `qc_fun`: the name of a quality control function. As detailed in `seasonder_readSeaSondeCSFileBlo` this function should be present in the shared environment `seasonder_the` and must accept `field_value` as its first argument, followed by any other arguments specified in `qc_params`. • `qc_params`: a list of additional parameters to pass to the quality control function. See `seasonder_readSeaSondeCSFileBlock` for detailed expectations of the QC function behavior.
`connection`	A connection to the binary file.
`endian`	A character string indicating the byte order. Options are "big" and "little" (default is "big").

Value

The value of the field after applying quality control.

Condition Management

This function utilizes the `rlang` package to manage conditions and provide detailed and structured condition messages:

Condition Classes:

- `seasonder_cs_field_skipped`: Condition that indicates a CSField was skipped during reading.
- `seasonder_cs_field_qc_fun_rerun`: Condition that indicates a rerun of the quality control function was triggered.
- `seasonder_cs_field_qc_fun_not_defined_error`: Error raised when the quality control function specified is not found in the shared environment `seasonder_the`.
- `seasonder_cs_field_qc_fun_error`: Error raised when an issue occurs while applying the quality control function.

Condition Cases:

- If a CSField is skipped during reading, the condition `seasonder_cs_field_skipped` is used to skip QC and then is re-signaled.
- If an alternate QC is rerun using the `seasonder_rerun_qc_with_fun` restart, the condition `seasonder_cs_field_qc_fun_rerun` is signaled.
- If the quality control function specified is not found in the shared environment `seasonder_the`, the error `seasonder_cs_field_qc_fun_not_defined_error` is raised.
- If there's an issue applying the quality control function, the error `seasonder_cs_field_qc_fun_error` is raised.

Restart Options: The function provides structured mechanisms to recover from errors/conditions during its execution using `withRestarts`. The following restart options are available:

- `seasonder_rerun_qc_with_fun`: Allows for rerunning QC with an alternate function.
  - **Usage**: In a custom condition handler, you can call `seasonder_rerun_qc_with_fun(cond, alternateQCfunction)` to trigger this restart and run an alternate QC using `alternateQCfunction`. `alternateQCfunction` will be used as follows alternateQCfunction(x) being x the value. No extra parameters are passed.
  - **Effect**: If invoked, the function logs an info message detailing the reason of the rerun, and then returns the value returned by `alternateQCfunction`.

See Also

`seasonder_rerun_qc_with_fun`, `seasonder_readCSField`

It's also important to note that within `read_and_qc_field`, the function `seasonder_readCSField` is used. This function has its own error management and restart options, which are detailed in its documentation.

read_matrix_row *Read a Row from a Matrix Represented as Text Lines*

Description

This function reads a row of numbers from a matrix that is represented as an array of text lines. It is used to facilitate reading data from SeaSonde APM files.

Usage

```
read_matrix_row(lines, start, number_of_lines_to_read)
```

Arguments

lines	The array of lines, each representing part of the row.
start	The start index of the lines to read from.
number_of_lines_to_read	The number of lines to read to form the row.

Value

A numeric vector containing the row values.

SeaSondeRAPM_amplitude_and_phase_corrections_step_text *Generate Amplitude and Phase Corrections Step Text*

Description

This function generates a message indicating the amplitude and phase corrections applied to the APM.

Usage

```
SeaSondeRAPM_amplitude_and_phase_corrections_step_text(
  amplitude1,
  amplitude2,
  phase1,
  phase2
)
```

Arguments

amplitude1	Amplitude correction for the first channel.
amplitude2	Amplitude correction for the second channel.
phase1	Phase correction (in degrees) for the first channel.
phase2	Phase correction (in degrees) for the second channel.

Value

A character string detailing the applied amplitude and phase corrections.

SeaSondeRAPM_amplitude_factors_override_step_text
Generate Amplitude Factors Override Step Text

Description

This function generates a message indicating that amplitude factors have been overridden.

Usage

```
SeaSondeRAPM_amplitude_factors_override_step_text(amplitude_factors)
```

Arguments

`amplitude_factors`
: A numeric vector with two elements for the new amplitude factors.

Value

A character string stating the new amplitude factors.

SeaSondeRAPM_antenna_bearing_override_step_text
Generate Antenna Bearing Override Step Text

Description

This function generates a message indicating that the AntennaBearing attribute was overridden.

Usage

```
SeaSondeRAPM_antenna_bearing_override_step_text(antenna_bearing)
```

Arguments

`antenna_bearing`
: The new antenna bearing value.

Value

A character string stating that the antenna bearing has been overridden.

SeaSondeRAPM_creation_step_text
Generate Creation Step Text

Description

This function generates a text message indicating the time an APM object was created based on the current system time and the provided file path.

Usage

```
SeaSondeRAPM_creation_step_text(file_path)
```

Arguments

file_path	A character string specifying the path to the file.

Value

A character string with the formatted creation message.

SeaSondeRAPM_phase_correction_override_step_text
Generate Phase Correction Override Step Text

Description

This function generates a message indicating that phase corrections have been overridden.

Usage

```
SeaSondeRAPM_phase_correction_override_step_text(phase_correction)
```

Arguments

phase_correction	A numeric vector with two elements for the new phase corrections.

Value

A character string stating the new phase correction values.

SeaSondeRAPM_SiteOrigin_override_step_text
Generate SiteOrigin Override Step Text

Description

This function generates a message indicating that the SiteOrigin has been overridden.

Usage

```
SeaSondeRAPM_SiteOrigin_override_step_text(SiteOrigin)
```

Arguments

SiteOrigin	A numeric vector with two elements representing the new latitude and longitude.

Value

A character string with the updated SiteOrigin details.

SeaSondeRAPM_smoothing_step_text
Generate Smoothing Step Text

Description

This function generates a message indicating that smoothing has been applied to the APM.

Usage

```
SeaSondeRAPM_smoothing_step_text(smoothing)
```

Arguments

smoothing	The smoothing parameter (number of points used).

Value

A character string detailing the smoothing operation.

SeaSondeRAPM_trimming_step_text

Generate Trimming Step Text

Description

This function generates a message indicating that trimming has been applied to the APM.

Usage

```
SeaSondeRAPM_trimming_step_text(trimming)
```

Arguments

trimming	The number of points trimmed from each end of the APM.

Value

A character string with the trimming details.

SeaSondeRCS_creation_step_text

Generate Creation Step Text

Description

This function generates a text message indicating the time an CS object was created based on the current system time and the provided file path.

Usage

```
SeaSondeRCS_creation_step_text(file_path)
```

Arguments

file_path	A character string specifying the path to the file.

Value

A character string with the formatted message indicating the time of creation and the file path.

`SeaSondeRCS_MUSIC_validate_doppler_interpolation` *Validate Doppler Interpolation Factor for SeaSondeRCS Objects*

Description

This function validates the `doppler_interpolation` factor for a `SeaSondeRCS` object, ensuring it is within the allowed range and does not result in exceeding the maximum number of Doppler bins after interpolation.

Usage

```
SeaSondeRCS_MUSIC_validate_doppler_interpolation(value, seasonder_cs_object)
```

Arguments

`value`	An integer specifying the Doppler interpolation factor. Must be one of 1, 2, 3, or 4.
`seasonder_cs_object`	A `SeaSondeRCS` object containing metadata for Doppler bin calculations.

Details

Doppler interpolation is a process that increases the number of Doppler bins by the specified factor before radial processing. The function performs the following validations:

- Ensures the `doppler_interpolation` factor is one of 1, 2, 3, or 4.
- Computes the total number of Doppler bins after applying the specified interpolation factor. If this number exceeds 2048, the function aborts with a descriptive error message.

The maximum Doppler bins (2048) constraint is derived from CODAR's SeaSonde R8 Radial Config Setup, which specifies that the product of the interpolation factor and the original number of Doppler bins should not exceed this limit.

Value

The validated `doppler_interpolation` factor as an integer.

Warnings

- Using Doppler interpolation factors of 3x or 4x is not recommended.
- Exceeding 2048 Doppler bins after interpolation will result in an error.

See Also

`seasonder_getnDopplerCells` for retrieving the number of Doppler bins, `seasonder_logAndAbort` for error handling and logging.

seasonder_applyAPMAmplitudeAndPhaseCorrections

Apply Amplitude and Phase Corrections to a SeaSonde RAPM Object

Description

This function applies amplitude and phase corrections to each antenna channel of a SeaSonde RAPM object based on the correction factors stored within the object.

Usage

```
seasonder_applyAPMAmplitudeAndPhaseCorrections(seasonder_apm_object)
```

Arguments

seasonder_apm_object	A SeaSonde RAPM object containing raw data and correction factors.

Value

The SeaSonde RAPM object with amplitude and phase corrections applied to the data.

Examples

```
# Apply amplitude & phase corrections to a test SeaSondeRAPM object
apm_file <- system.file("css_data/MeasPattern.txt", package = "SeaSondeR")
obj <- seasonder_readSeaSondeRAPMFile(apm_file)
corrected_obj <- seasonder_applyAPMAmplitudeAndPhaseCorrections(obj)
```

seasonder_applyCSSWSigns

Apply CSSW Sign Corrections

Description

Applies sign corrections to both cross-spectra and auto-spectra fields within a list of CSSW data cells.

Usage

```
seasonder_applyCSSWSigns(cs_data)
```

Arguments

cs_data	A list of CSSW data cells, where each cell may include fields for cross-spectra ('c12m', 'c12a', 'c13m', 'c13a', 'c23m', 'c23a') and auto-spectra ('cs1a', 'cs2a', 'cs3a') signs.

Value

The modified list of CSSW data cells with sign corrections applied.

`seasonder_areLogsEnabled` *Check if log recording is enabled in SeaSondeR*

Description

This function checks whether log recording is currently enabled in the SeaSondeR package.

Usage

```
seasonder_areLogsEnabled()
```

Value

Logical indicating whether logs are enabled or disabled.

Examples

```
seasonder_areLogsEnabled()
```

`seasonder_areMessagesEnabled` *Check if message logging is enabled in SeaSondeR*

Description

This function checks whether message logging is currently enabled.

Usage

```
seasonder_areMessagesEnabled()
```

Value

Logical value indicating whether messages are enabled.

Examples

```
seasonder_areMessagesEnabled()
```

`seasonder_asJSONSeaSondeRCSData` *Convert SeaSondeRCS Object to JSON*

Description

This function extracts the data from a `seasonder_cs_object`, representing a SeaSondeRCS object, and converts it into a JSON format. Optionally, it can write this JSON data to a specified file path.

Usage

```
seasonder_asJSONSeaSondeRCSData(seasonder_cs_object, path = NULL)
```

Arguments

`seasonder_cs_object`	A SeaSondeRCS object from which the data will be extracted.
`path`	Optional path to a file where the JSON output should be saved. If provided, the function will write the JSON data to this file. If NULL, the function will only return the JSON data as a string without writing it to a file.

Value

A character string in JSON format representing the data of the provided SeaSondeRCS object. If a path is provided, the function also writes this data to the specified file.

Note

If a path is provided and there is an issue writing to the file, the function logs an error message using `seasonder_logAndMessage` and returns the JSON data as a string.

See Also

`seasonder_createSeaSondeRCS`, `seasonder_getSeaSondeRCS_data`

Examples

```
# Example: create a simple SeaSondeRCS object and convert its data to JSON
cs_obj <- structure(list(data = list(a = 1, b = 2)), class = "SeaSondeRCS")
json_output <- seasonder_asJSONSeaSondeRCSData(cs_obj)
print(json_output)
```

`seasonder_asJSONSeaSondeRCSHeader` *Convert SeaSondeRCS Object to JSON*

Description

This function extracts the header data from a `seasonder_cs_object`, representing a SeaSondeRCS object, and converts it into a JSON format. Optionally, it can write this JSON data to a specified file path.

Usage

```
seasonder_asJSONSeaSondeRCSHeader(seasonder_cs_object, path = NULL)
```

Arguments

`seasonder_cs_object`	A SeaSondeRCS object from which the header data will be extracted.
`path`	Optional path to a file where the JSON output should be saved. If provided, the function will write the JSON data to this file. If NULL, the function will only return the JSON data as a string without writing it to a file.

Value

A character string in JSON format representing the header data of the provided SeaSondeRCS object. If a path is provided, the function also writes this data to the specified file.

Note

If a path is provided and there is an issue writing to the file, the function logs an error message using `seasonder_logAndMessage` and returns the JSON data as a string.

See Also

`seasonder_createSeaSondeRCS`, `seasonder_getSeaSondeRCS_header`

Examples

```
# Example: create a simple SeaSondeRCS object and convert its header to JSON
cs_obj <- structure(list(data = list(a = 1, b = 2)), class = "SeaSondeRCS")
attr(cs_obj, "header") <- list(
  nSiteCodeName = "Station1",
  nDateTime = Sys.time(),
  nDopplerCells = 2,
  nRangeCells = 3
)
json_header <- seasonder_asJSONSeaSondeRCSHeader(cs_obj)
print(json_header)
```

seasonder_Bins2DopplerFreq

Convert Doppler Bins to Doppler Frequencies

Description

This function retrieves the Doppler frequency values corresponding to the specified bin indices in a given SeaSondeR object.

Usage

```
seasonder_Bins2DopplerFreq(seasonder_cs_object, bins)
```

Arguments

seasonder_cs_object	A SeaSondeR cross-spectral object containing Doppler bin metadata.
bins	A numeric vector specifying the Doppler bin indices.

Details

This function retrieves the full set of Doppler bin frequencies using seasonder_getDopplerBinsFrequency in non-normalized form. It then selects the Doppler frequencies corresponding to the specified bin indices.

Value

A numeric vector of Doppler frequencies (in Hz) corresponding to the specified bins.

See Also

seasonder_DopplerFreq2Bins for the reverse operation. seasonder_getDopplerBinsFrequency for retrieving the full set of Doppler frequencies.

seasonder_Bins2NormalizedDopplerFreq

Convert Doppler Bins to Normalized Doppler Frequency

Description

This function retrieves the normalized Doppler frequencies corresponding to the specified bins in a given SeaSondeR object.

Usage

```
seasonder_Bins2NormalizedDopplerFreq(seasonder_cs_object, bins)
```

Arguments

seasonder_cs_object	A SeaSondeR cross-spectral object containing Doppler bin metadata.
bins	A numeric vector specifying the Doppler bin indices.

Details

This function first retrieves the Doppler bin frequencies in normalized form using seasonder_getDopplerBinsFrequency. It then selects the normalized Doppler frequencies corresponding to the specified bin indices.

Normalized Doppler Frequency Calculation: The normalized Doppler frequency is typically defined as:

$$f_{norm} = \frac{f_{doppler}}{f_{bragg}}$$

where:

- f_{norm} is the normalized Doppler frequency,
- $f_{doppler}$ is the Doppler frequency of a given bin,
- f_{bragg} is the Bragg frequency, computed based on radar wavelength.

Value

A numeric vector of normalized Doppler frequencies corresponding to the specified bins.

See Also

seasonder_getDopplerBinsFrequency for retrieving Doppler bin frequencies.

---

seasonder_check_specs *Validate Field Specifications*

Description

This function checks if the provided specifications (specs) contain entries for all the required fields listed in fields.

Usage

```
seasonder_check_specs(specs, fields)
```

Arguments

specs	A list containing field specifications.
fields	A character vector of field names to be checked in the specs.

Details

The function iterates over each field in the `fields` vector and checks if there is an associated entry in the `specs` list. If any field is missing, an error is thrown using `seasonder_logAndAbort` indicating the missing field specification.

Value

Invisibly returns NULL.

Condition Management

This function utilizes the `rlang` package to manage conditions, and provide detailed and structured condition messages:

Condition Classes:

- `spsr_field_specification_missing_error`: This error is thrown when a required field specification is missing from the `specs` list.

Condition Cases:

- Required field specification is missing.

`seasonder_computeBinsRadialVelocity`

Compute Radial Velocities for Doppler Bins

Description

This function calculates the radial velocities corresponding to the Doppler bins in a SeaSondeRCS object, based on the provided Doppler frequencies. The calculation uses the radar's wave number and Bragg angular frequencies.

Usage

```
seasonder_computeBinsRadialVelocity(seasonder_cs_object, freq)
```

Arguments

`seasonder_cs_object`	A `SeaSondeRCS` object containing data and metadata necessary for the calculation of Doppler bin frequencies and velocities.
`freq`	A numeric vector representing the Doppler frequencies for which the radial velocities are to be calculated.

Details

The radial velocity v for each Doppler bin is computed using the formula:

$$v = \frac{\text{Freq} - \text{BraggFreq}}{2 \cdot k_0}$$

where:

- Freq is the Doppler frequency of the bin.
- BraggFreq is the Bragg Doppler angular frequency for the bin.
- k_0 is the radar wave number divided by 2π.

The Bragg frequency is negative for bins with frequencies below zero and positive for bins with frequencies above zero.

Value

A numeric vector containing the radial velocities (in meters per second, m/s) corresponding to the provided Doppler frequencies.

See Also

`seasonder_getBraggDopplerAngularFrequency` to retrieve the Bragg angular frequencies. `seasonder_getRadarWaveNum` to obtain the radar wave number.

`seasonder_computeCenterDopplerBin`
Compute the Center Doppler Bin

Description

This function calculates the center Doppler bin for a SeaSondeRCS object based on the total number of Doppler bins. The center bin corresponds to the bin representing zero Doppler frequency.

Usage

```
seasonder_computeCenterDopplerBin(seasonder_cs_object, nDoppler)
```

Arguments

`seasonder_cs_object`	A `SeaSondeRCS` object containing metadata about Doppler bins and other radar parameters.
`nDoppler`	An integer representing the total number of Doppler bins.

Details

The center Doppler bin is computed as: $center\_bin = nDoppler/2$ where nDoppler is the total number of Doppler bins. This represents the bin at zero Doppler frequency in a zero-indexed system. Since R uses one-based indexing, users might observe an offset when comparing the output of this function to CODAR's Radia Suite programs.

Value

A numeric value representing the center Doppler bin. The calculation assumes zero-based indexing from CODAR data files, but note that R uses one-based indexing, which may result in differences compared to CODAR's Radia Suite outputs.

See Also

`seasonder_getSeaSondeRCS_MUSIC_nDopplerCells` to retrieve the number of Doppler cells from a SeaSondeRCS object.

`seasonder_computeDopplerBinsFrequency` *Compute Doppler Bins Frequencies*

Description

This function computes the Doppler frequencies associated with each Doppler bin in a SeaSondeRCS object. The output can be normalized by the positive Bragg frequency if specified.

Usage

```
seasonder_computeDopplerBinsFrequency(
  seasonder_cs_object,
  nDoppler,
  center_bin,
  spectra_res,
  normalized = FALSE
)
```

Arguments

`seasonder_cs_object`	A SeaSonde CS object created by `seasonder_createSeaSondeRCS()`. This object contains the necessary metadata, such as Doppler resolution and center bin, for frequency computation.
`nDoppler`	Integer. The total number of Doppler bins.
`center_bin`	Numeric. The index of the central Doppler bin corresponding to 0 Hz.
`spectra_res`	Numeric. The spectral resolution in Hz for each Doppler bin.
`normalized`	Logical. If `TRUE`, the frequencies are normalized by dividing them by the positive Bragg frequency. Default is `FALSE`.

Details

Doppler frequencies are calculated using the formula:

$$\text{frequency}_i = (\text{bin index}_i - \text{center bin}) \times \text{resolution}$$

For normalized frequencies:

$$\text{frequency}_i = \frac{\text{frequency}_i}{\text{positive Bragg frequency}}$$

The center bin is typically determined using `seasonder_getCenterDopplerBin()`, and the resolution is obtained from `seasonder_getDopplerSpectrumResolution()`. Normalization is based on the positive Bragg frequency calculated by `seasonder_getBraggDopplerAngularFrequency()`.

Value

A numeric vector representing the Doppler frequencies for each bin. If `normalized = TRUE`, the values are dimensionless and relative to the positive Bragg frequency. Otherwise, they are in Hz.

See Also

`seasonder_getCenterDopplerBin`, `seasonder_getDopplerSpectrumResolution`, `seasonder_getBraggDopplerAngula`

`seasonder_computeDopplerFreq2Bins`
Convert Doppler Frequencies to Doppler Bins

Description

This function converts a set of Doppler frequency values into their corresponding Doppler bin indices using predefined Doppler frequency bins and frequency step size.

Usage

```
seasonder_computeDopplerFreq2Bins(
  seasonder_cs_object,
  doppler_values,
  doppler_freqs,
  delta_freq,
  nDoppler
)
```

Arguments

seasonder_cs_object	A SeaSondeR cross-spectral object.
doppler_values	A numeric vector specifying the Doppler frequencies to be converted into bin indices.
doppler_freqs	A numeric vector containing the Doppler frequencies corresponding to each bin.
delta_freq	A numeric scalar specifying the frequency step size (difference between consecutive Doppler bins).
nDoppler	An integer indicating the total number of Doppler bins.

Details

The function constructs a set of bin boundaries using the Doppler frequencies. The leftmost boundary is adjusted by subtracting `delta_freq` from the first Doppler frequency to extend the range.

The function then applies `findInterval` to determine the corresponding bin index for each input Doppler frequency. The bin assignment process follows these rules:

- `rightmost.closed = TRUE`: The last bin interval includes its upper boundary.
- `all.inside = FALSE`: Values outside the defined frequency range are assigned indices below 1 or above `nDoppler`.
- `left.open = TRUE`: The left interval is open, meaning values exactly equal to a boundary are assigned to the higher bin.

After determining the bin indices, values that are out of range (`bins < 1` or `bins > nDoppler`) are set to `NA`.

Value

An integer vector of Doppler bin indices corresponding to the input Doppler frequencies. Values that fall outside the valid bin range are assigned `NA`.

See Also

`seasonder_Bins2NormalizedDopplerFreq` for converting bins back to normalized Doppler frequencies. `findInterval` for details on interval-based bin selection.

---

seasonder_computeFORs *Compute First Order Regions (FOR) Based on Selected Method*

Description

This function processes a `SeaSondeRCS` object to compute the First Order Regions (FOR) using the specified method. It allows the user to configure the processing method and parameters dynamically.

Usage

```
seasonder_computeFORs(seasonder_cs_object, method = NULL, FOR_control = NULL)
```

Arguments

seasonder_cs_object	A SeaSondeRCS object containing the spectral data and FOR parameters.
method	Optional; A character string specifying the method to be used for FOR computation. Defaults to NULL, in which case the method stored in the object is used. Currently supported method: "SeaSonde".
FOR_control	Optional; A list of parameters for configuring the FOR computation process. Defaults to NULL, in which case the parameters already stored in the object are used.

Details

Steps:

1. **Set Method:**
   - If a method is provided, it updates the SeaSondeRCS object with the specified method.
2. **Retrieve Method:**
   - If no method is specified, the function retrieves the method stored in the object.
3. **Set Parameters:**
   - If FOR_control is provided, the function updates the object's FOR parameters.
4. **Method Execution:**
   - Based on the selected method, the corresponding processing function is called. Currently, the only supported method is "SeaSonde", which calls seasonder_computeFORsSeaSondeMethod.

Use Case: This function provides a flexible interface for computing FORs, allowing users to dynamically select methods and configure parameters without modifying the internal object structure.

Value

The updated SeaSondeRCS object with the computed First Order Regions (FOR).

See Also

- seasonder_computeFORsSeaSondeMethod for processing FORs using the SeaSonde method.
- seasonder_setSeaSondeRCS_FOR_method for setting the processing method.
- seasonder_setFOR_parameters for configuring FOR parameters.

Examples

```
# Set sample file paths
seasonder_disableMessages()
cs_file <- system.file("css_data/CSS_TORA_24_04_04_0700.cs", package = "SeaSondeR")
apm_file <- system.file("css_data/MeasPattern.txt", package = "SeaSondeR")
# Read the antenna pattern file to create a SeaSondeRAPM object
apm_obj <- seasonder_readSeaSondeRAPMFile(apm_file)
# Create a SeaSondeRCS object from a spectral file
cs_obj <- seasonder_createSeaSondeRCS(cs_file, seasonder_apm_object = apm_obj)
cs_obj <- seasonder_computeFORs(cs_obj, method = "SeaSonde")
# Retrieve existing FOR control parameters from the object
FOR_control <- seasonder_getSeaSondeRCS_FORConfig(cs_obj)
cs_obj <- seasonder_computeFORs(cs_obj, FOR_control = FOR_control)
```

---

seasonder_computeFORsSeaSondeMethod

Compute First Order Regions (FOR) Using the SeaSonde Method

Description

This function processes a SeaSondeRCS object to compute the First Order Regions (FOR) using the SeaSonde method. The workflow includes detecting null points, filtering amplitudes, limiting currents to a maximum range, and rejecting peaks based on proximity to Bragg indices and noise/ionospheric contamination.

Usage

```
seasonder_computeFORsSeaSondeMethod(seasonder_cs_object)
```

Arguments

`seasonder_cs_object`
: A `SeaSondeRCS` object containing the spectral data and configuration parameters.

Details

Workflow Steps:

1. **Initialize Processing Steps:**
   - Marks the start of the SeaSonde method processing in the object's metadata.
2. **Detect Null Points:**
   - Locates the nulls defining the First Order Regions (FOR) using `seasonder_findFORNulls`.
3. **Filter Amplitudes:**
   - Removes regions with insufficient amplitudes using `seasonder_filterFORAmplitudes`.
4. **Limit Currents to Maximum Range:**

- Ensures that currents exceed the configured maximum radial velocity using `seasonder_limitFORCurrentRange`.

5. **Reject Distant Bragg Peaks:**
   - If enabled, rejects peaks that are too far from Bragg indices using `seasonder_rejectDistantBragg`.
6. **Reject Noise/Ionospheric Peaks:**
   - If enabled, removes peaks where non-Bragg power significantly exceeds Bragg power using `seasonder_rejectNoiseIonospheric`.
7. **Finalize Processing Steps:**
   - Marks the end of the SeaSonde method processing in the object's metadata.

Use Case: This function is designed for processing SeaSonde radar data to accurately identify and validate the First Order Regions, ensuring reliable current and wave measurements.

Value

The updated `SeaSondeRCS` object with the computed First Order Regions (FOR).

See Also

- `seasonder_findFORNulls` for detecting nulls in the FOR.
- `seasonder_filterFORAmplitudes` for amplitude filtering.
- `seasonder_limitFORCurrentRange` for limiting radial velocity.
- `seasonder_rejectDistantBragg` for rejecting distant Bragg peaks.
- `seasonder_rejectNoiseIonospheric` for rejecting noise/ionospheric contamination.

---

seasonder_computeLonLatFromOriginDistBearing

Compute Geographic Coordinates from Origin, Distance, and Bearing

---

Description

This function calculates the geographic coordinates (latitude and longitude) for a given distance and bearing from a specified origin.

Usage

```
seasonder_computeLonLatFromOriginDistBearing(
  origin_lon,
  origin_lat,
  dist,
  bearing
)
```

Arguments

origin_lon	A numeric value representing the longitude of the origin point in decimal degrees.
origin_lat	A numeric value representing the latitude of the origin point in decimal degrees.
dist	A numeric value representing the distance from the origin in kilometers.
bearing	A numeric vector of bearings (in degrees) indicating the direction from the origin.

Details

The function uses the geodetic formulas provided by the geosphere package to compute the destination point based on:

- Origin longitude and latitude
- Distance in meters (converted from kilometers)
- Bearing in degrees

The calculation employs the geosphere::destPoint function, which handles the spherical geometry of the Earth.

Value

A data frame with two columns:

- lon: The longitude of the computed geographic coordinates.
- lat: The latitude of the computed geographic coordinates.

See Also

destPoint

Examples

```
# Example with a point at 100 km to the north of the origin
result <- seasonder_computeLonLatFromOriginDistBearing(-123.3656, 48.4284, 100, 0)
print(result)
```

---

seasonder_computeNoiseLevel

Compute Noise Level for First Order Region (FOR) Processing

Description

This function estimates the noise level in the self-spectra of a SeaSondeR cross-spectral object. The noise level is determined by averaging the spectral power over a predefined frequency range where no first-order Bragg signal is expected. This value is later used in setting signal-to-noise thresholds for FOR detection.

Usage

```
seasonder_computeNoiseLevel(seasonder_cs_object, antenna = 3, smoothed = FALSE)
```

Arguments

seasonder_cs_object	A SeaSondeRCS object containing spectral data and FOR parameters.
antenna	A numeric value specifying the antenna from which to extract self-spectra (default is 3).
smoothed	Logical; if TRUE, the function uses a smoothed version of the self-spectra.

Details

The noise level is computed via the following steps:

1. **Determine Noise Reference Limits:**
   - Retrieves the normalized Doppler frequency limits for noise reference from the FOR parameters (using seasonder_getFOR_parameters).
   - Converts these normalized limits into Doppler bin indices using seasonder_SwapDopplerUnits.
   - If any of the resulting bin indices are missing, they are replaced with appropriate default boundaries (i.e., upper limit set to the total number of Doppler cells and lower limit set to 1).
2. **Extract Spectral Data for Noise Estimation:**
   - The function extracts the self-spectra from the specified antenna (using seasonder_getSeaSondeRCS_SelfSpectr limiting the extraction to the Doppler bins within the computed noise reference range (both negative and positive regions).
3. **Compute the Average Noise Level:**
   - The spectral data from both the negative and positive Doppler regions are concatenated, and the row-wise mean is calculated to estimate the average noise level.
4. **Store the Noise Level:**
   - The computed average noise level is stored in the object's NoiseLevel attribute by calling seasonder_setSeaSondeRCS_NoiseLevel.
   - A processing step message is logged using SeaSondeRCS_computeNoiseLevel_step_text.

The resulting noise level is essential for setting accurate thresholds during FOR detection.

Value

The updated SeaSondeRCS object with the computed noise level stored in its attributes.

See Also

- seasonder_getFOR_parameters for retrieving noise reference limits.
- seasonder_SwapDopplerUnits for converting normalized Doppler frequencies into bin indices.
- seasonder_getSeaSondeRCS_SelfSpectra for extracting self-spectra.
- seasonder_setSeaSondeRCS_NoiseLevel for storing the computed noise level.

`seasonder_computePowerMatrix` *Compute Power Matrix*

Description

This function calculates the power matrix based on the provided steering vector, eigenvalues, and eigenvectors. The computation differs depending on the number of columns in the steering vector matrix.

Usage

```
seasonder_computePowerMatrix(eig, a)
```

Arguments

`eig`	A list containing the eigenvalues and eigenvectors of a covariance matrix. The list should include: • `values`: A numeric vector of eigenvalues. • `vectors`: A matrix where each column is an eigenvector.
`a`	A complex matrix representing the steering vector(s). Each column corresponds to a direction of arrival.

Details

The function computes the power matrix using the following steps:

- If `a` has two columns:
  1. Select the first two eigenvalues and their corresponding eigenvectors.
  2. Compute the matrix $G = a^* \cdot \text{eigVector}$, where a^* is the conjugate transpose of `a`.
  3. Calculate the inverse of `G` and its conjugate transpose.
  4. Compute the power matrix $P = G^*_{\text{inv}} \cdot \text{diag(eigValues)} \cdot G_{\text{inv}}$.
- If `a` has one column:
  1. Select the first eigenvalue and its corresponding eigenvector.
  2. Follow similar steps as above with single-column operations.

If `a` has no columns, the function returns `NULL`.

Value

A complex matrix representing the power matrix, calculated based on the provided eigenvalues, eigenvectors, and steering vectors. If the number of columns in `a` is zero, the function returns `NULL`.

Mathematical Formula

For a steering vector matrix a, eigenvectors eigVector, and eigenvalues eigValues, the power matrix is calculated as:

$$P = G_{\text{inv}}^{*} \cdot \text{diag}(\text{eigValues}) \cdot G_{\text{inv}}$$

where: $G = a^{*} \cdot \text{eigVector}$ and G_{inv} is the inverse of G.

References

- Paolo, T. de, Cook, T., & Terrill, E. (2007). Properties of HF RADAR Compact Antenna Arrays and Their Effect on the MUSIC Algorithm. *OCEANS 2007*, 1–10. doi:10.1109/oceans.2007.4449265.

---

seasonder_compute_antenna_pattern_proyections

Compute Antenna Pattern Projections for the MUSIC Algorithm

Description

This function computes the projection of the antenna pattern vector onto the noise subspace, a critical step in the Multiple Signal Classification (MUSIC) algorithm. It is used to estimate the direction of arrival (DOA) by identifying the bearing that minimizes this projection.

Usage

```
seasonder_compute_antenna_pattern_proyections(En, a, Conj_t_a)
```

Arguments

`En`	A matrix containing the eigenvectors of the noise subspace, derived from the covariance matrix of the signals.
`a`	A complex-valued vector representing the antenna manifold response for a specific bearing. Each element corresponds to the response of an antenna element.
`Conj_t_a`	The conjugate transpose of the antenna manifold vector `a`.

Details

The MUSIC algorithm leverages the property that the antenna manifold vector is orthogonal to the noise subspace eigenvectors in an ideal scenario. However, in practice, noise in the covariance matrix perturbs the noise subspace, resulting in a small but non-zero projection. This function calculates the magnitude of this projection using the formula:

$$P = a^{H}(EnE_{n}^{H})a$$

where:

- a is the antenna manifold vector.
- En is the noise subspace eigenvector matrix.

- H denotes the Hermitian (conjugate transpose) operator.

The bearing that produces the smallest projection is considered the best estimate of the signal bearing, as it corresponds to the direction where the signal is strongest relative to the noise.

Value

A complex scalar representing the magnitude of the projection of the antenna manifold vector onto the noise subspace. This value indicates how close the antenna manifold vector is to being orthogonal to the noise subspace.

References

- Paolo, T. de, Cook, T., & Terrill, E. (2007). Properties of HF RADAR Compact Antenna Arrays and Their Effect on the MUSIC Algorithm. *OCEANS 2007*, 1–10. doi:10.1109/oceans.2007.4449265.

`seasonder_createSeaSondeRAPM`

Create a SeaSondeRAPM Object

Description

This function creates a SeaSondeRAPM object to store antenna pattern calibration data.

Usage

```
seasonder_createSeaSondeRAPM(
 calibration_matrix = matrix(complex(real = NA_real_, imaginary = NA_real_), nrow = 3,
    ncol = 0),
  ...
)
```

Arguments

`calibration_matrix`	A 3 x b complex matrix, where b is the number of bearings for calibration.
`...`	Additional named attributes that will be passed to `seasonder_initializeAttributesSeaSondeRAPM`.

Details

The function performs the following operations:

1. Validates the `calibration_matrix` with `seasonder_validateCalibrationMatrixSeaSondeRAPM`.
2. Initializes all other attributes either with default or user-provided values.
3. Merges the initialized attributes into `calibration_matrix`.
4. Sets the object's class to 'SeaSondeRAPM'.

For more details on the attributes, see `seasonder_initializeAttributesSeaSondeRAPM`.

Value

A SeaSondeRAPM object containing a complex matrix with class attribute 'SeaSondeRAPM' and additional attributes for metadata. Row names are set "A13", "A23" and "A33" and column names are set to be the values in BEAR.

See Also

seasonder_validateCalibrationMatrixSeaSondeRAPM, seasonder_initializeAttributesSeaSondeRAPM

Examples

```
# Create a test SeaSondeRAPM object by reading sample file
apm_file <- system.file("css_data/MeasPattern.txt", package = "SeaSondeR")
obj <- seasonder_readSeaSondeRAPMFile(apm_file)
```

---

seasonder_createSeaSondeRCS

Create a SeaSondeRCS object

Description

This generic function creates a SeaSondeRCS object either from a file path or directly from a list containing header and data. When `x` is a character string, the function determines the file type (either "CS", "CSSY" or "CSSW") by analyzing the spectra file and reads it using the appropriate reading function. If `specs_path` is not provided (or set to `rlang::zap()`), the default YAML specifications path corresponding to the detected file type is used.

Usage

```
seasonder_createSeaSondeRCS(x, specs_path = NULL, ...)
```

Arguments

`x`	Either a character string specifying the path to the SeaSonde CS file or a list containing header and data.
`specs_path`	A character string specifying the path to the YAML specifications for the CS file. Used only if `x` is a character string.
`...`	Additional parameters passed to the underlying functions.

Details

For character inputs, the function first checks if the specified file exists. It then determines the file type using `seasonder_find_spectra_file_type`. If the `specs_path` parameter is not provided or is set to `rlang::zap()`, the default specifications file path is obtained using `seasonder_defaultSpecsFilePath` based on the detected file type. The file is then read using the appropriate reading function:

- `seasonder_readSeaSondeCSFile` for CS files.

- `seasonder_readSeaSondeRCSSYFile` for CSSY files.
- `seasonder_readSeaSondeRCSSWFile` for CSSW files.

For list inputs, the SeaSondeRCS object is created directly from the provided header and data. Additionally, a processing step is appended to the object using `seasonder_setSeaSondeRCS_ProcessingSteps` with a creation step text that indicates the source.

Value

A SeaSondeRCS object.

See Also

new_SeaSondeRCS, seasonder_readSeaSondeCSFile, seasonder_readSeaSondeRCSSWFile, seasonder_setSeaSondeRC

Examples

```
# Creating a SeaSondeRCS object from a list
cs_file <- system.file("css_data/CSS_TORA_24_04_04_0700.cs", package = "SeaSondeR")
specs_path <- seasonder_defaultSpecsFilePath("CS")
temp_obj <- seasonder_readSeaSondeCSFile(cs_file, specs_path)
cs_list <- list(header = temp_obj$header, data = temp_obj$data)
rcs_object <- seasonder_createSeaSondeRCS(cs_list)

# Creating a SeaSondeRCS object from a file path using default YAML specifications
rcs_object <- seasonder_createSeaSondeRCS(system.file("css_data/CSS_TORA_24_04_04_0700.cs",
package = "SeaSondeR"))

# Creating a SeaSondeRCS object from a file path with a specified YAML specifications file
rcs_object <- seasonder_createSeaSondeRCS(
  system.file("css_data/CSS_TORA_24_04_04_0700.cs", package = "SeaSondeR"),
  specs_path = seasonder_defaultSpecsFilePath("CS")
)
```

---

seasonder_createSeaSondeRCS.character

Create a SeaSondeRCS object from a file path

Description

This method creates a SeaSondeRCS object by reading a file from the specified file path. It verifies the file's existence, determines the file type ("CS", "CSSY" or "CSSW") using `seasonder_find_spectra_file_type`, and then reads the file using the appropriate function. If `specs_path` is not provided (or is set to `rlang::zap()`), the default YAML specifications file path is retrieved using `seasonder_defaultSpecsFilePath` based on the detected file type.

Usage

```
## S3 method for class 'character'
seasonder_createSeaSondeRCS(x, specs_path = rlang::zap(), endian = "big", ...)
```

Arguments

x	A character string specifying the path to the SeaSonde CS file.
specs_path	A character string specifying the path to the YAML specifications for the CS file. If not provided or set to `rlang::zap()`, the default specifications path for the detected file type is used.
endian	A character string indicating the byte order. Options are "big" (default) or "little".
...	Additional parameters passed to `new_SeaSondeRCS` for creating the object.

Details

The function performs the following steps:

1. Checks if the file specified by `x` exists; if not, it aborts with an error.
2. Determines the file type using `seasonder_find_spectra_file_type`.
3. If `specs_path` is not provided or is set to `rlang::zap()`, retrieves the default YAML specifications path using `seasonder_defaultSpecsFilePath` based on the detected file type.
4. Reads the file using the appropriate function:
   - `seasonder_readSeaSondeCSFile` for CS files.
   - `seasonder_readSeaSondeRCSSYFile` for CSSY files.
   - `seasonder_readSeaSondeRCSSWFile` for CSSW files.
5. Creates a SeaSondeRCS object using `new_SeaSondeRCS` with the header and data obtained from the file.
6. Appends a processing step indicating the creation source via `seasonder_setSeaSondeRCS_ProcessingSteps` with a creation step text generated by `SeaSondeRCS_creation_step_text(x)`.

Value

A SeaSondeRCS object.

See Also

new_SeaSondeRCS, seasonder_find_spectra_file_type, seasonder_defaultSpecsFilePath, seasonder_readSeaSondeCSFile, seasonder_readSeaSondeRCSSYFile, seasonder_readSeaSondeRCSSWFile, seasonder_setSeaSondeRCS_ProcessingSteps, SeaSondeRCS_creation_step_text

Examples

```
# Create a SeaSondeRCS object from a file using the default YAML specifications
rcs_object <- seasonder_createSeaSondeRCS(
  system.file("css_data/CSS_TORA_24_04_04_0700.cs", package = "SeaSondeR")
)

# Create a SeaSondeRCS object from a file with a specified YAML specifications file
rcs_object <- seasonder_createSeaSondeRCS(
  system.file("css_data/CSS_TORA_24_04_04_0700.cs", package = "SeaSondeR"),
  specs_path = seasonder_defaultSpecsFilePath("CS")
)
```

`seasonder_createSeaSondeRCS.list` *Create a SeaSondeRCS object from a list*

Description

This method creates a SeaSondeRCS object directly from a list containing the header and data.

Usage

```
## S3 method for class 'list'
seasonder_createSeaSondeRCS(x, specs_path = NULL, ...)
```

Arguments

`x`	A list with components `header` and `data` required for constructing the SeaSondeRCS object.
`specs_path`	Not used for list inputs.
`...`	Additional parameters that may be used for setting object attributes.

Details

The function creates a new SeaSondeRCS object using `new_SeaSondeRCS` with the provided header and data. It then appends a processing step, generated by `SeaSondeRCS_creation_step_text("list")`, to the object via `seasonder_setSeaSondeRCS_ProcessingSteps`.

Value

A SeaSondeRCS object.

See Also

`new_SeaSondeRCS`, `seasonder_setSeaSondeRCS_ProcessingSteps`, `SeaSondeRCS_creation_step_text`

Examples

```
# Given a list with header and data, create a SeaSondeRCS object
cs_file <- system.file("css_data/CSS_TORA_24_04_04_0700.cs", package = "SeaSondeR")
specs_path <- seasonder_defaultSpecsFilePath("CS")
temp_obj <- seasonder_readSeaSondeCSFile(cs_file, specs_path)
cs_list <- list(header = temp_obj$header, data = temp_obj$data)
rcs_object <- seasonder_createSeaSondeRCS(cs_list)
```

`seasonder_CSSW2CSData` *Transform CSSW Body to SeaSonde CS Data Structure*

Description

This function converts the body structure of a CSSW file into a list of matrices that conform to the data structure required for creating a SeaSondeRCS object. The conversion is performed by mapping specific fields:

SSA1, SSA2, SSA3 Matrices are built using the numeric vectors found in the `cs1a`, `cs2a` and `cs3a` fields respectively.

CS12, CS13, CS23 Each complex cross-spectra matrix is formed by combining the real parts from `c12m`, `c13m` and `c23m` with the corresponding imaginary parts from `c12a`, `c13a` and `c23a`.

QC The quality control matrix is obtained directly from the `csqf` field.

Usage

```
seasonder_CSSW2CSData(body)
```

Arguments

`body`	A list representing the body of a CSSW file. Each element of the list is expected to be a cell containing the following fields: `indx` (which includes an `index`), `cs1a`, `cs2a`, `cs3a`, `c12m`, `c12a`, `c13m`, `c13a`, `c23m`, `c23a` and `csqf`.

Details

Each row in the output matrices corresponds to the index provided by `cell$indx$index` in the input list.

The function first determines the maximum index among the cells in the body, which defines the number of rows for the matrices. Then, it calculates the number of columns for each matrix based on the length of the corresponding vectors from the first cell where they appear. Finally, each cell's data is inserted into the appropriate row of the matrices as indicated by the cell's `indx$index` value.

Value

A list with the following components:

SSA1 A numeric matrix containing self-spectra from `cs1a`.

SSA2 A numeric matrix containing self-spectra from `cs2a`.

SSA3 A numeric matrix containing self-spectra from `cs3a`.

CS12 A complex matrix formed by pairing `c12m` (real) and `c12a` (imaginary).

CS13 A complex matrix formed by pairing `c13m` (real) and `c13a` (imaginary).

CS23 A complex matrix formed by pairing `c23m` (real) and `c23a` (imaginary).

QC A numeric matrix containing the quality control data from `csqf`.

Examples

```
# Example with a single cell
cell <- list(
  indx  = list(index = 1),
  cs1a  = c(1, 2, 3),
  cs2a  = c(4, 5, 6),
  cs3a  = c(7, 8, 9),
  c12m  = c(10, 11, 12),
  c12a  = c(13, 14, 15),
  c13m  = c(16, 17, 18),
  c13a  = c(19, 20, 21),
  c23m  = c(22, 23, 24),
  c23a  = c(25, 26, 27),
  csqf  = c(28, 29, 30)
)
body <- list(cell)
transformed <- seasonder_CSSW2CSData(body)
print(transformed)
```

---

seasonder_CSSW2CSHeader

Transform CSSW Header to SeaSonde CS Header

Description

Extracts the 'cs4h' component from a CSSW header and reorganizes the remaining header information under 'header_csr'.

Usage

```
seasonder_CSSW2CSHeader(header)
```

Arguments

header	A list representing the CSSW header, which must contain a 'cs4h' component.

Value

A transformed list representing a valid SeaSonde CS header with embedded CSSW header information.

`seasonder_CSSW_read_asign` *Read Self Spectra Sign Information from a Connection*

Description

This function reads a raw binary stream from a provided connection, expecting a specific format that contains the sign bits for self spectra values. The data is divided into 3 groups corresponding to: `cs1a`, `cs2a`, and `cs3a`.

Usage

```
seasonder_CSSW_read_asign(connection, key)
```

Arguments

`connection`	A binary connection to read raw bytes from.
`key`	A list containing:
	size An integer specifying the total number of bytes to be read. It must equal 3 times the number of bytes per group.
	key A string identifier (expected to be `"asign"`).

Details

The function performs the following steps:

- Reads `key$size` bytes from the specified connection.
- Verifies that the number of bytes read matches the expected size.
- Checks that the total number of bytes is divisible by 3, allowing equal distribution among the groups.
- Splits the raw byte vector into 3 groups based on the calculated number of bytes per group.
- Converts each byte into its 8-bit binary representation (using `rawToBits`) and flattens the results for each group.

Value

A named list of 3 vectors, each containing bits as integers (0 or 1) for self spectra sign data.

`seasonder_CSSY2CSData` *Transform CSSY Body to SeaSonde CS Data Structure*

Description

This function converts the body structure of a CSSY file into a list of matrices that conform to the data structure required for creating a SeaSondeRCS object. The conversion is performed by mapping specific fields:

SSA1, SSA2, SSA3 Matrices are built using the numeric vectors found in the `cs1a`, `cs2a` and `cs3a` fields respectively.

CS12, CS13, CS23 Each complex cross-spectra matrix is formed by combining the real parts from `c12r`, `c13r` and `c23r` with the corresponding imaginary parts from `c12i`, `c13i` and `c23i`.

QC The quality control matrix is obtained directly from the `csqf` field.

Usage

```
seasonder_CSSY2CSData(body)
```

Arguments

`body`	A list representing the body of a CSSY file. Each element of the list is expected to be a cell containing the following fields: `indx` (which includes an `index`), `cs1a`, `cs2a`, `cs3a`, `c12r`, `c12i`, `c13r`, `c13i`, `c23r`, `c23i` and `csqf`.

Details

Each row in the output matrices corresponds to the index provided by `cell$indx$index` in the input list.

The function first determines the maximum index among the cells in the body, which defines the number of rows for the matrices. Then, it calculates the number of columns for each matrix based on the length of the corresponding vectors from the first cell where they appear. Finally, each cell's data is inserted into the appropriate row of the matrices as indicated by the cell's `indx$index` value.

Value

A list with the following components:

SSA1 A numeric matrix containing self-spectra from `cs1a`.

SSA2 A numeric matrix containing self-spectra from `cs2a`.

SSA3 A numeric matrix containing self-spectra from `cs3a`.

CS12 A complex matrix formed by pairing `c12r` (real) and `c12i` (imaginary).

CS13 A complex matrix formed by pairing `c13r` (real) and `c13i` (imaginary).

CS23 A complex matrix formed by pairing `c23r` (real) and `c23i` (imaginary).

QC A numeric matrix containing the quality control data from `csqf`.

Examples

```
# Example with a single cell
cell <- list(
  indx  = list(index = 1),
  cs1a  = c(1, 2, 3),
  cs2a  = c(4, 5, 6),
  cs3a  = c(7, 8, 9),
  c12r  = c(10, 11, 12),
  c12i  = c(13, 14, 15),
  c13r  = c(16, 17, 18),
  c13i  = c(19, 20, 21),
  c23r  = c(22, 23, 24),
  c23i  = c(25, 26, 27),
  csqf  = c(28, 29, 30)
)
body <- list(cell)
transformed <- seasonder_CSSY2CSData(body)
print(transformed)
```

---

seasonder_CSSY2CSHeader

Transform CSSY Header to SeaSondeRCS Header

Description

This helper function extracts the 'cs4h' component from a CSSY header, removes it from the original header, and embeds the remaining header information within the 'header_csr' field of the CS header.

Usage

```
seasonder_CSSY2CSHeader(header)
```

Arguments

header	A list representing the CSSY header. Must contain a 'cs4h' component.

Value

A transformed header where the primary CS header is taken from 'cs4h' and the remaining CSSY header fields are stored in the 'header_csr' element.

seasonder_CSSY_read_asign

Read Self Spectra Sign Information from a Connection

Description

This function reads a raw binary stream from a provided connection, expecting a specific format that contains the sign bits for self spectra values. The data is divided into 3 groups corresponding to: `cs1a`, `cs2a`, and `cs3a`.

Usage

```
seasonder_CSSY_read_asign(connection, key)
```

Arguments

`connection`	A binary connection to read raw bytes from.
`key`	A list containing:
	size An integer specifying the total number of bytes to be read. It must equal 3 times the number of bytes per group.
	key A string identifier (expected to be `"asign"`).

Value

A named list of 3 vectors. Each vector represents one group (i.e., `cs1a`, `cs2a`, `cs3a`) and contains integers (0 or 1) corresponding to the bits (in little-endian order) extracted from the raw data.

seasonder_CSSY_read_csign

Read Complex Spectral Sign Information from a Connection

Description

This function reads a raw binary stream from a provided connection, expecting a specific format that contains the sign bits for complex spectral values. The data is divided into 6 groups corresponding to: `C13r`, `C13i`, `C23r`, `C23i`, `C12r`, and `C12i`.

Usage

```
seasonder_CSSY_read_csign(connection, key)
```

Arguments

`connection`	A binary connection to read raw bytes from.
`key`	A list containing:
	size An integer specifying the total number of bytes to be read. It must equal 6 times the number of bytes per group.
	key A string identifier (expected to be `"csign"`).

Details

The function performs the following steps:

- Reads `key$size` bytes from the specified connection.
- Checks if enough bytes were read.
- Ensures that the total number of bytes is divisible by 6, allowing equal distribution among the groups.
- Splits the raw byte vector into 6 groups based on the calculated number of bytes per group.
- Converts each byte into its 8-bit representation (using `rawToBits`) and flattens the result.

Value

A named list of 6 vectors. Each vector represents one group (e.g., `C13r`, `C13i`, etc.) and contains integers (0 or 1) corresponding to the bits (in little-endian order) extracted from the raw data.

`seasonder_defaultFOR_parameters`

Default First-Order Radial Processing Parameters

Description

This function returns a list of default parameters for first-order radial processing in CODAR's Radial Suite R7. Each parameter has an equivalent R8 version, where applicable, often expressed in decibels (dB).

Usage

```
seasonder_defaultFOR_parameters()
```

Details

Parameter Descriptions:

1. **nsm (R8: Doppler Smoothing)**
   - **Default value:** 2
   - **Usage:** Sets how many Doppler bins (points) are smoothed. Smoothing helps remove jagged edges in the sea echo spectrum, aiding in locating the null between the first and second order (or noise floor).

- **Recommended values:** Typically 2 to 6. Default in Radial Suite R8 is 2
- **Effects of over-/under-smoothing:**
  - Too high: May smear out the real null, causing the first order to appear wider.
  - Too low: Jagged minima may cause the null to be detected inside the first-order region, making it appear too narrow.

2. **fdown (R8: Null Below Peak Power)**
   - **Default value (R7):** 10
   - **Equivalent in dB (R8):** 10 dB
   - **Usage:** Defines how far below the peak power the algorithm must descend (in dB) before searching for the null that separates the first and second order. This helps avoid including second-order energy as part of the first-order.
   - **Recommended range:** 3.981072 to 31.62278 (6 to 15 dB in R8). Default in Radial Suite R8 is 10 dB
   - **Effects of misconfiguration:**
     - Too large: The null search may be bypassed entirely, causing second-order content to be included in the first order.
     - Too small: The null may be found inside the first-order region, excluding valid Bragg energy.
3. **flim (R8: Peak Power Dropoff)**
   - **Default value (R7):** 100
   - **Equivalent in dB (R8):** 20 dB
   - **Usage:** Once a peak is located, any spectral bins that are more than `flim` below the peak (in linear scale) or `20 dB` below the peak (in dB scale) are excluded from the first-order region.
   - **Recommended range:** 15.84893 to 316.2278 (12 to 25 dB in R8). Default in Radial Suite R8 is 20 dB.
   - **Effects of misconfiguration:**
     - Too high: May include non-Bragg signal and yield spurious high velocity estimates.
     - Too low: May cut out part of the actual Bragg signal, underestimating maximum velocities.
4. **noisefact (R8: Signal to Noise)**
   - **Default value (R7):** 3.981072
   - **Equivalent in dB (R8):** 6 dB
   - **Usage:** Sets the threshold above the noise floor that must be exceeded for the algorithm to accept Doppler bins as potential first-order.
   - **Recommended range:** 3.981072 to 7.943282 (6 to 9 dB in R8). Default in Radial Suite R8 is 6 dB
   - **Effects of misconfiguration:**
     - Too high: Useful Bragg data could be excluded.
     - Too low: Noise or spurious signals may be included as Bragg.
5. **currmax (R8: Maximum Velocity)**
   - **Default value:** 2 m/s

- **Usage:** Sets a maximum radial velocity, preventing first-order limits from extending beyond realistic current speeds for the site.
- **Effects of misconfiguration:**
  - Too high: May include non-Bragg data, producing overestimated velocities.
  - Too low: May exclude valid Bragg data, underestimating velocities.

6. **reject_distant_bragg (Reject Distant Bragg)**
   - **Default value:** TRUE
   - **Usage:** Rejects a first-order region if its limits are farther from the Bragg index (central Doppler bin for zero current) than the width of the region itself. Helps avoid misclassifying strong but isolated signals (e.g., ships) as Bragg.
   - **Recommendation:** Usually keep this enabled unless operating at a site where only strongly biased positive or negative currents are expected.
7. **reject_noise_ionospheric (Reject Noise/Ionospheric)**
   - **Default value:** TRUE
   - **Usage:** Rejects Bragg if the total non-Bragg power in a range cell exceeds the Bragg power by at least the threshold set in `reject_noise_ionospheric_threshold`. Recommended to set as FALSE for 42 MHz systems.
   - **Recommendation:** Enable if the site experiences significant noise.
8. **reject_noise_ionospheric_threshold (Reject Noise/Ionospheric Threshold)**
   - **Default value:** 0
   - **Equivalent in dB:** 0 dB
   - **Usage:** Difference threshold (in dB) for comparing non-Bragg power to Bragg power. If non-Bragg power is higher by this threshold, the Bragg is rejected.
   - **Recommended setting:** Typically 0 dB. Increase only if needed to be less sensitive to noise contamination.

Value

A named list containing the default parameter values.

References

COS. SeaSonde Radial Suite Release 7; CODAR Ocean Sensors (COS): Mountain View, CA, USA, 2013. COS. SeaSonde Radial Suite Release 8; CODAR Ocean Sensors (COS): Mountain View, CA, USA, 2016.

Examples

```
params <- seasonder_defaultFOR_parameters()
print(params)
```

`seasonder_defaultMUSICOptions` *Default Options for the MUSIC Algorithm*

Description

This function returns a list of default options used in the MUSIC algorithm.

Usage

```
seasonder_defaultMUSICOptions()
```

Details

The returned list includes:

- `PPMIN`: Lower threshold value (default is `NULL`).
- `PWMAX`: Upper threshold value (default is `NULL`).
- `smoothNoiseLevel`: Logical flag indicating whether the noise level should be smoothed (`FALSE` by default).
- `doppler_interpolation`: Doppler interpolation factor (default is `2`).
- `MUSIC_parameters`: A numeric vector of default parameters for the MUSIC algorithm, retrieved from `seasonder_defaultMUSIC_parameters()`.
- `discard_low_SNR`: Logical flag to discard solutions with low signal-to-noise ratio (`TRUE` by default).
- `discard_no_solution`: Logical flag to discard cases with no solution (`TRUE` by default).

Value

A list containing the default options for the MUSIC algorithm.

Examples

```
# Retrieve the default options for the MUSIC algorithm
opts <- seasonder_defaultMUSICOptions()
print(opts)
```

`seasonder_defaultMUSIC_parameters` *Default Parameters for MUSIC Algorithm*

Description

This function returns the default parameters for the MUSIC algorithm used in the SeaSondeR package.

Usage

```
seasonder_defaultMUSIC_parameters()
```

Details

The default parameters are:

- 40: Threshold used in `seasonder_MUSICCheckEigenValueRatio`.
- 20: Threshold used in `seasonder_MUSICCheckSignalPowers`.
- 2: Threshold used in `seasonder_MUSICCheckSignalMatrix`.
- 20: Threshold used in `seasonder_MUSICCheckBearingDistance`.

Value

A numeric vector containing the default parameters for the MUSIC algorithm: `c(40, 20, 2, 20)`.

See Also

`seasonder_MUSICTestDualSolutions` to understand the parameters in context.

Examples

```
# Retrieve default parameters
params <- seasonder_defaultMUSIC_parameters()
print(params)
```

seasonder_defaultSpecsFilePath

Get the Default Specifications File Path

Description

This function returns the default file path for the specifications YAML file corresponding to the provided type. The type must be one of the names defined in the default paths (i.e., "CS" or "CSSY").

Usage

```
seasonder_defaultSpecsFilePath(type = "CS")
```

Arguments

type	A character string specifying the type of specifications file. Default is "CS".

Value

A character string representing the full path to the YAML specifications file.

Examples

```
# Retrieve the default CS specifications file path
cs_specs_path <- seasonder_defaultSpecsFilePath("CS")

# Retrieve the default CSSY specifications file path
cssy_specs_path <- seasonder_defaultSpecsFilePath("CSSY")
```

seasonder_defaultSpecsPathForFile

Get the Default Specifications Path for a Spectra File

Description

This function returns the default YAML specifications file path corresponding to a given spectra file. It first determines the file type by analyzing the file content and then retrieves the associated default specifications path.

Usage

```
seasonder_defaultSpecsPathForFile(filepath, endian = "big")
```

Arguments

filepath	A character string specifying the path to the spectra file.
endian	A character string indicating the file's byte order ("big" by default).

Details

The function leverages `seasonder_find_spectra_file_type` to determine whether the file is of type "CS" or "CSSY". It then uses `seasonder_defaultSpecsFilePath` to obtain the corresponding default specifications path.

Value

A character string representing the default YAML specifications file path for the detected file type.

See Also

`seasonder_find_spectra_file_type`, `seasonder_defaultSpecsFilePath`

`seasonder_disableLogs` *Disable log recording in SeaSondeR*

Description

This function disables log recording in the SeaSondeR package. Once disabled, various SeaSondeR functions will no longer output logs.

Usage

```
seasonder_disableLogs()
```

Value

Invisibly returns FALSE, indicating that log recording has been disabled.

Examples

```
seasonder_disableLogs()
```

`seasonder_disableMessages` *Disable message logging in SeaSondeR*

Description

This function disables message logging in the SeaSondeR package. Once disabled, various SeaSondeR functions will no longer output informational messages.

Usage

```
seasonder_disableMessages()
```

Value

logical FALSE indicating that message logging was disabled.

Examples

```
seasonder_disableMessages()
```

---

seasonder_disable_all_debug_points

Disable all debug points in SeaSondeR

Description

This function resets the debug points to the default state ("none").

Usage

```
seasonder_disable_all_debug_points()
```

Value

A character vector containing only the default debug point "none".

Examples

```
seasonder_disable_all_debug_points()
```

---

seasonder_DopplerFreq2Bins

Convert Doppler Frequencies to Doppler Bins

Description

This function converts a set of Doppler frequency values into their corresponding Doppler bin indices within a SeaSondeR object.

Usage

```
seasonder_DopplerFreq2Bins(seasonder_cs_object, doppler_values)
```

Arguments

seasonder_cs_object	A SeaSondeR cross-spectral object containing metadata about the Doppler bins.
doppler_values	A numeric vector specifying the Doppler frequencies to be converted into bin indices.

Details

This function first retrieves the Doppler frequency bins from the given SeaSondeR object using seasonder_getDopplerBinsFrequency in non-normalized form. The spectral resolution, which defines the frequency step size (Δf), is obtained using seasonder_getDopplerSpectrumResolution.

The number of Doppler bins is then determined using seasonder_getnDopplerCells.

With this information, the function calls seasonder_computeDopplerFreq2Bins to determine the corresponding bin indices for each input Doppler frequency.

Value

An integer vector of Doppler bin indices corresponding to the input Doppler frequencies. Values that fall outside the valid bin range are assigned NA.

See Also

seasonder_Bins2NormalizedDopplerFreq for the reverse operation. seasonder_computeDopplerFreq2Bins for the core computation logic.

---

seasonder_DopplerFreq2NormalizedDopplerFreq

Convert Doppler Frequencies to Normalized Doppler Frequencies

---

Description

This function converts Doppler frequencies (in Hz) into their corresponding normalized Doppler frequencies within a SeaSondeR object.

Usage

```
seasonder_DopplerFreq2NormalizedDopplerFreq(
  seasonder_cs_object,
  doppler_values
)
```

Arguments

seasonder_cs_object	A SeaSondeR cross-spectral object containing metadata about the Doppler bins.
doppler_values	A numeric vector specifying the Doppler frequencies (in Hz) to be converted into normalized Doppler frequencies.

Details

The function follows these steps:

1. Calls `seasonder_DopplerFreq2Bins` to convert the input Doppler frequencies into Doppler bin indices.
2. Calls `seasonder_Bins2NormalizedDopplerFreq` to obtain the corresponding normalized Doppler frequencies.

The normalized Doppler frequency is computed as:

$$f_{doppler} = f_{norm} \times f_{bragg}$$

where:

- $f_{doppler}$ is the Doppler frequency in Hz,
- f_{norm} is the normalized Doppler frequency,
- f_{bragg} is the Bragg frequency, computed based on radar wavelength.

This function ensures consistency by mapping input frequencies to their closest bin representation before normalization.

Value

A numeric vector of normalized Doppler frequencies corresponding to the input Doppler values.

See Also

`seasonder_DopplerFreq2Bins` for converting Doppler frequencies to bin indices. `seasonder_Bins2NormalizedDopplerF` for converting bin indices to normalized frequencies.

---

seasonder_enableLogs *Enable log recording in SeaSondeR*

Description

This function enables log recording in the SeaSondeR package. Once enabled, various SeaSondeR functions will output logs.

Usage

```
seasonder_enableLogs()
```

Value

Invisibly returns TRUE, indicating that log recording has been enabled.

Examples

```
seasonder_enableLogs()
```

seasonder_enableMessages

Enable message logging in SeaSondeR

Description

This function enables message logging in the SeaSondeR package. Once enabled, various SeaSondeR functions will output informational messages.

Usage

```
seasonder_enableMessages()
```

Value

logical TRUE indicating that message logging was enabled.

Examples

```
seasonder_enableMessages()
```

seasonder_enable_debug_points

Enable debug points in SeaSondeR

Description

This function adds one or more debug points to the list of enabled debug points.

Usage

```
seasonder_enable_debug_points(debug_points)
```

Arguments

debug_points	A character vector of debug point names to enable.

Value

Updated character vector of enabled debug points.

Examples

```
seasonder_enable_debug_points("example_debug")
```

`seasonder_estimateReferenceNoiseNormalizedLimits` *Estimate Reference Noise Limits in Normalized Doppler Frequency*

Description

This function estimates the reference noise limits for normalized Doppler frequencies in a SeaSondeR cross-spectral object. These limits define the frequency range over which the noise floor is assessed for first-order region (FOR) detection.

Usage

```
seasonder_estimateReferenceNoiseNormalizedLimits(
  seasonder_cs_object,
  low_limit = 0.95,
  high_limit = 1
)
```

Arguments

`seasonder_cs_object`	A SeaSondeRCS object containing Doppler frequency metadata.
`low_limit`	Optional. A numeric value representing the fraction of the maximum normalized Doppler frequency to be used as the lower bound for noise estimation. Default is 0.95.
`high_limit`	Optional. A numeric value representing the fraction of the maximum normalized Doppler frequency to be used as the upper bound for noise estimation. Default is 1.0.

Details

The function operates as follows:

1. Retrieves the Doppler bin frequencies in normalized units (relative to the Bragg frequency) via `seasonder_getDopplerBinsFrequency`.
2. Computes the noise limits by scaling the maximum normalized Doppler frequency using the provided `low_limit` and `high_limit` factors:
   - The lower bound is given by `max(freq) * low_limit`.
   - The upper bound is given by `max(freq) * high_limit`.
3. The default empirical choice for the lower bound (56.5% in the original calibration process) is adjustable via the `low_limit` parameter. This parameter was determined through an iterative process where the initial lower bound was decreased in increments (e.g., 0.5%) until the computed noise floor closely matched the reference provided by the AnalyseSpectra Tool in Radial Suite R8.

This approach is crucial for setting the signal-to-noise ratio (SNR) thresholds used in FOR detection.

Value

A numeric vector of length two, representing the lower and upper reference noise limits in normalized Doppler frequency.

See Also

`seasonder_getDopplerBinsFrequency` for retrieving Doppler bin frequencies.

---

seasonder_exportCSVMUSICTable

Export MUSIC Table to CSV

Description

This function exports the MUSIC detection table from a `SeaSondeRCS` object to a CSV file.

Usage

```
seasonder_exportCSVMUSICTable(seasonder_cs_object, filepath)
```

Arguments

`seasonder_cs_object`	A `SeaSondeRCS` object containing MUSIC detection data.
`filepath`	A character string specifying the path to the output CSV file.

Details

This function performs the following steps:

1. Generates a MUSIC table using `seasonder_exportMUSICTable`.
2. Converts the resulting table to a data frame.
3. Writes the data frame to the specified CSV file using `data.table::fwrite`.

Value

The function returns `NULL` invisibly. The output is saved to the specified file.

See Also

- `seasonder_exportMUSICTable`
- `fwrite`

Examples

```
  # Prepare a SeaSondeRCS object for examples, including APM
  apm_file <- system.file("css_data/MeasPattern.txt", package = "SeaSondeR")
  apm_obj <- seasonder_readSeaSondeRAPMFile(apm_file)
  specs_path <- seasonder_defaultSpecsFilePath("CS")
  cs_obj <- seasonder_createSeaSondeRCS(
    system.file("css_data/CSS_TORA_24_04_04_0700.cs", package = "SeaSondeR"),
    specs_path = specs_path,
    seasonder_apm_object = apm_obj
  )
cs_obj <- seasonder_initMUSICData(
 cs_obj,
 range_cells = c(rep(5,11), rep(4,11)),
 doppler_bins = c(c(669:679),c(674:684))
)
cs_obj <- seasonder_runMUSIC(cs_obj)
  # Export MUSIC table to a temporary CSV file
  tmpfile <- tempfile(fileext = ".csv")
  seasonder_exportCSVMUSICTable(cs_obj, tmpfile)
  print(tmpfile)
```

`seasonder_exportCTFRangeInfo`

Export CTF Range Information to a File

Description

This function writes the formatted CTF range information, generated from a SeaSondeRCS object, to a specified file.

Usage

```
seasonder_exportCTFRangeInfo(seasonder_cs_object, file, tableStart = "")
```

Arguments

`seasonder_cs_object`	A SeaSondeRCS object containing the relevant MUSIC processing data.
`file`	A character string specifying the output file path where the range information will be written.
`tableStart`	A character string to prepend to the table output. Defaults to an empty string.

Details

The function internally calls `seasonder_exportCTFRangeInfo_string` to obtain a formatted string of range information. It then writes this output string to the specified file. Additionally, it returns the extracted range information invisibly, allowing further processing if necessary.

Value

Invisibly returns a data frame containing the range information.

Examples

```
# Prepare a SeaSondeRCS object with valid data
apm_file <- system.file("css_data/MeasPattern.txt", package = "SeaSondeR")
apm_obj <- seasonder_readSeaSondeRAPMFile(apm_file)
cs_file <- system.file("css_data/CSS_TORA_24_04_04_0700.cs", package = "SeaSondeR")
cs_obj <- seasonder_createSeaSondeRCS(
  cs_file,
  seasonder_apm_object = apm_obj
)
# Export CTF range information to a temporary text file
range_info <- seasonder_exportCTFRangeInfo(cs_obj, tempfile(fileext = ".txt"))
```

---

seasonder_exportLLUVRadialMetrics

Export LLUV Radial Metrics to a File

Description

This function extracts radial metrics from a SeaSondeRCS object and formats them for export using defined mustache templates. The formatted output, which includes MUSIC parameters, antenna pattern corrections, noise thresholds, and other spectral metrics, is written to a specified file. Additionally, the function returns the computed radial metrics as a data frame.

Usage

```
seasonder_exportLLUVRadialMetrics(seasonder_cs_object, LLUV_path, ...)
```

Arguments

`seasonder_cs_object`	A SeaSondeRCS object containing MUSIC detection data and related metadata.
`LLUV_path`	A character string specifying the output file path for the LLUV radial metrics.
`...`	Additional arguments passed to `seasonder_exportRadialMetrics`.

Details

The function performs the following steps:

1. Retrieves the radial metrics from the SeaSondeRCS object using `seasonder_exportRadialMetrics`.
2. Obtains MUSIC parameters and antenna pattern attributes from the object.
3. Formats numeric values using predefined formats for each column.

4. Renders a data template (from "LLUV_RDM1_data.mustache") with the formatted radial metrics.
5. Generates a deterministic UUID from the rendered data.
6. Renders an overall LLUV template (from "LLUV_RDM1.mustache") that incorporates the radial parameters, formatted data, header information, and the generated UUID.
7. Writes the rendered LLUV content to the file specified by `LLUV_path`.

Value

Invisibly returns a data frame containing the radial metrics used in the export.

Examples

```
# Prepare a SeaSondeRCS object with MUSIC data
apm_file <- system.file("css_data/MeasPattern.txt", package = "SeaSondeR")
apm_obj <- seasonder_readSeaSondeRAPMFile(apm_file)
cs_file <- system.file("css_data/CSS_TORA_24_04_04_0700.cs", package = "SeaSondeR")
cs_obj <- seasonder_createSeaSondeRCS(cs_file, seasonder_apm_object = apm_obj)
FOR <- seasonder_getSeaSondeRCS_FOR(cs_obj)
cs_obj <- seasonder_setSeaSondeRCS_FOR(cs_obj,FOR[4:5])
# Optionally, run MUSIC in FOR context to populate MUSIC data
cs_obj <- seasonder_runMUSICInFOR(cs_obj)
radial_metrics <- seasonder_exportLLUVRadialMetrics(cs_obj, tempfile(fileext = ".ruv"))
head(radial_metrics)
```

`seasonder_exportMUSICTable` *Export MUSIC Table from SeaSondeRCS Object*

Description

This function generates a table containing detailed MUSIC detection data from a `SeaSondeRCS` object. The output table includes geographic coordinates, signal parameters, and other metadata for each MUSIC detection.

Usage

```
seasonder_exportMUSICTable(seasonder_cs_object)
```

Arguments

`seasonder_cs_object`
: A `SeaSondeRCS` object containing MUSIC detection data and related metadata.

Details

This function performs the following operations:

1. Retrieves the timestamp (`nDateTime`) from the header of the `SeaSondeRCS` object. Defaults to `as.POSIXct(0)` if unavailable.
2. Initializes an empty data frame with predefined columns.
3. Retrieves MUSIC detection data, processes the Direction of Arrival (DOA) and geographic coordinates (`lonlat`), and unnests these fields.
4. Converts MUSIC bearings to geographic bearings using the associated Antenna Pattern Matrix (APM) object.
5. Computes additional metrics such as signal power in dB, signal-to-noise ratio (SNR), and DOA peak response in dB.
6. Appends the timestamp to the table and reorders columns for clarity.

Value

A data frame with the following columns:

- `datetime`: Timestamp of the data.
- `longitude`: Geographic longitude of the detection.
- `latitude`: Geographic latitude of the detection.
- `range_cell`: Range cell number.
- `range`: Range in kilometers.
- `doppler_bin`: Doppler bin number.
- `doppler_freq`: Doppler frequency.
- `radial_velocity`: Radial velocity in m/s.
- `signal_power`: Signal power.
- `bearing`: Geographic bearing in degrees.
- `bearing_raw`: Original MUSIC bearing in degrees.
- `noise_level`: Noise level in dB.
- `signal_power_db`: Signal power in dB.
- `SNR`: Signal-to-noise ratio in dB.
- `DOA_peak_resp_db`: DOA peak response in dB.

See Also

- `seasonder_getSeaSondeRCS_MUSIC`
- `seasonder_MUSICBearing2GeographicalBearing`
- `seasonder_getSeaSondeRAPM_AntennaBearing`

Examples

```
  # Load sample CSS and APM files
  cs_file  <- system.file("css_data/CSS_TORA_24_04_04_0700.cs", package = "SeaSondeR")
  apm_file <- system.file("css_data/MeasPattern.txt",       package = "SeaSondeR")
  apm_obj  <- seasonder_readSeaSondeRAPMFile(apm_file)
  # Create SeaSondeRCS object with APM
  cs_obj <- seasonder_createSeaSondeRCS(cs_file, seasonder_apm_object = apm_obj)
FOR <- seasonder_getSeaSondeRCS_FOR(cs_obj)
cs_obj <- seasonder_setSeaSondeRCS_FOR(cs_obj,FOR[4:5])

  # Run MUSIC algorithm (in FOR context) if MUSIC data is available:
  cs_obj <- seasonder_runMUSICInFOR(cs_obj)
  # Export MUSIC table
  music_table <- seasonder_exportMUSICTable(cs_obj)
  print(music_table)
```

---

seasonder_exportRadialMetrics

Export Radial Metrics from a SeaSondeRCS Object

Description

This function extracts and formats radial metrics from a SeaSondeRCS object for export. It processes the MUSIC table, computes various spectral metrics, applies antenna pattern corrections, and combines the results into a final data frame formatted according to predefined column specifications.

Usage

```
seasonder_exportRadialMetrics(seasonder_cs_object, AngSeg = list())
```

Arguments

`seasonder_cs_object`	A SeaSondeRCS object containing MUSIC detection data and related metadata.
`AngSeg`	An optional list of angular segments to be applied to the vector flag field (VFLG). Each element should be a numeric vector of length 3 defining a segment. Default is an empty list.

Details

The function proceeds as follows:

1. Retrieves the MUSIC table using `seasonder_getSeaSondeRCS_MUSIC` and the associated APM object.
2. Defines a template row with 34 predefined columns, initializing most numeric values to NA, except for specific defaults such as `MSA1`, `MDA1`, and `MDA2` (set to 1440L).

3. Copies basic numeric fields and computes additional fields from the MUSIC table, such as the radial velocity (scaled by 100), range, range cell, doppler cell (shifted by -1), eigenvalue ratio, signal power ratio, and offset power ratio.
4. Computes the metric MDRJ by applying the function seasonder_computeMDRJ on the MUSIC row.
5. Extracts eigen decomposition results from each MUSIC row to populate the eigenvalue fields (MEI1, MEI2, MEI3).
6. Processes the DOA solutions stored in each MUSIC row: - For solutions retained as "single", geographic bearing corrections are applied to populate MSA1. - For dual-bearing solutions, the first two elements of the DOA bearings populate MDA1 and MDA2, respectively.
7. Computes additional spectral metrics such as the self-spectra conversion to dB (fields MA1S, MA2S, and MA3S) after subtracting the noise level (obtained for each antenna).
8. Based on the retained solution type (either "single" or "dual"), assigns location data (if available), sets selection flags, and computes additional output metrics (e.g., PPFG and PWFG).
9. Finally, all rows are combined into a data frame. If angular segments are provided, additional modifications to the vector flag (VFLG) are applied.

Value

A data frame with 34 columns containing the computed radial metrics. The columns include geographic coordinates, velocity components, range, bearing information, signal power metrics, noise thresholds, and computed spectral parameters.

Examples

```
# Prepare a SeaSondeRCS object with MUSIC data
apm_file <- system.file("css_data/MeasPattern.txt", package = "SeaSondeR")
apm_obj <- seasonder_readSeaSondeRAPMFile(apm_file)
cs_file <- system.file("css_data/CSS_TORA_24_04_04_0700.cs", package = "SeaSondeR")
cs_obj <- seasonder_createSeaSondeRCS(cs_file, seasonder_apm_object = apm_obj)
FOR <- seasonder_getSeaSondeRCS_FOR(cs_obj)
cs_obj <- seasonder_setSeaSondeRCS_FOR(cs_obj,FOR[4:5])
# Run MUSIC algorithm to populate MUSIC data
cs_obj <- seasonder_runMUSICInFOR(cs_obj)
radial_metrics <- seasonder_exportRadialMetrics(cs_obj, AngSeg = list(c(5, 30, 60)))
head(radial_metrics)
```

---

seasonder_exportRangeInfo

Export Range Information from a SeaSondeRCS Object

Description

This function computes and exports range-related information based on the MUSIC data stored in a SeaSondeRCS object. The output table includes range cell identifiers, range values, noise levels (for each antenna), first-order region (FOR) boundaries, and counts of detections classified as single or dual solutions.

Usage

```
seasonder_exportRangeInfo(seasonder_cs_object)
```

Arguments

`seasonder_cs_object`
: A SeaSondeRCS object containing MUSIC data and associated metadata.

Details

The function performs the following operations:

1. Extracts key fields from the MUSIC data: range cell, range, and the retained solution type.
2. Aggregates counts of detections classified as single versus dual solutions.
3. Retrieves noise levels (in dB) for each antenna.
4. Obtains FOR boundaries using a dedicated export function and adjusts them based on the Doppler interpolation factor.
5. Merges the aggregated detection counts, noise levels, and FOR boundaries by range cell.
6. Selects and reorders the output columns.

Value

A data frame with the following columns:

SPRC Range cell identifier.

RNGC Range (in appropriate units).

NF01 Noise level (in dB) for antenna 1.

NF02 Noise level (in dB) for antenna 2.

NF03 Noise level (in dB) for antenna 3.

ALM1 Lower FOR boundary (after Doppler interpolation).

ALM2 Upper FOR boundary (after Doppler interpolation) for the first boundary set.

ALM3 Lower FOR boundary (after Doppler interpolation) for the second boundary set.

ALM4 Upper FOR boundary (after Doppler interpolation) for the second boundary set.

NVSC Count of detections classified as "single".

NVDC Count of detections classified as "dual".

NVAC Total adjusted count (NVSC plus twice NVDC).

Examples

```
# Prepare a SeaSondeRCS object with MUSIC data
apm_file <- system.file("css_data/MeasPattern.txt", package = "SeaSondeR")
apm_obj <- seasonder_readSeaSondeRAPMFile(apm_file)
cs_file <- system.file("css_data/CSS_TORA_24_04_04_0700.cs", package = "SeaSondeR")
cs_obj <- seasonder_createSeaSondeRCS(cs_file, seasonder_apm_object = apm_obj)
FOR <- seasonder_getSeaSondeRCS_FOR(cs_obj)
```

```
cs_obj <- seasonder_setSeaSondeRCS_FOR(cs_obj,FOR[4:5])
# Run MUSIC algorithm to populate MUSIC data
cs_obj <- seasonder_runMUSICInFOR(cs_obj)
range_info <- seasonder_exportRangeInfo(cs_obj)
head(range_info)
```

`seasonder_extractFOR` *Extract First Order Region (FOR) Spectral Data*

Description

This function extracts the spectral power corresponding to the First Order Region (FOR) from a given self-spectra (SS) matrix. It retrieves the spectral values within the Doppler bins identified as part of the positive and negative Bragg regions.

Usage

```
seasonder_extractFOR(seasonder_cs_object, spectrum, FOR)
```

Arguments

`seasonder_cs_object`	A `SeaSondeRCS` object containing the spectral data.
`spectrum`	A numeric matrix representing the self-spectra data for a single range cell.
`FOR`	A list containing the Doppler bin indices defining the FOR region, with two elements: - `negative_FOR`: A numeric vector of Doppler bins for the negative Bragg region. - `positive_FOR`: A numeric vector of Doppler bins for the positive Bragg region.

Details

The function performs the following steps:

1. **Initialize Empty Matrices:** Creates empty matrices to store extracted spectral data.
2. **Extract Negative FOR Data:**
   - If the negative FOR bins exist, calls `seasonder_extractSeaSondeRCS_dopplerRanges_from_SSdata` to extract the corresponding spectral values.
3. **Extract Positive FOR Data:**
   - If the positive FOR bins exist, extracts the spectral values using the same function.
4. **Return Extracted Spectral Data:** Outputs a list containing the extracted negative and positive Bragg region spectral data.

This function is primarily used for filtering and validating the first-order Bragg region in SeaSonde radar processing.

Value

A list with two elements:

- `negative_FOR`: A matrix containing spectral power for the negative Bragg region.
- `positive_FOR`: A matrix containing spectral power for the positive Bragg region.

See Also

- `seasonder_extractSeaSondeRCS_dopplerRanges_from_SSdata` for extracting Doppler bin subsets.
- `seasonder_filterFORAmplitudes` for filtering weak FOR detections.

---

seasonder_extractSeaSondeRCS_dopplerRanges_from_SSdata

Extract Doppler Ranges from Self-Spectra Data Matrix

Description

This function slices a self-spectra data matrix by selecting the columns corresponding to the specified Doppler cells.

Usage

```
seasonder_extractSeaSondeRCS_dopplerRanges_from_SSdata(SSmatrix, doppler_cells)
```

Arguments

`SSmatrix`	A matrix containing self-spectra data, where columns represent Doppler bins.
`doppler_cells`	A numeric vector specifying the indices of the Doppler bins to extract.

Details

The function extracts a subset of columns from the self-spectra matrix. No explicit validation is currently performed to verify that the provided Doppler cell indices fall within the range of the matrix columns.

Value

A matrix containing only the columns corresponding to the selected Doppler cells.

`seasonder_extrapolateAPM` *Extrapolate SeaSondeR APM Matrix*

Description

This function performs linear extrapolation on the SeaSondeR APM measurement matrix. It adds `n` extrapolated columns to both the left and right sides of the matrix.

Usage

```
seasonder_extrapolateAPM(seasonder_apm_object, n = 1)
```

Arguments

`seasonder_apm_object`	A matrix containing SeaSondeR APM measurements. Its attributes include "BEAR" (numeric vector of bearings) and "BearingResolution" (numeric resolution).
`n`	An integer specifying how many extrapolated columns to add on each side (default is 1).

Details

The function retrieves the original bearing vector from the APM object using `seasonder_getSeaSondeRAPM_BEAR` and obtains the bearing resolution (attribute "BearingResolution"). If `n == 0`, the original matrix is returned unchanged. For `n > 0`, new bearings are generated for both sides using the resolution. The left side is extrapolated using the slope computed from the first two columns of the matrix, and the right side is extrapolated using the slope from the last two columns. The new columns are then combined with the original matrix, and the column names and the "BEAR" attribute are updated to reflect the complete set of bearings.

Value

A modified matrix with `n` extrapolated columns added to both sides. The column names and the "BEAR" attribute are updated with the new bearings, while the "BearingResolution" attribute remains unchanged.

Examples

```
# Extrapolate loops for a test SeaSondeRAPM object
apm_file <- system.file("css_data/MeasPattern.txt", package = "SeaSondeR")
obj <- seasonder_readSeaSondeRAPMFile(apm_file)
result <- seasonder_extrapolateAPM(obj, n = 1)
```

`seasonder_filterFORAmplitudes` *Filter First Order Region (FOR) Based on Amplitude Thresholds*

Description

This function filters the First Order Region (FOR) Doppler bins based on amplitude thresholds. It applies a combination of noise-based and peak power-based criteria to remove low-amplitude bins that do not meet the required signal-to-noise ratio.

Usage

```
seasonder_filterFORAmplitudes(seasonder_cs_object)
```

Arguments

`seasonder_cs_object`
: A `SeaSondeRCS` object containing spectral data and FOR parameters.

Details

Steps in FOR Amplitude Filtering:

1. **Retrieve First Order Parameters:**
   - The function extracts `flim` (Null Below Peak Power) and `noisefact` (Signal-to-Noise Factor) from `seasonder_getFOR_parameters`.
2. **Compute Noise Levels:**
   - Calls `seasonder_computeNoiseLevel` to estimate the average noise level across all range cells.
   - Converts the noise level into a filtering threshold by multiplying it by `noisefact`.
3. **Extract Smoothed Self-Spectra Data:**
   - Retrieves the smoothed self-spectra (SSA3) using `seasonder_getSeaSondeRCS_FOR_SS_Smoothed`.
   - Extracts the FOR spectral power for each range cell using `seasonder_extractFOR`.
4. **Determine Filtering Thresholds:**
   - Computes a power threshold for each FOR region by taking the maximum amplitude in the FOR region and dividing it by `flim`.
5. **Apply Filtering Conditions:**
   - A Doppler bin is retained if its power is greater than:
     - The noise threshold (computed from `noisefact`).
     - The power threshold computed from `flim`.
6. **Store Filtered FOR in Object:**
   - Updates the SeaSondeRCS object with the filtered FOR bins.

This filtering ensures that only strong, reliable first-order Bragg signals are retained, reducing the impact of noise and second-order contamination.

Value

The updated SeaSondeRCS object with the filtered FOR bins.

See Also

- seasonder_computeNoiseLevel for computing noise levels.
- seasonder_getSeaSondeRCS_FOR_SS_Smoothed for retrieving smoothed self-spectra.
- seasonder_extractFOR for extracting FOR spectral power.
- seasonder_setSeaSondeRCS_FOR for storing the filtered FOR data.

---

seasonder_findFORNulls

Identify Nulls in First Order Region (FOR) Across All Range Cells

---

Description

This function locates the null points in the First Order Region (FOR) of a SeaSondeR cross-spectral object. It smooths the self-spectra (SS) data, extracts the relevant Doppler bins, and determines the boundaries of the first-order Bragg region for each range cell.

Usage

```
seasonder_findFORNulls(seasonder_cs_object)
```

Arguments

seasonder_cs_object
: A SeaSondeRCS object containing spectral data and FOR parameters.

Details

The function follows these steps:

1. **Smooth the Self-Spectra Data:** Calls seasonder_SmoothFORSS to apply a running mean filter.
2. **Extract Smoothed Self-Spectra:** Retrieves the processed SS matrix using seasonder_getSeaSondeRCS_FOR_SS_Smoo
3. **Identify the Doppler Center Bin:** Determines the central Doppler bin using seasonder_getCenterDopplerBin.
4. **Segment the Spectrum:** Splits the smoothed SS data into:
   - The negative Bragg region (left side of the Doppler spectrum).
   - The positive Bragg region (right side of the Doppler spectrum).
5. **Find Nulls in Each Region:** Uses seasonder_findFORNullsInSSMatrix to identify the null positions.
6. **Store Results:** Extracts:
   - The First Order Region (FOR).
   - The maximum power (MAXP).
   - The Doppler bin index of the maximum power (MAXP.bin).
7. **Update the SeaSondeRCS Object:** Saves the detected FOR boundaries and related metrics.

Value

The updated SeaSondeRCS object with the computed FOR nulls, maximum power, and bin indices.

See Also

- seasonder_findFORNullsInSpectrum for processing individual spectra.
- seasonder_findFORNullsInSSMatrix for batch processing spectra across multiple range cells.
- seasonder_getSeaSondeRCS_FOR_SS_Smoothed for retrieving smoothed SS data.

---

seasonder_findFORNullsInFOR

Find Nulls in First Order Region (FOR)

Description

This function locates the null point in the First Order Region (FOR) spectrum, which separates the first-order Bragg peak from second-order energy or the noise floor.

Usage

```
seasonder_findFORNullsInFOR(
  FOR,
  start_point_P,
  doppler_bins,
  left_region = FALSE
)
```

Arguments

FOR	A numeric vector representing the power spectrum in the FOR region.
start_point_P	A numeric value representing the power threshold at which the search for the null point begins.
doppler_bins	A numeric vector containing the Doppler bins corresponding to the spectrum in FOR.
left_region	A logical value indicating whether the null is being searched for in the negative Bragg region. Default is FALSE.

Details

The function follows these steps to determine the null point:

1. If left_region is TRUE, the FOR spectrum and Doppler bins are reversed.
2. The power spectrum is transformed to facilitate peak identification:
    - The absolute values of the power are taken and multiplied by -1.

- The `start_point_P` threshold is also inverted.

3. The function identifies the first local maximum in the transformed spectrum that exceeds `start_point_P`.
4. The corresponding Doppler bin at the detected peak is returned as the null position.

The function relies on `pracma::findpeaks` to identify the peak.

Value

A numeric value representing the Doppler bin at the detected null position.

See Also

- `seasonder_findFORNullsInSpectrum` for locating nulls in a full spectrum.
- `findpeaks` for peak detection.

`seasonder_findFORNullsInSpectrum`
Identify Nulls in First Order Region (FOR) Spectrum

Description

This function locates the null points in the First Order Region (FOR) of a Doppler spectrum. These nulls define the boundaries separating the first-order Bragg peak from the surrounding noise or second-order energy.

Usage

```
seasonder_findFORNullsInSpectrum(
  seasonder_cs_object,
  spectrum,
  doppler_bins,
  negative_Bragg_region = FALSE
)
```

Arguments

`seasonder_cs_object`	A `SeaSondeRCS` object containing the spectral data.
`spectrum`	A numeric vector representing the power spectrum to analyze.
`doppler_bins`	A numeric vector containing the Doppler bins corresponding to the spectrum.
`negative_Bragg_region`	A logical value indicating whether the function should analyze the negative Bragg region. Default is `FALSE`.

Details

The function executes the following steps:

1. **Retrieve First Order Settings:** The function extracts the `fdown` parameter, which defines the drop-off level relative to the maximum power.
2. **Prepare the Spectrum:**
   - Convert all values to negative absolute magnitudes to facilitate peak detection.
   - Reverse the spectrum and Doppler bins if analyzing the negative Bragg region.
3. **Find the Main Spectral Peak:**
   - The function identifies the first major peak using `findpeaks` with at least two consecutive increases and decreases.
   - The search is limited to the portion of the spectrum beyond this peak.
4. **Determine the First Order Boundaries:**
   - The maximum power (`MAXP`) is found along with its bin index (`MAXP.bin`).
   - A threshold value `start_point_P` is computed as `MAXP / fdown` to establish the cutoff point for the null search.
5. **Search for Nulls:** The spectrum is split into left and right sections:
   - The right-side spectrum is analyzed using `seasonder_findFORNullsInFOR` to find the right null.
   - The left-side spectrum undergoes the same process but reversed.
6. **Output the Results:** The function returns a list containing:
   - The sequence of Doppler bins defining the FOR region.
   - The maximum power detected (`MAXP`).
   - The Doppler bin index where `MAXP` occurred (`MAXP.bin`).

Value

A list with three elements:

- `FOR`: A sequence of Doppler bins defining the first order region.
- `MAXP`: The maximum power found in the spectrum.
- `MAXP.bin`: The Doppler bin index of the maximum power.

See Also

- `seasonder_findFORNullsInFOR` for detecting nulls within a selected region.
- `findpeaks` for peak identification.
- `seasonder_getFOR_parameters` for retrieving FOR settings.

`seasonder_findFORNullsInSSMatrix` *Identify Nulls in First Order Region (FOR) for a Self-Spectra Matrix*

Description

This function applies the null-finding algorithm to each row of a self-spectra (SS) matrix, determining the boundaries of the First Order Region (FOR) for each range cell.

Usage

```
seasonder_findFORNullsInSSMatrix(
  seasonder_cs_object,
  SS,
  doppler_bins,
  negative_Bragg_region = FALSE
)
```

Arguments

`seasonder_cs_object`	A `SeaSondeRCS` object containing spectral data and FOR parameters.
`SS`	A numeric matrix representing the self-spectra data, where rows correspond to range cells and columns correspond to Doppler bins.
`doppler_bins`	A numeric vector indicating the Doppler bins corresponding to the columns of `SS`.
`negative_Bragg_region`	A logical value indicating whether to analyze the negative Bragg region. Default is `FALSE`.

Details

This function processes each row of the self-spectra matrix, treating each row as an independent spectrum for which the FOR nulls are identified. The nulls define the boundaries of the first-order Bragg region.

Processing Steps:

1. Iterate through each row of the `SS` matrix.
2. Extract the power spectrum for the corresponding range cell.
3. Apply `seasonder_findFORNullsInSpectrum` to determine the null positions.
4. Store the results in a named list, where each entry corresponds to a range cell.

Value

A named list where each entry corresponds to a range cell, containing the detected FOR null positions.

See Also

- seasonder_findFORNullsInSpectrum for detecting nulls in a single spectrum.
- seasonder_findFORNulls for high-level null detection across all spectra.

---

seasonder_find_spectra_file_type
Determine the Spectra File Type

Description

This function identifies the type of a spectra file (either "CS" or "CSSY") by reading its header block based on YAML specifications. It first attempts to read a key size block using the CSSY specifications, and if that fails, it reopens the file and tries to read the CS header block.

Usage

```
seasonder_find_spectra_file_type(filepath, endian = "big")
```

Arguments

filepath	A character string specifying the path to the spectra file.
endian	A character string indicating the file's byte order ("big" by default).

Details

The function sets up error handling parameters and uses YAML specifications retrieved via seasonder_readYAMLSpecs and seasonder_defaultSpecsFilePath. It opens the file in binary read mode and ensures the connection is closed upon exit. If reading the key size block fails, it reopens the file to try reading the CS header block. The final file type is determined by the key returned from the file block.

Value

A character string representing the spectra file type ("CS" or "CSSY").

`seasonder_getBinsRadialVelocity` *Calculate Radial Velocities for Each Doppler Bin*

Description

Computes the radial velocities for each Doppler bin interval's high boundary for a SeaSonde radar cross-section (CS) object, as typically visualized in SpectraPlotterMap. This function utilizes the Doppler shift frequency alongside the radar's wave number and Bragg frequency to transform frequency measurements into radial velocities. The calculation is based on the relationship between the Doppler shift frequency and the velocity of surface currents within the radar's field of view.

Usage

```
seasonder_getBinsRadialVelocity(seasonder_cs_object)
```

Arguments

`seasonder_cs_object`
: A SeaSondeRCS object created using `seasonder_createSeaSondeRCS`. This object contains the necessary data for calculating the Doppler bins frequencies and, subsequently, radial velocities.

Details

Specifically, the radial velocity $v = (Freq - BraggFreq)/(2 * k_0)$ is used, where v is the radial velocity, $Freq$ is the Doppler shift frequency for the bin, $BraggFreq$ is the Bragg frequency (negative for frequencies below 0 and positive for frequencies equal or above 0), and k_0 is the radar wave number divided by 2π.

Value

A numeric vector containing the radial velocities (in m/s) for each Doppler bin, calculated for the high boundary of each Doppler bin interval. The velocities provide insight into the scatterers' radial movement within the radar's observation area.

See Also

`seasonder_getDopplerBinsFrequency`, `seasonder_getBraggDopplerAngularFrequency`, `seasonder_getRadarWaveNu`

`seasonder_getBraggDopplerAngularFrequency` *Calculate the Bragg Doppler Angular Frequency*

Description

This function computes the Bragg Doppler angular frequencies for a SeaSonde radar system. These frequencies represent the characteristic Doppler shifts due to wave resonance at the Bragg wavelength.

Usage

```
seasonder_getBraggDopplerAngularFrequency(seasonder_cs_object)
```

Arguments

`seasonder_cs_object`
: A `SeaSondeRCS` object containing the necessary data to compute the radar wave number.

Details

The Bragg Doppler angular frequency ω_B is calculated using the formula $\omega_B = \sqrt{2 \cdot g \cdot k}$ where:

- g is the gravitational acceleration (approximately $9.8\, m/s^2$),
- k is the radar wave number in radians per meter.

The returned vector contains the negative ($-\omega_B$) and positive ($+\omega_B$) angular frequencies.

Value

A numeric vector of length two, containing the negative and positive Bragg Doppler angular frequencies (in radians per second).

See Also

`seasonder_getRadarWaveNumber` to compute the radar wave number.

`seasonder_getBraggLineBins` *Get Bragg Line Doppler Bins*

Description

This function calculates the Doppler bin indices corresponding to the first-order Bragg frequencies (-1 and 1) for a SeaSonde Cross Spectra (CS) object.

Usage

```
seasonder_getBraggLineBins(seasonder_cs_object)
```

Arguments

`seasonder_cs_object`
: A SeaSonde Cross Spectra (CS) object created by `seasonder_createSeaSondeRCS()`. This object contains the metadata required for the computation, including the normalized Doppler frequencies and their mapping to Doppler bins.

Details

This function uses the normalized Doppler frequencies for the first-order Bragg peaks (-1 and 1) and maps them to their corresponding Doppler bin indices. The mapping is performed using the helper function `seasonder_NormalizedDopplerFreq2Bins()`, which converts normalized frequencies to bin indices based on the spectral resolution and the Doppler range of the radar system.

The bins are critical for identifying the Doppler shifts associated with the first-order Bragg scattering in HF radar systems, which correspond to surface waves with wavelengths half that of the transmitted radar signal.

Value

A numeric vector of length 2, where:

- The first value is the Doppler bin corresponding to the -1 Bragg frequency.
- The second value is the Doppler bin corresponding to the 1 Bragg frequency.

See Also

`seasonder_NormalizedDopplerFreq2Bins` for the frequency-to-bin mapping logic.

seasonder_getBraggWaveLength *Calculate the Bragg Wavelength*

Description

This function computes the Bragg wavelength λ_B for a SeaSonde radar system. The Bragg wavelength is defined as half the radar wavelength and is used to identify the fundamental scattering mechanisms in oceanographic radar measurements.

Usage

```
seasonder_getBraggWaveLength(seasonder_cs_object)
```

Arguments

seasonder_cs_object	A SeaSondeRCS object containing the necessary data to compute the radar wavelength.

Details

The Bragg wavelength λ_B is calculated as: $\lambda_B = \frac{\lambda}{2}$ where:

- λ is the radar wavelength in meters, obtained using seasonder_getRadarWaveLength.

The Bragg wavelength is a critical parameter in interpreting the resonance scattering from the sea surface, which is fundamental to the operation of HF radar systems.

Value

A numeric value representing the Bragg wavelength (in meters).

See Also

seasonder_getRadarWaveLength to compute the radar wavelength.

seasonder_getCenterDopplerBin

Retrieve Center Doppler Bin

Description

This function calculates the center Doppler bin index for a SeaSondeRCS object. It obtains the total number of Doppler cells from the object using `seasonder_getnDopplerCells` and computes the center bin with `seasonder_computeCenterDopplerBin`.

Usage

```
seasonder_getCenterDopplerBin(seasonder_cs_object)
```

Arguments

`seasonder_cs_object`
: A SeaSondeRCS object containing metadata about Doppler bins.

Details

The center Doppler bin is computed by retrieving the total number of Doppler cells (via `seasonder_getnDopplerCells`) and then processing that value with `seasonder_computeCenterDopplerBin`. Note that while CODAR data files might use zero-based indexing, R uses one-based indexing.

Value

A numeric value representing the center Doppler bin.

seasonder_getCenterFreqMHz

Retrieve Center Frequency in MHz

Description

This function extracts the center frequency (in MHz) from the header of a SeaSondeRCS object. It accesses the header field named "CenterFreq" using `seasonder_getSeaSondeRCS_headerField`.

Usage

```
seasonder_getCenterFreqMHz(seasonder_cs_object)
```

Arguments

`seasonder_cs_object`
: A SeaSondeRCS object containing header information.

Value

A numeric value representing the center frequency in MHz.

`seasonder_getCSHeaderByPath` *Retrieve a value from the SeaSondeRCS header by a specific path*

Description

This function retrieves a specific value from the SeaSondeRCS object's header based on the provided path. The path can be a single field name or a list of nested field names.

Usage

```
seasonder_getCSHeaderByPath(seasonder_obj, path, warn_missing = TRUE)
```

Arguments

`seasonder_obj`	A SeaSondeRCS object.
`path`	A character vector specifying the field or nested fields to retrieve.
`warn_missing`	Logical; if `TRUE`, a warning is issued if the specified path is not found in the header.

Value

The value at the specified path in the header. If the path is not found, NULL is returned and a warning is thrown.

Condition Management

This function utilizes the `rlang` package to manage errors and conditions, and provide detailed and structured condition messages:

Condition Classes:

- `seasonder_SeaSonderCS_field_not_found_in_header`: Indicates that the specified path was not found in the header.

Condition Cases:

- Field or nested fields specified by the path are not found in the header.

Examples

```
# Minimal example for seasonder_getCSHeaderByPath
cs_file <- system.file("css_data/CSS_TORA_24_04_04_0700.cs", package = "SeaSondeR")
apm_file <- system.file("css_data/MeasPattern.txt", package = "SeaSondeR")
apm_obj <- seasonder_readSeaSondeRAPMFile(apm_file)
cs_obj <- seasonder_createSeaSondeRCS(cs_file, seasonder_apm_object = apm_obj)
field_value <- seasonder_getCSHeaderByPath(cs_obj, c("nRangeCells"))
print(field_value)
```

seasonder_getDopplerBinsFrequency *Get Doppler Bins Frequency*

Description

This function calculates the frequency limits for each Doppler bin within a SeaSonde Cross Spectrum (CS) object. It can return frequencies either in their original Hz values or normalized by the second Bragg frequency. The frequencies are calculated as the high limit of each Doppler bin interval, similar to what is displayed in SpectraPlotterMap.

Usage

```
seasonder_getDopplerBinsFrequency(seasonder_cs_object, normalized = FALSE)
```

Arguments

seasonder_cs_object	A SeaSonde Cross Spectrum (CS) object created by `seasonder_createSeaSondeRCS()`. This object contains the necessary metadata and spectral data to compute Doppler bin frequencies.
normalized	A logical value indicating if the returned frequencies should be normalized by the second Bragg frequency. When `TRUE`, frequencies are divided by the second Bragg frequency, returning dimensionless values relative to it. Default is `FALSE`, returning frequencies in Hz.

Details

The function internally utilizes several helper functions such as `seasonder_getCenterDopplerBin()`, `seasonder_getnDopplerCells()`, and `seasonder_getDopplerSpectrumResolution()` to calculate the Doppler bin frequencies. Furthermore, when normalization is requested, it uses `seasonder_getBraggDopplerAngu` to obtain the second Bragg frequency for normalization purposes.

Value

A numeric vector of frequencies representing the high limit of each Doppler bin interval. If `normalized` is TRUE, these frequencies are dimensionless values relative to the second Bragg frequency; otherwise, they are in Hz.

`seasonder_getDopplerSpectrumResolution` *Calculate the Doppler Spectrum Resolution*

Description

This function computes the Doppler spectrum resolution for a given SeaSondeRCS object. The resolution reflects the frequency difference between consecutive Doppler bins in the spectrum.

Usage

```
seasonder_getDopplerSpectrumResolution(seasonder_cs_object)
```

Arguments

`seasonder_cs_object`
: A `SeaSondeRCS` object containing the necessary data and metadata for Doppler spectrum analysis.

Details

The Doppler spectrum resolution is calculated using the formula: $SpectralResolution = SweepRate/NumberOfDopple$ where:

- SweepRate is the frequency repetition rate of the radar, obtained from the field `fRepFreqHz` in the object's header.
- NumberOfDopplerCells is the total number of Doppler bins in the spectrum.

This calculation is fundamental for understanding the frequency spacing between adjacent Doppler bins in the radar spectrum.

Value

A numeric value representing the Doppler spectrum resolution in Hertz (Hz).

See Also

`seasonder_getnDopplerCells` to retrieve the number of Doppler cells. `seasonder_getSeaSondeRCS_headerField` to access specific header fields.

seasonder_getFORParameter

Retrieve a Specific FOR Parameter

Description

This function extracts a specified First Order Region (FOR) parameter from a SeaSondeRCS object.

Usage

```
seasonder_getFORParameter(seasonder_cs_object, FOR_parameter)
```

Arguments

seasonder_cs_object	A SeaSondeRCS object containing FOR parameters.
FOR_parameter	A character string specifying the name of the FOR parameter to retrieve.

Details

The function retrieves the list of FOR parameters using `seasonder_getFOR_parameters()` and extracts the value associated with `FOR_parameter`. If the parameter is not found, an error is logged.

Value

The value of the specified FOR parameter if found; otherwise, an error message is logged.

Examples

```
# Minimal example for seasonder_getFORParameter
cs_file <- system.file("css_data/CSS_TORA_24_04_04_0700.cs", package = "SeaSondeR")
apm_file <- system.file("css_data/MeasPattern.txt", package = "SeaSondeR")
apm_obj <- seasonder_readSeaSondeRAPMFile(apm_file)
cs_obj <- seasonder_createSeaSondeRCS(cs_file, seasonder_apm_object = apm_obj)
nsm_value <- seasonder_getFORParameter(cs_obj, "nsm")
print(nsm_value)
```

seasonder_getFOR_currmax

Retrieve FOR Maximum Radial Velocity Limit (currmax)

Description

This function retrieves the maximum radial velocity ('currmax') parameter from the FOR parameters in a SeaSondeRCS object.

Usage

```
seasonder_getFOR_currmax(seasonder_cs_object)
```

Arguments

seasonder_cs_object	A SeaSondeRCS object containing FOR parameters.

Value

The value of the 'currmax' parameter.

Examples

```
# Minimal example for seasonder_getFOR_currmax
cs_file <- system.file("css_data/CSS_TORA_24_04_04_0700.cs", package = "SeaSondeR")
apm_file <- system.file("css_data/MeasPattern.txt", package = "SeaSondeR")
apm_obj <- seasonder_readSeaSondeRAPMFile(apm_file)
cs_obj <- seasonder_createSeaSondeRCS(cs_file, seasonder_apm_object = apm_obj)
currmax_value <- seasonder_getFOR_currmax(cs_obj)
print(currmax_value)
```

---

seasonder_getFOR_fdown

Retrieve FOR Power Dropoff Threshold (fdown)

Description

This function retrieves the power dropoff threshold ('fdown') for First Order Region detection from a SeaSondeRCS object.

Usage

```
seasonder_getFOR_fdown(seasonder_cs_object)
```

Arguments

seasonder_cs_object	A SeaSondeRCS object containing FOR parameters.

Value

The value of the 'fdown' parameter.

Examples

```
# Minimal example for seasonder_getFOR_fdown
cs_file <- system.file("css_data/CSS_TORA_24_04_04_0700.cs", package = "SeaSondeR")
apm_file <- system.file("css_data/MeasPattern.txt", package = "SeaSondeR")
apm_obj <- seasonder_readSeaSondeRAPMFile(apm_file)
cs_obj <- seasonder_createSeaSondeRCS(cs_file, seasonder_apm_object = apm_obj)
fdown_value <- seasonder_getFOR_fdown(cs_obj)
print(fdown_value)
```

---

seasonder_getFOR_flim *Retrieve FOR Null Limit (flim)*

Description

This function retrieves the null limit ('flim') parameter for FIRST Order Region processing from a SeaSondeRCS object.

Usage

```
seasonder_getFOR_flim(seasonder_cs_object)
```

Arguments

`seasonder_cs_object`
: A SeaSondeRCS object containing FOR parameters.

Value

The value of the 'flim' parameter.

Examples

```
# Minimal example for seasonder_getFOR_flim
cs_file <- system.file("css_data/CSS_TORA_24_04_04_0700.cs", package = "SeaSondeR")
apm_file <- system.file("css_data/MeasPattern.txt", package = "SeaSondeR")
apm_obj <- seasonder_readSeaSondeRAPMFile(apm_file)
cs_obj <- seasonder_createSeaSondeRCS(cs_file, seasonder_apm_object = apm_obj)
flim_value <- seasonder_getFOR_flim(cs_obj)
print(flim_value)
```

seasonder_getFOR_noisefact *Retrieve FOR Noise Factor (noisefact)*

Description

This function retrieves the noise factor ('noisefact') used in FOR processing from a SeaSondeRCS object.

Usage

```
seasonder_getFOR_noisefact(seasonder_cs_object)
```

Arguments

seasonder_cs_object	A SeaSondeRCS object containing FOR parameters.

Value

The value of the 'noisefact' parameter.

Examples

```
# Minimal example for seasonder_getFOR_noisefact
cs_file <- system.file("css_data/CSS_TORA_24_04_04_0700.cs", package = "SeaSondeR")
apm_file <- system.file("css_data/MeasPattern.txt", package = "SeaSondeR")
apm_obj <- seasonder_readSeaSondeRAPMFile(apm_file)
cs_obj <- seasonder_createSeaSondeRCS(cs_file, seasonder_apm_object = apm_obj)
noise_factor <- seasonder_getFOR_noisefact(cs_obj)
print(noise_factor)
```

seasonder_getFOR_nsm *Retrieve FOR Doppler Smoothing Factor (nsm)*

Description

This function retrieves the Doppler smoothing factor ('nsm') from the FOR parameters in a SeaSondeRCS object.

Usage

```
seasonder_getFOR_nsm(seasonder_cs_object)
```

Arguments

seasonder_cs_object	A SeaSondeRCS object containing FOR parameters.

Value

The value of the 'nsm' parameter.

Examples

```
# Minimal example for seasonder_getFOR_nsm
cs_file <- system.file("css_data/CSS_TORA_24_04_04_0700.cs", package = "SeaSondeR")
apm_file <- system.file("css_data/MeasPattern.txt", package = "SeaSondeR")
apm_obj <- seasonder_readSeaSondeRAPMFile(apm_file)
cs_obj <- seasonder_createSeaSondeRCS(cs_file, seasonder_apm_object = apm_obj)
smoothing_factor <- seasonder_getFOR_nsm(cs_obj)
print(smoothing_factor)
```

---

seasonder_getFOR_parameters

Retrieve First Order Region (FOR) Parameters

Description

This function retrieves the First Order Region (FOR) parameters associated with a SeaSondeR cross-spectral object. If no FOR parameters are found in the object's attributes, it initializes them using `seasonder_validateFOR_parameters`.

Usage

```
seasonder_getFOR_parameters(seasonder_cs_object)
```

Arguments

`seasonder_cs_object`
: A `SeaSondeRCS` object containing FOR-related metadata.

Details

The function extracts the FOR parameters stored within the object. If the parameters are missing, the function initializes them using `seasonder_validateFOR_parameters` and assigns default values where necessary.

FOR Parameters:

- **nsm**: Doppler smoothing factor.
- **fdown**: Peak power dropoff threshold.
- **flim**: Null below peak power threshold.
- **noisefact**: Signal-to-noise threshold.
- **currmax**: Maximum velocity allowed.
- **reject_distant_bragg**: Flag to reject distant Bragg signals.
- **reject_noise_ionospheric**: Flag to reject ionospheric noise contamination.

- **reject_noise_ionospheric_threshold**: Threshold (in dB) for rejecting noise-affected Bragg signals.
- **reference_noise_normalized_limits**: Estimated limits for reference noise in normalized Doppler frequency.

Value

A named list containing the validated FOR parameters.

See Also

`seasonder_validateFOR_parameters` for initializing and validating FOR parameters. `seasonder_defaultFOR_parameter` for retrieving default parameter values.

`seasonder_getLog` *Retrieve the Last Logs*

Description

This function fetches the most recent log entries from the global log variable `seasonder_the$log`.

Usage

```
seasonder_getLog(n = 100)
```

Arguments

`n`	An integer specifying the number of recent log entries to retrieve.

Value

A character vector of the `n` most recent log entries from the global log.

Examples

```
head(seasonder_getLog())
```

`seasonder_getMUSICConfig` *Retrieve the MUSIC Configuration from a SeaSondeRCS Object*

Description

This function returns the key configuration parameters for the MUSIC algorithm from a SeaSondeRCS object.

Usage

```
seasonder_getMUSICConfig(seasonder_cs_object)
```

Arguments

`seasonder_cs_object`	A SeaSondeRCS object containing MUSIC data and options.

Details

The configuration is aggregated from the MUSIC_data attribute of the object for easy access.

Value

A list containing:

- `doppler_interpolation`: The Doppler interpolation factor.
- `MUSIC_parameters`: The numeric vector of MUSIC parameters.

Examples

```
# Minimal example for seasonder_getMUSICConfig
cs_file <- system.file("css_data/CSS_TORA_24_04_04_0700.cs", package = "SeaSondeR")
apm_file <- system.file("css_data/MeasPattern.txt", package = "SeaSondeR")
apm_obj <- seasonder_readSeaSondeRAPMFile(apm_file)
cs_obj <- seasonder_createSeaSondeRCS(cs_file, seasonder_apm_object = apm_obj)
config <- seasonder_getMUSICConfig(cs_obj)
print(config)
```

seasonder_getMUSICDopplerInterpolation

Retrieve the Doppler Interpolation Factor from MUSIC Options

Description

This function obtains the Doppler interpolation factor used in the MUSIC algorithm from a SeaSondeRCS object.

Usage

```
seasonder_getMUSICDopplerInterpolation(seasonder_cs_object)
```

Arguments

`seasonder_cs_object`
: A SeaSondeRCS object containing MUSIC data and options.

Details

The function accesses the MUSIC_data attribute under MUSIC_options and retrieves the doppler_interpolation parameter. If absent, it defaults to 1L.

Value

An integer representing the Doppler interpolation factor.

Examples

```
# Assuming cs_object is a valid SeaSondeRCS object.
# Minimal example for seasonder_getMUSICDopplerInterpolation
cs_file <- system.file("css_data/CSS_TORA_24_04_04_0700.cs", package = "SeaSondeR")
apm_file <- system.file("css_data/MeasPattern.txt", package = "SeaSondeR")
apm_obj <- seasonder_readSeaSondeRAPMFile(apm_file)
cs_obj <- seasonder_createSeaSondeRCS(cs_file, seasonder_apm_object = apm_obj)
interp_factor <- seasonder_getMUSICDopplerInterpolation(cs_obj)
print(interp_factor)
```

seasonder_getMUSICDualSolutionsProportion

Retrieve Proportion of Dual Solutions from MUSIC Data

Description

This function extracts the proportion of dual solutions from the MUSIC data in a SeaSondeRCS object.

Usage

```
seasonder_getMUSICDualSolutionsProportion(seasonder_cs_object)
```

Arguments

seasonder_cs_object	A SeaSondeRCS object containing MUSIC data.

Details

The function checks the MUSIC_data attribute for a dual_solutions_proportion value. If not available, it defaults to NA_real_.

Value

A numeric value representing the dual solutions proportion, or NA if not set.

Examples

```
# Minimal example for seasonder_getMUSICDualSolutionsProportion
cs_file <- system.file("css_data/CSS_TORA_24_04_04_0700.cs", package = "SeaSondeR")
apm_file <- system.file("css_data/MeasPattern.txt", package = "SeaSondeR")
apm_obj <- seasonder_readSeaSondeRAPMFile(apm_file)
cs_obj <- seasonder_createSeaSondeRCS(cs_file, seasonder_apm_object = apm_obj)
dual_prop <- seasonder_getMUSICDualSolutionsProportion(cs_obj)
print(dual_prop)
```

---

seasonder_getMUSICInterpolatedData

Retrieve Interpolated MUSIC Data from a SeaSondeRCS Object

Description

This function extracts the interpolated MUSIC cross-spectra data from a SeaSondeRCS object.

Usage

```
seasonder_getMUSICInterpolatedData(seasonder_cs_object)
```

Arguments

seasonder_cs_object	A SeaSondeRCS object containing interpolated MUSIC data as an attribute.

Details

The function first checks if the interpolated data is set in the MUSIC_data attribute. If absent, it initializes the data with `seasonder_MUSICInitInterpolatedData()`.

Value

A list representing the interpolated cross-spectra data.

Examples

```
# Minimal example for seasonder_getMUSICInterpolatedData
cs_file <- system.file("css_data/CSS_TORA_24_04_04_0700.cs", package = "SeaSondeR")
apm_file <- system.file("css_data/MeasPattern.txt", package = "SeaSondeR")
apm_obj <- seasonder_readSeaSondeRAPMFile(apm_file)
cs_obj <- seasonder_createSeaSondeRCS(cs_file, seasonder_apm_object = apm_obj)
interp_data <- seasonder_getMUSICInterpolatedData(cs_obj)
str(interp_data)
```

---

seasonder_getMUSICInterpolatedDopplerCellsIndex

Retrieve Interpolated Doppler Cells Index from a SeaSondeRCS Object

---

Description

This function extracts the index of interpolated Doppler cells, stored in the MUSIC_data attribute of a SeaSondeRCS object.

Usage

```
seasonder_getMUSICInterpolatedDopplerCellsIndex(seasonder_cs_object)
```

Arguments

seasonder_cs_object	A SeaSondeRCS object containing MUSIC data.

Details

The interpolated doppler cells index is part of the MUSIC_data and is used to identify which Doppler bins were introduced during the interpolation process.

Value

A vector of indices corresponding to the interpolated Doppler cells.

Examples

```
# Minimal example for seasonder_getMUSICInterpolatedDopplerCellsIndex
cs_file <- system.file("css_data/CSS_TORA_24_04_04_0700.cs", package = "SeaSondeR")
apm_file <- system.file("css_data/MeasPattern.txt", package = "SeaSondeR")
apm_obj <- seasonder_readSeaSondeRAPMFile(apm_file)
cs_obj <- seasonder_createSeaSondeRCS(cs_file, seasonder_apm_object = apm_obj)
doppler_index <- seasonder_getMUSICInterpolatedDopplerCellsIndex(cs_obj)
print(doppler_index)
```

seasonder_getMUSICOptions

Retrieve MUSIC Options from a SeaSondeRCS Object

Description

This function extracts the MUSIC options from a SeaSondeRCS object.

Usage

```
seasonder_getMUSICOptions(seasonder_cs_object)
```

Arguments

`seasonder_cs_object`
: A SeaSondeRCS object containing MUSIC data as an attribute.

Details

The function retrieves the MUSIC options from the object's MUSIC_data attribute. In the absence of user-defined options, it returns the default options provided by `seasonder_defaultMUSICOptions()`.

Value

A list of MUSIC options.

Examples

```
# Minimal example for seasonder_getMUSICOptions
cs_file <- system.file("css_data/CSS_TORA_24_04_04_0700.cs", package = "SeaSondeR")
apm_file <- system.file("css_data/MeasPattern.txt", package = "SeaSondeR")
apm_obj <- seasonder_readSeaSondeRAPMFile(apm_file)
cs_obj <- seasonder_createSeaSondeRCS(cs_file, seasonder_apm_object = apm_obj)
opts <- seasonder_getMUSICOptions(cs_obj)
print(opts)
```

seasonder_getnDopplerCells

Get the nDopplerCells value from a SeaSondeRCS object

Description

Get the nDopplerCells value from a SeaSondeRCS object

Usage

```
seasonder_getnDopplerCells(seasonder_obj)
```

Arguments

seasonder_obj	A SeaSondeRCS object.

Value

The nDopplerCells value.

Examples

```
# Minimal example for seasonder_getnDopplerCells
cs_file <- system.file("css_data/CSS_TORA_24_04_04_0700.cs", package = "SeaSondeR")
apm_file <- system.file("css_data/MeasPattern.txt", package = "SeaSondeR")
apm_obj <- seasonder_readSeaSondeRAPMFile(apm_file)
cs_obj <- seasonder_createSeaSondeRCS(cs_file, seasonder_apm_object = apm_obj)
n_doppler_cells <- seasonder_getnDopplerCells(cs_obj)
print(n_doppler_cells)
```

---

seasonder_getnRangeCells

Get the nRangeCells value from a SeaSondeRCS object

Description

Get the nRangeCells value from a SeaSondeRCS object

Usage

```
seasonder_getnRangeCells(seasonder_obj)
```

Arguments

seasonder_obj	A SeaSondeRCS object.

Value

The nRangeCells value.

Examples

```
# Minimal example for seasonder_getnRangeCells
cs_file <- system.file("css_data/CSS_TORA_24_04_04_0700.cs", package = "SeaSondeR")
apm_file <- system.file("css_data/MeasPattern.txt", package = "SeaSondeR")
apm_obj <- seasonder_readSeaSondeRAPMFile(apm_file)
cs_obj <- seasonder_createSeaSondeRCS(cs_file, seasonder_apm_object = apm_obj)
n_range_cells <- seasonder_getnRangeCells(cs_obj)
print(n_range_cells)
```

`seasonder_getRadarWaveLength` *Calculate the Radar Wavelength*

Description

This function computes the radar wavelength based on the center frequency of the SeaSonde radar system. The wavelength is derived using the speed of light and the radar's center frequency.

Usage

```
seasonder_getRadarWaveLength(seasonder_cs_object)
```

Arguments

`seasonder_cs_object`
: A `SeaSondeRCS` object containing metadata about the radar system, including its center frequency.

Details

The radar wavelength λ is calculated using the formula: $\lambda = \frac{c}{f}$ where:

- c is the speed of light (approximately $3 * 10^8$ m/s),
- f is the radar's center frequency in Hz, retrieved from the SeaSondeRCS object.

The center frequency is initially stored in MHz and is converted to Hz by multiplying it by 10^6.

Value

A numeric value representing the radar wavelength in meters (m).

See Also

`seasonder_getCenterFreqMHz` to retrieve the radar's center frequency.

`seasonder_getRadarWaveNumber` *Calculate the Radar Wave Number*

Description

This function computes the radar wave number k for a SeaSonde radar system based on its wavelength. The wave number represents the spatial frequency of the radar wave.

Usage

```
seasonder_getRadarWaveNumber(seasonder_cs_object)
```

Arguments

`seasonder_cs_object`
: A `SeaSondeRCS` object containing the necessary data to compute the radar wavelength.

Details

The radar wave number k is calculated using the formula: $k = \frac{2\pi}{\lambda}$ where:

- λ is the radar wavelength in meters, calculated using `seasonder_getRadarWaveLength`.
- 2π represents the relationship between the wavelength and wave number.

The wave number is an essential parameter for analyzing radar signals and their interaction with the medium being measured.

Value

A numeric value representing the radar wave number k in radians per meter.

See Also

`seasonder_getRadarWaveLength` to compute the radar wavelength.

seasonder_getRadialVelocityResolution
Calculate Radial Velocity Resolution

Description

Computes the radial velocity resolution for a SeaSonde radar cross-section (CS) object. This measurement indicates the smallest change in velocity that the radar can discern between different targets or scatterers within its observation area. The calculation is based on the Doppler spectrum resolution and the radar wave number, providing a crucial parameter for analyzing the radar's capability to distinguish between velocities.

Usage

```
seasonder_getRadialVelocityResolution(seasonder_cs_object)
```

Arguments

`seasonder_cs_object`
: A SeaSondeRCS object created using `seasonder_createSeaSondeRCS`. This object contains the necessary data to calculate the Doppler spectrum resolution and, subsequently, the radial velocity resolution.

Details

The radial velocity resolution v_{res} is determined using the formula:

$$v_{res} = \frac{\text{SpectraRes}}{2 \cdot k_0}$$

where v_{res} is the radial velocity resolution, SpectraRes is the Doppler spectrum resolution, and k_0 is the radar wave number divided by 2π. This formula reflects the relationship between the frequency resolution of the radar's Doppler spectrum and the corresponding velocity resolution, taking into account the wave number which is a fundamental characteristic of the radar system.

Value

A single numeric value representing the radial velocity resolution in meters per second (m/s), indicating the radar's ability to differentiate between closely spaced velocities.

See Also

`seasonder_getDopplerSpectrumResolution`, `seasonder_getRadarWaveNumber`

seasonder_getReceiverGain_dB *Retrieve Receiver Gain in Decibels*

Description

This function retrieves the receiver gain value (in decibels) from the header of a given SeaSondeRCS object. If the receiver gain field is missing or NULL, a default value of -34.2 dB is returned.

Usage

```
seasonder_getReceiverGain_dB(seasonder_cs_object)
```

Arguments

seasonder_cs_object	A SeaSondeRCS object containing header information about the radar system.

Details

The function extracts the value of the header field fReferenceGainDB using seasonder_getSeaSondeRCS_headerField. If the field is not present or has a NULL value, the function defaults to a receiver gain of -34.2 dB (CODAR, 2016).

Value

A numeric value representing the receiver gain in decibels (dB).

References

Cross Spectra File Format Version 6, CODAR. (2016).

See Also

seasonder_getSeaSondeRCS_headerField to retrieve specific fields from the SeaSondeRCS header.

seasonder_getSeaSondeRAPM_AmplitudeFactors *Getter for AmplitudeFactors*

Description

Getter for AmplitudeFactors

Usage

```
seasonder_getSeaSondeRAPM_AmplitudeFactors(seasonde_apm_obj)
```

Arguments

seasonde_apm_obj
: SeaSonderAPM object

Value

The AmplitudeFactors attribute from the object.

Examples

```
# Minimal example for seasonder_getSeaSondeRAPM_AmplitudeFactors
apm_file <- system.file("css_data/MeasPattern.txt", package = "SeaSondeR")
apm_obj <- seasonder_readSeaSondeRAPMFile(apm_file)
amplitude_factors <- seasonder_getSeaSondeRAPM_AmplitudeFactors(apm_obj)
print(amplitude_factors)
```

---

seasonder_getSeaSondeRAPM_AntennaBearing
Getter for AntennaBearing

Description

Getter for AntennaBearing

Usage

```
seasonder_getSeaSondeRAPM_AntennaBearing(seasonde_apm_obj)
```

Arguments

seasonde_apm_obj
: SeaSonderAPM object

Value

The AntennaBearing attribute from the object.

Examples

```
# Minimal example for seasonder_getSeaSondeRAPM_AntennaBearing
apm_file <- system.file("css_data/MeasPattern.txt", package = "SeaSondeR")
apm_obj <- seasonder_readSeaSondeRAPMFile(apm_file)
antenna_bearing <- seasonder_getSeaSondeRAPM_AntennaBearing(apm_obj)
print(antenna_bearing)
```

seasonder_getSeaSondeRAPM_BEAR *Getter for BEAR*

Description

Getter for BEAR

Usage

```
seasonder_getSeaSondeRAPM_BEAR(seasonde_apm_obj)
```

Arguments

seasonde_apm_obj	SeaSonderAPM object

Value

The BEAR attribute (bearing values) from the object.

Examples

```
# Minimal example for seasonder_getSeaSondeRAPM_BEAR
apm_file <- system.file("css_data/MeasPattern.txt", package = "SeaSondeR")
apm_obj <- seasonder_readSeaSondeRAPMFile(apm_file)
bear <- seasonder_getSeaSondeRAPM_BEAR(apm_obj)
print(bear)
```

seasonder_getSeaSondeRAPM_BearingResolution *Getter for BearingResolution*

Description

Getter for BearingResolution

Usage

```
seasonder_getSeaSondeRAPM_BearingResolution(seasonde_apm_obj)
```

Arguments

seasonde_apm_obj	SeaSonderAPM object

Value

The BearingResolution attribute from the object.

Examples

```
# Minimal example for seasonder_getSeaSondeRAPM_BearingResolution
apm_file <- system.file("css_data/MeasPattern.txt", package = "SeaSondeR")
apm_obj <- seasonder_readSeaSondeRAPMFile(apm_file)
bearing_resolution <- seasonder_getSeaSondeRAPM_BearingResolution(apm_obj)
print(bearing_resolution)
```

---

seasonder_getSeaSondeRAPM_CommentLine

Getter for CommentLine

Description

Getter for CommentLine

Usage

```
seasonder_getSeaSondeRAPM_CommentLine(seasonde_apm_obj)
```

Arguments

seasonde_apm_obj	SeaSonderAPM object

Value

The CommentLine attribute from the object.

Examples

```
# Minimal example for seasonder_getSeaSondeRAPM_CommentLine
apm_file <- system.file("css_data/MeasPattern.txt", package = "SeaSondeR")
apm_obj <- seasonder_readSeaSondeRAPMFile(apm_file)
comment_line <- seasonder_getSeaSondeRAPM_CommentLine(apm_obj)
print(comment_line)
```

seasonder_getSeaSondeRAPM_CreateTimeStamp *Getter for CreateTimeStamp*

Description

Getter for CreateTimeStamp

Usage

```
seasonder_getSeaSondeRAPM_CreateTimeStamp(seasonde_apm_obj)
```

Arguments

seasonde_apm_obj	SeaSonderAPM object

Value

The CreateTimeStamp attribute from the object.

Examples

```
# Minimal example for seasonder_getSeaSondeRAPM_CreateTimeStamp
apm_file <- system.file("css_data/MeasPattern.txt", package = "SeaSondeR")
apm_obj <- seasonder_readSeaSondeRAPMFile(apm_file)
create_time_stamp <- seasonder_getSeaSondeRAPM_CreateTimeStamp(apm_obj)
print(create_time_stamp)
```

seasonder_getSeaSondeRAPM_Creator *Getter for Creator*

Description

Getter for Creator

Usage

```
seasonder_getSeaSondeRAPM_Creator(seasonde_apm_obj)
```

Arguments

seasonde_apm_obj	SeaSonderAPM object

Value

The Creator attribute from the object.

Examples

```
# Create a default SeaSondeRAPM object
obj <- seasonder_createSeaSondeRAPM()
creator <- seasonder_getSeaSondeRAPM_Creator(obj)
```

---

seasonder_getSeaSondeRAPM_FileID

Getter for FileID

Description

Getter for FileID

Usage

```
seasonder_getSeaSondeRAPM_FileID(seasonde_apm_obj)
```

Arguments

seasonde_apm_obj	SeaSonderAPM object

Value

The FileID attribute from the object.

Examples

```
# Minimal example for seasonder_getSeaSondeRAPM_FileID
apm_file <- system.file("css_data/MeasPattern.txt", package = "SeaSondeR")
apm_obj <- seasonder_readSeaSondeRAPMFile(apm_file)
file_id <- seasonder_getSeaSondeRAPM_FileID(apm_obj)
print(file_id)
```

seasonder_getSeaSondeRAPM_FileName *Getter for FileName*

Description

Getter for FileName

Usage

```
seasonder_getSeaSondeRAPM_FileName(seasonde_apm_obj)
```

Arguments

seasonde_apm_obj	SeaSonderAPM object

Value

The FileName attribute from the object.

Examples

```
# Minimal example for seasonder_getSeaSondeRAPM_FileName
apm_file <- system.file("css_data/MeasPattern.txt", package = "SeaSondeR")
apm_obj <- seasonder_readSeaSondeRAPMFile(apm_file)
file_name <- seasonder_getSeaSondeRAPM_FileName(apm_obj)
print(file_name)
```

seasonder_getSeaSondeRAPM_PhaseCorrections *Getter for PhaseCorrections*

Description

Getter for PhaseCorrections

Usage

```
seasonder_getSeaSondeRAPM_PhaseCorrections(seasonde_apm_obj)
```

Arguments

seasonde_apm_obj	SeaSonderAPM object

Value

The PhaseCorrections attribute from the object.

Examples

```
# Create a default SeaSondeRAPM object
obj <- seasonder_createSeaSondeRAPM()
phase_corrections <- seasonder_getSeaSondeRAPM_PhaseCorrections(obj)
```

---

seasonder_getSeaSondeRAPM_ProcessingSteps

Getter for ProcessingSteps

Description

Getter for ProcessingSteps

Usage

```
seasonder_getSeaSondeRAPM_ProcessingSteps(seasonde_apm_obj)
```

Arguments

seasonde_apm_obj	SeaSonderAPM object

Value

The ProcessingSteps attribute from the object.

Examples

```
# Minimal example for seasonder_getSeaSondeRAPM_ProcessingSteps
apm_file <- system.file("css_data/MeasPattern.txt", package = "SeaSondeR")
apm_obj <- seasonder_readSeaSondeRAPMFile(apm_file)
processing_steps <- seasonder_getSeaSondeRAPM_ProcessingSteps(apm_obj)
print(processing_steps)
```

seasonder_getSeaSondeRAPM_quality_matrix *Getter for quality_matrix*

Description

Getter for quality_matrix

Usage

```
seasonder_getSeaSondeRAPM_quality_matrix(seasonde_apm_obj)
```

Arguments

seasonde_apm_obj	SeaSonderAPM object

Value

The quality_matrix attribute from the object.

Examples

```
# Create a default SeaSondeRAPM object
obj <- seasonder_createSeaSondeRAPM()
quality_matrix <- seasonder_getSeaSondeRAPM_quality_matrix(obj)
```

seasonder_getSeaSondeRAPM_SiteName *Getter for SiteName*

Description

Getter for SiteName

Usage

```
seasonder_getSeaSondeRAPM_SiteName(seasonde_apm_obj)
```

Arguments

seasonde_apm_obj	SeaSonderAPM object

Value

The SiteName attribute from the object.

Examples

```
# Minimal example for seasonder_getSeaSondeRAPM_SiteName
apm_file <- system.file("css_data/MeasPattern.txt", package = "SeaSondeR")
apm_obj <- seasonder_readSeaSondeRAPMFile(apm_file)
site_name <- seasonder_getSeaSondeRAPM_SiteName(apm_obj)
print(site_name)
```

---

seasonder_getSeaSondeRAPM_SiteOrigin *Getter for SiteOrigin*

Description

Getter for SiteOrigin

Usage

```
seasonder_getSeaSondeRAPM_SiteOrigin(seasonde_apm_obj)
```

Arguments

seasonde_apm_obj	SeaSonderAPM object

Value

The SiteOrigin attribute from the object.

Examples

```
# Minimal example for seasonder_getSeaSondeRAPM_SiteOrigin
apm_file <- system.file("css_data/MeasPattern.txt", package = "SeaSondeR")
apm_obj <- seasonder_readSeaSondeRAPMFile(apm_file)
site_origin <- seasonder_getSeaSondeRAPM_SiteOrigin(apm_obj)
print(site_origin)
```

---

seasonder_getSeaSondeRAPM_Smoothing *Getter for Smoothing*

Description

Getter for Smoothing

Usage

```
seasonder_getSeaSondeRAPM_Smoothing(seasonde_apm_obj)
```

Arguments

seasonde_apm_obj
: SeaSonderAPM object

Value

The Smoothing attribute from the object.

Examples

```
# Minimal example for seasonder_getSeaSondeRAPM_Smoothing
apm_file <- system.file("css_data/MeasPattern.txt", package = "SeaSondeR")
apm_obj <- seasonder_readSeaSondeRAPMFile(apm_file)
smoothing <- seasonder_getSeaSondeRAPM_Smoothing(apm_obj)
print(smoothing)
```

---

seasonder_getSeaSondeRAPM_StationCode

Getter for StationCode

Description

Getter for StationCode

Usage

```
seasonder_getSeaSondeRAPM_StationCode(seasonde_apm_obj)
```

Arguments

seasonde_apm_obj
: SeaSonderAPM object

Value

The StationCode attribute from the object.

Examples

```
# Minimal example for seasonder_getSeaSondeRAPM_StationCode
apm_file <- system.file("css_data/MeasPattern.txt", package = "SeaSondeR")
apm_obj <- seasonder_readSeaSondeRAPMFile(apm_file)
station_code <- seasonder_getSeaSondeRAPM_StationCode(apm_obj)
print(station_code)
```

seasonder_getSeaSondeRAPM_Type

Getter for Type

Description

Getter for Type

Usage

```
seasonder_getSeaSondeRAPM_Type(seasonde_apm_obj)
```

Arguments

seasonde_apm_obj	SeaSonderAPM object

Value

The Type attribute from the object.

Examples

```
# Create a default SeaSondeRAPM object
obj <- seasonder_createSeaSondeRAPM()
type <- seasonder_getSeaSondeRAPM_Type(obj)
```

seasonder_getSeaSondeRCS_antenna_SSdata

Retrieve Self-Spectra Data for a Specific Antenna from a SeaSondeRCS Object

Description

This function extracts the self-spectra (SSA) data matrix for a given antenna from a SeaSondeRCS object.

Usage

```
seasonder_getSeaSondeRCS_antenna_SSdata(seasonder_cs_object, antenna)
```

Arguments

seasonder_cs_object	A SeaSondeRCS object containing spectral data.
antenna	An integer specifying the antenna number (1, 2, or 3).

Details

The function constructs the matrix name dynamically by appending the antenna number to the prefix `"SSA"` (e.g., `"SSA1"`, `"SSA2"`, or `"SSA3"`). It then retrieves the corresponding matrix from the SeaSondeRCS data using `seasonder_getSeaSondeRCS_dataMatrix`.

Value

A matrix containing the self-spectra data for the specified antenna. If the antenna number is invalid, an error is thrown.

See Also

`seasonder_getSeaSondeRCS_dataMatrix` for extracting specific data matrices. `seasonder_getSeaSondeRCS_data` for retrieving the complete data structure.

---

seasonder_getSeaSondeRCS_APM

Retrieve the APM Attribute from a SeaSondeRCS Object

Description

This function extracts the APM (Antenna Pattern Matrix or similar metadata) attribute from a SeaSondeRCS object. This attribute is stored as an attribute named "APM" within the object.

Usage

```
seasonder_getSeaSondeRCS_APM(seasonder_cs_object)
```

Arguments

`seasonder_cs_object`
: A SeaSondeRCS object.

Details

The function uses `attr(..., exact = TRUE)` to ensure that the correct attribute is retrieved.

Value

The value of the "APM" attribute from the SeaSondeRCS object.

Examples

```
# Create a minimal SeaSondeRCS object
cs_obj <- structure(list(data = list(a = 1, b = 2)), class = "SeaSondeRCS")
apm_value <- seasonder_getSeaSondeRCS_APM(cs_obj)
print(apm_value)
```

seasonder_getSeaSondeRCS_data *Getter for data*

Description

Getter for data

Usage

```
seasonder_getSeaSondeRCS_data(seasonder_cs_object)
```

Arguments

seasonder_cs_object	SeaSondeRCS object

Value

A list containing the data matrices for the SeaSondeRCS object. If the data is not set, it initializes the data structure with the number of range and Doppler cells.

See Also

seasonder_getnRangeCells seasonder_getnDopplerCells seasonder_initCSDataStructure

Examples

```
# Create a minimal SeaSondeRCS object
cs_obj <- structure(list(data = list(a = 1, b = 2)), class = "SeaSondeRCS")
data_list <- seasonder_getSeaSondeRCS_data(cs_obj)
print(data_list)
```

seasonder_getSeaSondeRCS_dataMatrix *Retrieve a Specific Data Matrix from a SeaSondeRCS Object*

Description

This function extracts a specific data matrix from a SeaSondeRCS object. The available matrices correspond to self-spectra and cross-spectra components used in SeaSonde radar processing.

Usage

```
seasonder_getSeaSondeRCS_dataMatrix(seasonder_cs_object, matrix_name)
```

Arguments

`seasonder_cs_object`
: A SeaSondeRCS object containing the spectral data.

`matrix_name`
: A string specifying the name of the matrix to retrieve. Must be one of:
  - `"SSA1"`: Self-spectra for antenna 1.
  - `"SSA2"`: Self-spectra for antenna 2.
  - `"SSA3"`: Self-spectra for antenna 3.
  - `"CS12"`: Cross-spectra between antennas 1 and 2.
  - `"CS13"`: Cross-spectra between antennas 1 and 3.
  - `"CS23"`: Cross-spectra between antennas 2 and 3.
  - `"QC"`: Quality control matrix.

Details

The function first verifies that the provided `matrix_name` is valid. If the name is not in the list of accepted values, it logs an error and aborts execution using `seasonder_logAndAbort`. Once validated, the function extracts the requested matrix from the `data` component of the SeaSondeRCS object.

Value

A matrix containing the requested spectral data. If the matrix name is invalid, an error is thrown.

See Also

`seasonder_getSeaSondeRCS_data` for retrieving the complete data structure. `seasonder_logAndAbort` for error handling.

`seasonder_getSeaSondeRCS_FOR`

Retrieve First Order Region (FOR) Data from SeaSondeRCS Object

---

Description

This function extracts the First Order Region (FOR) data from a SeaSondeRCS object. If the FOR data is not found in the object's attributes, it is initialized using `seasonder_initSeaSondeRCS_FOR()`.

Usage

```
seasonder_getSeaSondeRCS_FOR(seasonder_cs_object)
```

Arguments

`seasonder_cs_object`
: A SeaSondeRCS object containing FOR-related data.

Details

The function attempts to retrieve the 'FOR' element from the object's "FOR_data" attribute. If it does not exist, it calls `seasonder_initSeaSondeRCS_FOR()` to initialize the FOR data.

Value

The FOR data structure.

Examples

```
# Minimal example for seasonder_getSeaSondeRCS_FOR
cs_file <- system.file("css_data/CSS_TORA_24_04_04_0700.cs", package = "SeaSondeR")
apm_file <- system.file("css_data/MeasPattern.txt", package = "SeaSondeR")
apm_obj <- seasonder_readSeaSondeRAPMFile(apm_file)
cs_obj <- seasonder_createSeaSondeRCS(cs_file, seasonder_apm_object = apm_obj)
FOR_data <- seasonder_getSeaSondeRCS_FOR(cs_obj)
head(FOR_data)
```

---

seasonder_getSeaSondeRCS_FORConfig

Retrieve First Order Region (FOR) Configuration from a SeaSondeRCS Object

---

Description

This function extracts the configuration related to the First Order Region (FOR) from a SeaSondeRCS object. It returns a list containing the FOR parameters and the noise level assigned to the object.

Usage

```
seasonder_getSeaSondeRCS_FORConfig(seasonder_cs_object)
```

Arguments

`seasonder_cs_object`
: A SeaSondeRCS object containing FOR-related metadata.

Details

The FOR configuration is composed of parameters that define the first order region and the noise level used during FOR processing. This function aggregates these components by calling `seasonder_getFOR_parameters()` and `seasonder_getSeaSondeRCS_NoiseLevel()`.

Value

A list with two components:

- `FOR_parameters`: A list of parameters used for FOR processing.
- `NoiseLevel`: The noise level values retrieved from the object.

Examples

```
# Create a minimal SeaSondeRCS object
cs_file <- system.file("css_data/CSS_TORA_24_04_04_0700.cs", package = "SeaSondeR")
apm_file <- system.file("css_data/MeasPattern.txt", package = "SeaSondeR")
apm_obj <- seasonder_readSeaSondeRAPMFile(apm_file)
cs_obj <- seasonder_createSeaSondeRCS(cs_file, seasonder_apm_object = apm_obj)
config <- seasonder_getSeaSondeRCS_FORConfig(cs_obj)
print(config)
```

---

seasonder_getSeaSondeRCS_FOR_SS_Smoothed

Retrieve Smoothed Self-Spectra for First Order Region (FOR)

Description

This function retrieves the smoothed self-spectra (SS) matrix stored in the `FOR_data` attribute of a `SeaSondeRCS` object. The smoothed self-spectra are used in First Order Region (FOR) processing to refine the detection of the first-order boundaries.

Usage

```
seasonder_getSeaSondeRCS_FOR_SS_Smoothed(seasonder_cs_object)
```

Arguments

`seasonder_cs_object`
: A `SeaSondeRCS` object containing smoothed self-spectra data.

Details

The function extracts the matrix assigned by `seasonder_setSeaSondeRCS_FOR_SS_Smoothed`. If no smoothed self-spectra are found, the function returns `NULL`.

The smoothed self-spectra are typically generated using `seasonder_SmoothSS` and applied to the self-spectra of antenna 3. This smoothing aids in detecting the nulls that separate first- and second-order regions.

Value

A matrix representing the smoothed self-spectra, or `NULL` if no smoothed data is stored.

See Also

- `seasonder_SmoothFORSS` for applying smoothing and storing the result.
- `seasonder_setSeaSondeRCS_FOR_SS_Smoothed` for setting smoothed self-spectra.

seasonder_getSeaSondeRCS_header *Getter for header*

Description

Getter for header

Usage

```
seasonder_getSeaSondeRCS_header(seasonder_cs_object)
```

Arguments

seasonder_cs_object	SeaSondeRCS object

Value

A list containing the header data of the SeaSondeRCS object.

Examples

```
# Create a minimal SeaSondeRCS object with a header attribute
cs_obj <- structure(list(data = list(a = 1, b = 2)), class = "SeaSondeRCS")
attr(cs_obj, "header") <- list(
  nSiteCodeName = "Station1",
  nDateTime = Sys.time(),
  nDopplerCells = 2,
  nRangeCells = 3
)
header_data <- seasonder_getSeaSondeRCS_header(cs_obj)
print(header_data)
```

seasonder_getSeaSondeRCS_headerField *Retrieve a Specific Field from a SeaSondeRCS Header*

Description

This function extracts a specific field from the header of a SeaSondeRCS object.

Usage

```
seasonder_getSeaSondeRCS_headerField(seasonder_cs_object, field)
```

Arguments

`seasonder_cs_object`
: A SeaSondeRCS object.

`field`
: A string specifying the field name to retrieve from the header.

Details

This function first retrieves the full header using `seasonder_getSeaSondeRCS_header` and then attempts to extract the requested field using `pluck`. The header is flattened before extraction to accommodate nested structures.

Value

The value of the specified field from the header. If the field is not found, NULL is returned.

See Also

`seasonder_getSeaSondeRCS_header` for retrieving the full header. `pluck` for selective element extraction.

`seasonder_getSeaSondeRCS_MUSIC` *Retrieve MUSIC Data from a SeaSondeRCS Object*

Description

This function extracts the MUSIC data structure from a SeaSondeRCS object.

Usage

```
seasonder_getSeaSondeRCS_MUSIC(seasonder_cs_object)
```

Arguments

`seasonder_cs_object`
: A SeaSondeRCS object containing MUSIC data as an attribute.

Details

If the MUSIC data does not exist in the object, the function initializes it via `seasonder_initSeaSondeRCS_MUSIC()`.

Value

The MUSIC data structure, typically a data frame or tibble with MUSIC results.

Examples

```
# Minimal example for seasonder_getMUSIC
cs_file <- system.file("css_data/CSS_TORA_24_04_04_0700.cs", package = "SeaSondeR")
apm_file <- system.file("css_data/MeasPattern.txt", package = "SeaSondeR")
apm_obj <- seasonder_readSeaSondeRAPMFile(apm_file)
cs_obj <- seasonder_createSeaSondeRCS(cs_file, seasonder_apm_object = apm_obj)
music_data <- seasonder_getSeaSondeRCS_MUSIC(cs_obj)
print(music_data)
```

`seasonder_getSeaSondeRCS_MUSIC_BinsRadialVelocity`
Retrieve Radial Velocities for MUSIC Doppler Bins

Description

This function calculates the radial velocities for MUSIC Doppler bins based on the given SeaSonde cross-spectral object.

Usage

```
seasonder_getSeaSondeRCS_MUSIC_BinsRadialVelocity(seasonder_cs_object)
```

Arguments

`seasonder_cs_object`
: A SeaSondeRCS object representing the cross-spectral data structure. It contains necessary metadata and Doppler frequency information.

Details

The function uses the following process:

- It retrieves the Doppler bin frequencies using `seasonder_getSeaSondeRCS_MUSIC_DopplerBinsFrequency`.
- It computes the radial velocities associated with the bins using `seasonder_computeBinsRadialVelocity`.

The computed velocities are returned as a numeric vector, which can be used in subsequent analyses or visualizations.

Value

A numeric vector containing the radial velocities corresponding to each MUSIC Doppler bin.

See Also

`seasonder_getSeaSondeRCS_MUSIC_DopplerBinsFrequency`, `seasonder_computeBinsRadialVelocity`

`seasonder_getSeaSondeRCS_MUSIC_CenterDopplerBin` *Retrieve the Center Doppler Bin for MUSIC Analysis*

Description

This function calculates the center Doppler bin for a `SeaSondeRCS` object. The center bin corresponds to the Doppler bin representing zero frequency, and the computation accounts for adjustments from the MUSIC Doppler interpolation.

Usage

```
seasonder_getSeaSondeRCS_MUSIC_CenterDopplerBin(seasonder_cs_object)
```

Arguments

`seasonder_cs_object`
: A `SeaSondeRCS` object containing data and parameters for MUSIC analysis.

Details

The function performs the following steps:

1. Retrieves the total number of Doppler cells, including adjustments for MUSIC interpolation, using `seasonder_getSeaSondeRCS_MUSIC_nDopplerCells`.
2. Computes the center Doppler bin using `seasonder_computeCenterDopplerBin`, which determines the bin corresponding to zero frequency.

The center Doppler bin is a key parameter for organizing Doppler frequency data around zero and is critical for spectral analysis.

Value

An integer representing the center Doppler bin.

See Also

`seasonder_getSeaSondeRCS_MUSIC_nDopplerCells` to retrieve the adjusted number of Doppler cells. `seasonder_computeCenterDopplerBin` for the center bin calculation.

`seasonder_getSeaSondeRCS_MUSIC_DopplerBinsFrequency` *Calculate Doppler Bins Frequencies for MUSIC Analysis*

Description

This function computes the Doppler bin frequencies for a given `SeaSondeRCS` object, incorporating adjustments from the MUSIC analysis. The computation accounts for Doppler interpolation and the spectrum resolution.

Usage

```
seasonder_getSeaSondeRCS_MUSIC_DopplerBinsFrequency(
  seasonder_cs_object,
  normalized = FALSE
)
```

Arguments

`seasonder_cs_object`	A `SeaSondeRCS` object containing the data and parameters for MUSIC analysis.
`normalized`	Logical. If `TRUE`, the returned frequencies are normalized by the positive Bragg frequency. Default is `FALSE`, returning frequencies in Hz.

Details

The function performs the following steps:

1. Retrieves the central Doppler bin corresponding to 0 frequency using `seasonder_getSeaSondeRCS_MUSIC_CenterDop`
2. Retrieves the total number of Doppler cells (adjusted for interpolation) using `seasonder_getSeaSondeRCS_MUSIC_nDo`
3. Retrieves the Doppler spectrum resolution using `seasonder_getSeaSondeRCS_MUSIC_DopplerSpectrumResolution`.
4. Computes the Doppler bin frequencies using `seasonder_computeDopplerBinsFrequency`.

The resulting Doppler bins frequencies are crucial for analyzing the spectral properties of the MUSIC output.

Value

A numeric vector representing the frequency values for each Doppler bin. If `normalized = TRUE`, these values are dimensionless, relative to the positive Bragg frequency. Otherwise, they are in Hz.

See Also

`seasonder_getSeaSondeRCS_MUSIC_CenterDopplerBin` to retrieve the central bin. `seasonder_getSeaSondeRCS_MUSIC_` for the number of Doppler cells. `seasonder_getSeaSondeRCS_MUSIC_DopplerSpectrumResolution` for the adjusted spectrum resolution. `seasonder_computeDopplerBinsFrequency` for the frequency calculation.

`seasonder_getSeaSondeRCS_MUSIC_DopplerSpectrumResolution` *Retrieve the Adjusted Doppler Spectrum Resolution for MUSIC Analysis*

Description

This function calculates the Doppler spectrum resolution adjusted by the Doppler interpolation factor for a given `SeaSondeRCS` object. The adjustment ensures that the spectrum resolution reflects the impact of interpolation applied in the MUSIC analysis.

Usage

```
seasonder_getSeaSondeRCS_MUSIC_DopplerSpectrumResolution(seasonder_cs_object)
```

Arguments

`seasonder_cs_object` A `SeaSondeRCS` object containing the data and parameters for MUSIC analysis.

Details

The function performs the following steps:

1. Retrieves the base Doppler spectrum resolution using `seasonder_getDopplerSpectrumResolution`.
2. Obtains the Doppler interpolation factor using `seasonder_getSeaSondeRCS_MUSIC_doppler_interpolation`.
3. Divides the base resolution by the interpolation factor to compute the adjusted resolution.

This adjustment is critical for accurately interpreting MUSIC data in cases where Doppler interpolation has been applied.

Value

A numeric value representing the adjusted Doppler spectrum resolution.

See Also

`seasonder_getDopplerSpectrumResolution` to retrieve the base Doppler spectrum resolution. `seasonder_getSeaSondeRCS_MUSIC_doppler_interpolation` to retrieve the Doppler interpolation factor.

`seasonder_getSeaSondeRCS_MUSIC_nDopplerCells` *Retrieve the Interpolated Number of Doppler Cells for MUSIC*

Description

This function calculates the interpolated number of Doppler cells for the MUSIC data in a given `SeaSondeRCS` object. It applies a Doppler interpolation factor to the original number of Doppler cells.

Usage

```
seasonder_getSeaSondeRCS_MUSIC_nDopplerCells(seasonder_cs_object)
```

Arguments

`seasonder_cs_object`
: A `SeaSondeRCS` object containing metadata and configurations related to MUSIC data processing.

Details

The function performs the following steps:

1. Retrieves the total number of Doppler cells using `seasonder_getnDopplerCells`.
2. Retrieves the Doppler interpolation factor using `seasonder_getSeaSondeRCS_MUSIC_doppler_interpolation`.
3. Multiplies the number of Doppler cells by the interpolation factor to compute the interpolated number of Doppler cells.

Value

An integer representing the number of Doppler cells adjusted by the Doppler interpolation factor.

See Also

`seasonder_getnDopplerCells` to obtain the base number of Doppler cells. `seasonder_getSeaSondeRCS_MUSIC_doppler_` to retrieve the Doppler interpolation factor.

seasonder_getSeaSondeRCS_MUSIC_parameters
Retrieve MUSIC Parameters from a SeaSondeRCS Object

Description

This function extracts the MUSIC algorithm parameters from a SeaSondeRCS object.

Usage

```
seasonder_getSeaSondeRCS_MUSIC_parameters(seasonder_cs_object)
```

Arguments

`seasonder_cs_object`
: A SeaSondeRCS object containing MUSIC data as an attribute.

Details

The function checks for the presence of MUSIC parameters in the object's MUSIC_data attribute. If not found, it defaults to the values returned by `seasonder_defaultMUSIC_parameters()`.

Value

A numeric vector of MUSIC parameters.

Examples

```
# Minimal example for seasonder_getSeaSondeRCS_MUSIC_parameters
cs_file <- system.file("css_data/CSS_TORA_24_04_04_0700.cs", package = "SeaSondeR")
apm_file <- system.file("css_data/MeasPattern.txt", package = "SeaSondeR")
apm_obj <- seasonder_readSeaSondeRAPMFile(apm_file)
cs_obj <- seasonder_createSeaSondeRCS(cs_file, seasonder_apm_object = apm_obj)
params <- seasonder_getSeaSondeRCS_MUSIC_parameters(cs_obj)
print(params)
```

seasonder_getSeaSondeRCS_ProcessingSteps
Getter for ProcessingSteps

Description

Getter for ProcessingSteps

Usage

```
seasonder_getSeaSondeRCS_ProcessingSteps(seasonder_cs_object)
```

Arguments

seasonder_cs_object
: SeaSonderCS object

Value

A list containing the processing steps of the SeaSondeRCS object.

Examples

```
# Create a SeaSondeRCS object for examples
cs_file <- system.file("css_data/CSS_TORA_24_04_04_0700.cs", package = "SeaSondeR")
apm_file <- system.file("css_data/MeasPattern.txt", package = "SeaSondeR")
apm_obj <- seasonder_readSeaSondeRAPMFile(apm_file)
cs_obj <- seasonder_createSeaSondeRCS(cs_file, seasonder_apm_object = apm_obj)
# Retrieve processing steps
processing_steps <- seasonder_getSeaSondeRCS_ProcessingSteps(cs_obj)
print(processing_steps)
```

---

seasonder_getSeaSondeRCS_reference_noise_normalized_limits_estimation_interval
Retrieve the Reference Noise Normalized Limits Estimation Interval

---

Description

This function extracts the reference noise normalized limits estimation interval from a SeaSondeRCS object's attributes. These limits are stored under the attribute name "reference_noise_normalized_limits_estima

Usage

```
seasonder_getSeaSondeRCS_reference_noise_normalized_limits_estimation_interval(
  seasonder_cs_object
)
```

Arguments

seasonder_cs_object
: A SeaSondeRCS object.

Details

This interval is typically used during the noise level estimation process for the SeaSondeRCS object.

Value

The reference noise normalized limits estimation interval as stored in the object.

Examples

```
# Create a minimal SeaSondeRCS object
cs_obj <- structure(list(data = list(a = 1, b = 2)), class = "SeaSondeRCS")
interval <- seasonder_getSeaSondeRCS_reference_noise_normalized_limits_estimation_interval(cs_obj)
print(interval)
```

---

seasonder_getSeaSondeRCS_SelfSpectra

Retrieve Self-Spectra Power Matrices for Specified Antenna, Range, and Doppler Intervals

---

Description

This function returns a list of power spectra extracted from a SeaSondeRCS object for each combination of the specified antennae, range intervals, and Doppler intervals. It allows users to focus on subregions of the self-spectra data. Additionally, the resulting nested list can be collapsed into a single-level list.

Usage

```
seasonder_getSeaSondeRCS_SelfSpectra(
  seasonder_cs_object,
  antennae,
  dist_ranges = NULL,
  doppler_ranges = NULL,
  dist_in_km = FALSE,
  collapse = FALSE,
  smoothed = FALSE
)
```

Arguments

`seasonder_cs_object`	A SeaSondeRCS object containing spectral data.
`antennae`	A vector specifying the antenna(s) from which to extract self-spectra. If not named, the antennae will be automatically named as "A1", "A2", etc.
`dist_ranges`	Optional. A list (or vector) of range cell indices or ranges of interest. If not provided, it defaults to using the full range available.
`doppler_ranges`	Optional. A list (or vector) of Doppler bin indices or ranges of interest. If not provided, defaults to the complete Doppler range.
`dist_in_km`	Logical; if `TRUE`, the distance ranges provided in kilometers are converted into range cell numbers.
`collapse`	Logical; if `TRUE`, the nested list structure of the output is flattened into a single list.
`smoothed`	Logical; if `TRUE`, smoothed self-spectra data is used (via `seasonder_SmoothSS`); otherwise, raw self-spectra data is used.

Details

The function operates as follows:

1. If `doppler_ranges` is not provided, it sets a default list with the full Doppler range, using the total number of Doppler cells.
2. If `dist_ranges` is not provided, it sets a default list with the full range, using the total number of range cells.
3. If any of `antennae`, `dist_ranges`, or `doppler_ranges` are not named, they are automatically named using a default naming scheme.
4. Based on the `smoothed` flag, the function retrieves either smoothed self-spectra data via `seasonder_SmoothSS` or raw self-spectra data via `seasonder_getSeaSondeRCS_antenna_SSdata`.
5. If `dist_in_km` is `TRUE`, the distance ranges provided in kilometers are converted to range cell numbers using `seasonder_rangeCellsDists2RangeNumber`.
6. For each self-spectra matrix, the function slices the matrix over the specified range and Doppler intervals.
7. Finally, if `collapse = TRUE`, the nested list is flattened into a single-level list.

Value

A (potentially nested) list of self-spectra power matrices corresponding to each combination of antenna, range interval, and Doppler interval. If `collapse = TRUE`, the list is flattened.

`seasonder_getVersion` *Get the version value from a SeaSondeR object*

Description

Get the version value from a SeaSondeR object

Usage

```
seasonder_getVersion(seasonder_obj)
```

Arguments

`seasonder_obj`	A SeaSondeR object.

Value

The version value.

Examples

```
# Get version from a SeaSondeRCS object
cs_file <- system.file("css_data/CSS_TORA_24_04_04_0700.cs", package = "SeaSondeR")
apm_file <- system.file("css_data/MeasPattern.txt", package = "SeaSondeR")
apm_obj <- seasonder_readSeaSondeRAPMFile(apm_file)
cs_obj <- seasonder_createSeaSondeRCS(cs_file, seasonder_apm_object = apm_obj)
value <- seasonder_getVersion(cs_obj)
print(value)
```

---

seasonder_getVersion.SeaSondeRAPM

Get the version value from a SeaSondeRAPM object

Description

Get the version value from a SeaSondeRAPM object

Usage

```
## S3 method for class 'SeaSondeRAPM'
seasonder_getVersion(seasonder_obj)
```

Arguments

seasonder_obj	A SeaSondeRAPM object.

Value

The version value.

Examples

```
# Create a default SeaSondeRAPM object
obj <- seasonder_createSeaSondeRAPM()
# Retrieve version via the generic function
version <- seasonder_getVersion(obj)
print(version)
```

seasonder_getVersion.SeaSondeRCS

Get the version value from a SeaSondeRCS object

Description

Get the version value from a SeaSondeRCS object

Usage

```
## S3 method for class 'SeaSondeRCS'
seasonder_getVersion(seasonder_obj)
```

Arguments

seasonder_obj	A SeaSondeRCS object.

Value

The version value.

Examples

```
# Get version from a SeaSondeRCS object
cs_file <- system.file("css_data/CSS_TORA_24_04_04_0700.cs", package = "SeaSondeR")
apm_file <- system.file("css_data/MeasPattern.txt", package = "SeaSondeR")
apm_obj <- seasonder_readSeaSondeRAPMFile(apm_file)
cs_obj <- seasonder_createSeaSondeRCS(cs_file, seasonder_apm_object = apm_obj)
value <- seasonder_getVersion(cs_obj)
print(value)
```

seasonder_get_enabled_debug_points

Get enabled debug points in SeaSondeR

Description

This function returns the currently enabled debug points.

Usage

```
seasonder_get_enabled_debug_points()
```

Value

A character vector of enabled debug points.

Examples

```
seasonder_get_enabled_debug_points()
```

`seasonder_initCSDataStructure` *Initialize Cross-Spectra Data Structure for SeaSondeR*

Description

This function initializes a data structure for storing cross-spectra data related to SeaSonde radar measurements. It creates a list of matrices, each corresponding to different components of the SeaSonde data.

Usage

```
seasonder_initCSDataStructure(nRanges, nDoppler)
```

Arguments

`nRanges`	Integer, number of range cells in the radar measurement. Specifies the number of rows in each matrix.
`nDoppler`	Integer, number of Doppler bins in the radar measurement. Specifies the number of columns in each matrix.

Value

A list containing matrices for different cross-spectra components:

- `SSA1`: Matrix for SSA1 component, filled with `NA_real_`.
- `SSA2`: Matrix for SSA2 component, filled with `NA_real_`.
- `SSA3`: Matrix for SSA3 component, filled with `NA_real_`.
- `CS12`: Matrix for CS12 component, complex numbers with `NA_real_` real and imaginary parts.
- `CS13`: Matrix for CS13 component, complex numbers with `NA_real_` real and imaginary parts.
- `CS23`: Matrix for CS23 component, complex numbers with `NA_real_` real and imaginary parts.
- `QC`: Quality control matrix, filled with `NA_real_`.

s

`seasonder_initializeAttributesSeaSondeRAPM` *Initialize Attributes for a SeaSondeRAPM Object*

Description

This function initializes attributes for a SeaSondeRAPM object, including metadata and properties.

Usage

```
seasonder_initializeAttributesSeaSondeRAPM(calibration_matrix, ...)
```

Arguments

`calibration_matrix`	A 2 x b complex matrix, where b is the number of bearings for calibration.
`...`	Additional named attributes that may override the defaults.

Details

The function initializes the following attributes:

- `quality_matrix`: A 3 x b complex matrix for quality data, where b is the number of bearings.
- `BEAR`: A numeric vector for bearings (degrees CCW from the site bearing).
- `Type`: Character string for antenna pattern type.
- `Creator`: Object creator name. Default is an empty character vector.
- `SiteName`: Site name (not the same as SiteCode). Default is an empty character vector.
- `SiteOrigin`: Numeric vector with two elements representing the Station GPS location. Default is `c(0,0)`.
- `FileName`: Default is an empty character vector.
- `CreateTimeStamp`: APM file creation time. Default is current system date and time.
- `ProcessingSteps`: Processing steps applied to this object. Default is an empty character vector.
- `AmplitudeFactors`: Numeric vector with two elements for the amplitude factors. Default is `c(0,0)`.
- `AntennaBearing`: Site bearing (CW degrees from true north). Default is an empty numeric vector.
- `StationCode`: 4-character station code. Default is an empty character vector.
- `BearingResolution`: In degrees. Default is an empty numeric vector.
- `Smoothing`: Numeric vector indicating smoothing applied to the antenna pattern. Default is an empty numeric vector.
- `CommentLine`: Metadata lines in the data file not matching any other attribute. Default is an empty character vector.
- `FileID`: File's UUID. Default is an empty character vector.
- `PhaseCorrections`: Numeric vector with two elements for phase corrections. Default is `c(0,0)`.

Value

A list containing initialized attributes for a SeaSondeRAPM object.

See Also

seasonder_createSeaSondeRAPM, seasonder_validateAttributesSeaSondeRAPM

Examples

```
# Initialize attributes for a dummy calibration matrix
attrs <- seasonder_initializeAttributesSeaSondeRAPM(matrix(1:6, nrow = 3))
```

---

seasonder_initMUSICData

Initialize MUSIC Data for SeaSondeR

Description

This function initializes the MUSIC data structure for a SeaSondeR cross-spectral object, including optional interpolation, parameter setup, and pre-computed placeholders for MUSIC analysis.

Usage

```
seasonder_initMUSICData(
  seasonder_cs_object,
  range_cells = NULL,
  doppler_bins = NULL,
  NULL_MUSIC = FALSE
)
```

Arguments

seasonder_cs_object	A SeaSondeR cross-spectral object containing metadata about the radar system.
range_cells	An optional vector specifying the range cells to include. Defaults to all range cells in the object.
doppler_bins	An optional vector specifying the Doppler bins to include. Defaults to all Doppler bins in the object.
NULL_MUSIC	Logical. If TRUE, initializes the MUSIC structure with a NULL placeholder (see seasonder_NULLSeaSondeRCS_MUSIC). Defaults to FALSE.

Details

The function performs the following steps:

1. Ensures the SeaSondeR object has valid interpolation and parameter settings for MUSIC analysis.
2. Initializes the MUSIC data structure. If `NULL_MUSIC` is `FALSE`, the structure is populated with range cell and Doppler bin combinations.
3. Computes proportion of dual solutions for MUSIC using `seasonder_MUSICComputePropDualSols`.
4. Initializes interpolated data for cross-spectral analysis using `seasonder_MUSICInitInterpolatedData`.

The final object is ready for further MUSIC analysis steps, such as computing Direction of Arrival (DOA).

Value

The updated SeaSondeR cross-spectral object with initialized MUSIC-related attributes.

See Also

`seasonder_NULLSeaSondeRCS_MUSIC` for initializing a NULL structure. `seasonder_initSeaSondeRCS_MUSIC` for range and Doppler-based initialization. `seasonder_MUSICInitInterpolatedData` for interpolated data initialization.

Examples

```
# Minimal example for initializing MUSIC data (all range cells and Doppler bins)
cs_file <- system.file("css_data/CSS_TORA_24_04_04_0700.cs", package = "SeaSondeR")
apm_file <- system.file("css_data/MeasPattern.txt", package = "SeaSondeR")
apm_obj <- seasonder_readSeaSondeRAPMFile(apm_file)
cs_obj <- seasonder_createSeaSondeRCS(cs_file, seasonder_apm_object = apm_obj)
music_obj <- seasonder_initMUSICData(cs_obj)

# Example: specific range cells and Doppler bins
music_obj2 <- seasonder_initMUSICData(
  cs_obj,
  range_cells = c(1, 2),
  doppler_bins = c(1, 2, 5, 10)
)
```

`seasonder_initSeaSondeRCS_MUSIC`

Initialize SeaSondeR MUSIC Data Structure

Description

This function initializes a data structure for storing MUSIC analysis results for a given SeaSondeR cross-spectral object.

Usage

```
seasonder_initSeaSondeRCS_MUSIC(
  seasonder_cs_object,
  range_cells = NULL,
  doppler_bins = NULL
)
```

Arguments

seasonder_cs_object	A SeaSondeR cross-spectral object containing metadata about the radar system.
range_cells	An optional vector specifying the range cells to include. Defaults to all range cells in the object.
doppler_bins	An optional vector specifying the Doppler bins to include. Defaults to all Doppler bins in the object.

Details

The function creates a tibble with pre-computed range, frequency, and radial velocity values for the specified range cells and Doppler bins. It also initializes placeholders for MUSIC-related parameters such as covariance matrices, eigen decompositions, projections, DOA solutions, and more.

Columns in the resulting tibble include:

- `range_cell`: Range cell indices.
- `doppler_bin`: Doppler bin indices.
- `range`: Computed range values for the specified range cells.
- `freq`: Computed frequency values for the specified Doppler bins.
- `radial_v`: Computed radial velocities for the specified Doppler bins.
- `cov`: Initialized covariance matrices (see `seasonder_MUSICInitCov`).
- `eigen`: Initialized eigen decompositions (see `seasonder_MUSICInitEigenDecomp`).
- `projections`: Initialized projection matrices (see `seasonder_MUSICInitProjections`).
- `DOA_solutions`: Initialized DOA solutions (see `seasonder_MUSICInitDOASolutions`).
- `eigen_values_ratio`: Placeholder for the ratio of eigenvalues.
- `P1_check`: Logical placeholder for the P1 criterion (default is `TRUE`).
- `retained_solution`: Placeholder for the type of retained solution (`"dual"` by default).
- `DOA`: Placeholder for final DOA results.
- `lonlat`: Placeholder for longitude and latitude data as a data frame.

Value

A tibble with initialized MUSIC analysis data for the specified range cells and Doppler bins.

See Also

seasonder_NULLSeaSondeRCS_MUSIC for a NULL initialized structure. seasonder_MUSICInitCov, seasonder_MUSICInitEigenDecomp, seasonder_MUSICInitProjections, seasonder_MUSICInitDOASolutions for initializing individual components.

---

seasonder_int_to_raw *Convert an integer to raw bytes using a 64-bit representation*

Description

This function converts an integer to a raw byte representation using a 64-bit (8-byte) format. It leverages the bit64 package to handle the 64-bit integer representation and conversion.

Usage

```
seasonder_int_to_raw(x)
```

Arguments

x	An integer to be converted to raw bytes.

Details

The function follows these steps:

1. Convert the integer to a 64-bit format using bit64::as.integer64.
2. Convert the 64-bit integer to a bit string.
3. Split the bit string into individual bits.
4. Reorder the bits into groups of 8, reversing the order within each group.
5. Convert the reordered bits back to raw bytes.

Value

A raw vector representing the 64-bit format of the provided integer.

seasonder_is_debug_point_enabled *Check if a debug point is enabled in SeaSondeR*

Description

This function checks whether the provided debug point is enabled.

Usage

```
seasonder_is_debug_point_enabled(debug_point)
```

Arguments

debug_point	A character string specifying the debug point.

Value

TRUE if the debug point is enabled, FALSE otherwise.

Examples

```
seasonder_is_debug_point_enabled("example_debug")
```

seasonder_lastLog *Retrieve the Last Log Entry*

Description

This function fetches and splits the log entries, then returns the last entry.

Usage

```
seasonder_lastLog(...)
```

Arguments

...	Arguments passed to seasonder_splitLog.

Value

A character vector representing the last log entry.

Examples

```
# Enable logging
seasonder_enableLogs()
# Log a test message
seasonder_log("Test log entry", "info")
# Retrieve the last log entry
head(seasonder_lastLog())
```

`seasonder_limitFORCurrentRange` *Limit First Order Region (FOR) Based on Maximum Radial Velocity*

Description

This function removes Doppler bins from the detected First Order Region (FOR) if their corresponding radial velocity exceeds a predefined maximum threshold.

Usage

```
seasonder_limitFORCurrentRange(seasonder_cs_object)
```

Arguments

`seasonder_cs_object`
: A `SeaSondeRCS` object containing spectral data and FOR parameters.

Details

Steps in Current Range Limiting:

1. **Retrieve Maximum Velocity Threshold:**
   - Extracts the `currmax` parameter from `seasonder_getFOR_parameters`.
2. **Obtain Current FOR Detection Results:**
   - Retrieves the existing FOR Doppler bin indices from `seasonder_getSeaSondeRCS_FOR`.
3. **Compute Radial Velocities for Doppler Bins:**
   - Calls `seasonder_getBinsRadialVelocity` to convert Doppler bins into radial velocities.
4. **Identify Bins Exceeding Maximum Velocity:**
   - Finds the Doppler bins where the absolute radial velocity is greater than or equal to `currmax`.
5. **Filter Out Exceeding Bins:**
   - Uses `setdiff` to remove bins exceeding `currmax` from the FOR region.
6. **Store Updated FOR Data in Object:**
   - Updates the `SeaSondeRCS` object with the filtered FOR results.

This function ensures that only Doppler bins corresponding to physically realistic radial velocities are included in the first-order Bragg region.

Value

The updated SeaSondeRCS object with the FOR bins filtered based on maximum velocity.

See Also

- seasonder_getBinsRadialVelocity for computing radial velocities.
- seasonder_getSeaSondeRCS_FOR for retrieving FOR bin indices.
- seasonder_setSeaSondeRCS_FOR for storing updated FOR data.

---

seasonder_log *seasonder_log function*

Description

This function creates a logging message and signals a seasonder_log condition.

Usage

```
seasonder_log(message, level = "info")
```

Arguments

message	A character string indicating the message to be logged.
level	A character string that defines the level of the log. It can be "info", "error", or "fatal". Default is "info".

Value

Invisibly returns the generated log message string.

Examples

```
seasonder_log("This is an info message")
seasonder_log("This is an error message", "error")
seasonder_log("This is a fatal message", "fatal")
```

seasonder_logAndAbort *Log and Abort Message in SeaSondeR*

Description

This function logs a message to the SeaSondeR logging system and aborts execution. It prefixes the abort message with the name of the calling function.

Usage

```
seasonder_logAndAbort(msg, calling_function = NULL, ...)
```

Arguments

msg	A character string indicating the message.
calling_function	Function where the condition occurred. If NULL (default), the code determines the caller.
...	Additional arguments passed to `rlang::abort`.

Value

This function does not return as it always aborts execution.

Examples

```
my_function <- function() {
  seasonder_logAndAbort("This is a message")
}
# Demonstrate abort without stopping execution
try(my_function(), silent = TRUE)
```

seasonder_logAndMessage *Log and Inform Message in SeaSondeR*

Description

This function logs a message to the SeaSondeR logging system and also informs the message to the console. It prefixes the message with the name of the calling function.

Usage

```
seasonder_logAndMessage(msg, log_level = "info", calling_function = NULL, ...)
```

Arguments

msg	A character string indicating the message to be logged and informed.
log_level	A character string indicating the level of the log ("info", "error", "fatal"). Default is "info".
calling_function	Function where the condition occurred. If NULL (default), the code determines the caller.
...	Additional arguments passed to `rlang::inform` (if log_level="info") or `rlang::warn` (if log_level="error").

Value

Invisibly returns no value; used solely for its side effects of logging and messaging.

Examples

```
my_function <- function() {
  seasonder_logAndMessage("This is a message", "info")
}
my_function()
```

---

seasonder_logArchiver *Archive Log Entries*

Description

Archives log entries based on their levels: INFO, ERROR, or FATAL. If paths are not provided, temporary files will be used.

Usage

```
seasonder_logArchiver(
  log_path = NULL,
  log_info_path = log_path,
  log_error_path = log_info_path,
  log_fatal_path = log_error_path
)
```

Arguments

log_path	Path to the main log file.
log_info_path	Path to the INFO level log file.
log_error_path	Path to the ERROR level log file.
log_fatal_path	Path to the FATAL level log file.

Value

When temporary files are used, returns a character string with the main log file path; otherwise, returns an invisible value indicating that logs were archived.

Examples

```
seasonder_logArchiver()
```

`seasonder_MUSICBearing2GeographicalBearing`
Convert MUSIC Bearings to Geographic Bearings

Description

This function converts MUSIC bearings (relative to the antenna) into geographic bearings using the antenna's bearing information from a `SeaSondeRAPM` object.

Usage

```
seasonder_MUSICBearing2GeographicalBearing(bearings, seasonder_apm_object)
```

Arguments

`bearings`	A list of numeric vectors containing MUSIC bearings in degrees. Each vector corresponds to a set of bearings relative to the antenna.
`seasonder_apm_object`	A `SeaSondeRAPM` object containing the antenna's metadata, including the antenna's bearing.

Details

The geographic bearing is calculated by:

1. Multiplying the MUSIC bearings by -1 to invert their direction.
2. Adjusting the angles to the range [0, 360) using modulo 360.
3. Adding the antenna bearing to each value and wrapping the result to the range [0, 360) again using modulo 360.

The formula for each bearing is: $geo_bearing = ((-1 * music_bearing \%\%360) + antenna_bearing) \%\%360$.

Value

A list of numeric vectors containing the geographic bearings in degrees.

See Also

- seasonder_getSeaSondeRAPM_AntennaBearing
- %>%
- map

---

seasonder_MUSICCheckEigenValueRatio *Validate Eigenvalue Ratio Using MUSIC Algorithm*

Description

This function implements the P1 test for solutions derived using the MUSIC algorithm. The test checks the ratio between the largest and the second-largest eigenvalues, which serves as an indicator of signal quality.

Usage

```
seasonder_MUSICCheckEigenValueRatio(seasonder_cs_object)
```

Arguments

seasonder_cs_object	A SeaSondeRCS object containing the MUSIC solutions and related data.

Details

The P1 test is based on the ratio of the largest eigenvalue (lambda1) to the second-largest eigenvalue (lambda2):

Ratio = lambda1 / lambda2

This ratio is compared to a threshold defined in the MUSIC parameters to determine whether the solution is considered valid. Solutions failing this test are marked as "single."

Value

The updated SeaSondeRCS object with the following modifications:

- A new column eigen_values_ratio in the MUSIC data.
- A logical column P1_check indicating whether each solution passes the P1 test.
- Updated retained_solution values for solutions that fail the test.

See Also

seasonder_getSeaSondeRCS_MUSIC, seasonder_setSeaSondeRCS_MUSIC

`seasonder_MUSICCheckSignalMatrix` *Validate Signal Matrix Power Ratios Using MUSIC Algorithm*

Description

This function implements the P3 test for solutions derived using the MUSIC algorithm. The test evaluates the ratio between the diagonal (P_diag) and off-diagonal (P_off-diag) elements of the signal covariance matrix. Specifically, the ratio is computed as:

Usage

```
seasonder_MUSICCheckSignalMatrix(seasonder_cs_object)
```

Arguments

`seasonder_cs_object`
: A SeaSondeRCS object containing MUSIC data (including DOA solutions and power matrices).

Details

Ratio = P_off_diag / P_diag

where P_diag is the product of the absolute values of the diagonal elements and P_off_diag is the square of the absolute value of the upper-left off-diagonal element.

The computed ratio is compared with the threshold parameter (the third element in the MUSIC parameters). For each dual-bearing solution (i.e. when exactly two bearings are present), if the ratio is less than the reciprocal of the threshold, the solution passes the P3 test; otherwise, it is marked as "single".

For each entry in the MUSIC data, the function:

1. Extracts the covariance matrix power from the dual DOA solution (`DOA_sol$dual$P`).
2. Computes the ratio by taking the product of the absolute diagonal elements and the square of the absolute off-diagonal element.
3. Retrieves the threshold parameter for the P3 test.
4. Validates each solution by checking that:
   - The solution has exactly two bearings.
   - The computed ratio is available (not NA) and less than 1 divided by the threshold.
5. Updates the `retained_solution` field to "single" for solutions that do not pass the test.

Value

The updated SeaSondeRCS object in which:

- A new column `diag_off_diag_power_ratio` is added to the MUSIC data.
- A logical column `P3_check` indicates if each solution passes the P3 test.
- The `retained_solution` field of solutions that fail the test is updated to "single".

See Also

seasonder_getSeaSondeRCS_MUSIC to retrieve MUSIC data, seasonder_setSeaSondeRCS_MUSIC to update MUSIC data, and seasonder_getSeaSondeRCS_MUSIC_parameters to retrieve MUSIC parameters.

---

seasonder_MUSICCheckSignalPowers

Validate Signal Power Ratios Using MUSIC Algorithm

Description

This function implements the P2 test for solutions derived using the MUSIC algorithm. The test evaluates the ratio between the largest and smallest signal powers for dual-bearing solutions.

Usage

```
seasonder_MUSICCheckSignalPowers(seasonder_cs_object)
```

Arguments

`seasonder_cs_object`
: A SeaSondeRCS object containing the MUSIC solutions and related data.

Details

The P2 test is based on the ratio of the largest signal power (P_{max}) to the smallest signal power (P_{min}):

$$Ratio = \frac{P_{max}}{P_{min}}$$

This ratio is compared to a threshold defined in the MUSIC parameters. Only solutions that meet the following criteria are retained:

- The solution has two bearings.
- The signal power ratio is below the threshold.

Solutions failing this test are marked as "single."

Value

The updated SeaSondeRCS object with the following modifications:

- A new column `signal_power_ratio` in the MUSIC data.
- A logical column `P2_check` indicating whether each solution passes the P2 test.
- Updated `retained_solution` values for solutions that fail the test.

See Also

seasonder_getSeaSondeRCS_MUSIC, seasonder_setSeaSondeRCS_MUSIC

`seasonder_MUSICComputeCov` *Calculate the MUSIC Covariance Matrix for each Given Cell Range and Doppler Bin*

Description

This function computes the Multiple Signal Classification (MUSIC) covariance matrix for each cell range and Doppler bin from SeaSonde Cross Spectra (CS) data. The MUSIC algorithm is used in direction finding and spectral estimation.

Usage

```
seasonder_MUSICComputeCov(seasonder_cs_object)
```

Arguments

`seasonder_cs_object` A SeaSondeRCS object containing the cross-spectra data.

Details

The MUSIC algorithm estimates the direction of arrival (DOA) of signals, requiring the computation of a covariance matrix from sensor data. This function constructs the covariance matrix by iterating through the auto-spectra (`SSA{i}`) and cross-spectra (`CSij`) fields of the cross-spectra data.

For diagonal elements ($i = j$), the matrix uses data from the auto-spectra field corresponding to the antenna index (`SSA1`, `SSA2`, or `SSA3`). Negative values in `SSA3`, which indicate noise or interference, are converted to their absolute values before use, as per the Cross Spectra File Format Version 6 guidelines.

Off-diagonal elements ($i \neq j$) are derived from cross-spectra fields, such as `CS12` or `CS23`. If the row index is greater than the column index, the conjugate of the value is used.

Value

A SeaSondeRCS object updated with a computed 3x3 complex covariance matrix for each cell range and Doppler bin. The covariance matrix is stored in the MUSIC data field. Each matrix element C_{ij} is calculated based on auto-spectra (for diagonal elements) or cross-spectra (for off-diagonal elements). - Diagonal elements ($i = j$) are derived from auto-spectra `SSA{i}`. - Off-diagonal elements ($i \neq j$) are derived from cross-spectra `CSij`. - Auto-spectra values for the third antenna (`SSA3`) are taken as absolute values to comply with CODAR's recommendation to handle negative values indicating noise or interference.

References

Cross Spectra File Format Version 6, CODAR. (2016). Paolo, T. de, Cook, T. & Terrill, E. Properties of HF RADAR Compact Antenna Arrays and Their Effect on the MUSIC Algorithm. OCEANS 2007 1–10 (2007) doi:10.1109/oceans.2007.4449265.

See Also

seasonder_getSeaSondeRCS_MUSIC, seasonder_setSeaSondeRCS_MUSIC

`seasonder_MUSICComputeDOAProjections`
Compute DOA Functions Using the MUSIC Algorithm

Description

This function calculates the Direction of Arrival (DOA) functions based on the MUSIC algorithm for a given SeaSonde cross-spectra (CS) object. It projects the antenna patterns onto the noise subspace for each Doppler bin and computes single and dual signal solutions, following the MUSIC method.

Usage

```
seasonder_MUSICComputeDOAProjections(seasonder_cs_object)
```

Arguments

`seasonder_cs_object`
: An object representing the cross-spectra (CS) data from SeaSonde.

Details

The function operates as follows:

1. It sets a processing step indicating the start of DOA function computation.
2. Retrieves the Antenna Pattern Measurement (APM) and bearings associated with the CS object.
3. Iteratively computes projections of antenna pattern responses into the noise subspace for each Doppler bin using the MUSIC algorithm. This includes:
   - Initializing storage for projection results.
   - Calculating projections for single (m = 1) and dual (m = 2) signal solutions using the eigenvectors defining the noise subspace.
   - For each bearing, projecting the antenna manifold vector onto the noise subspace, as described by the formula:

$$DOA(\theta) = \frac{1}{A^*(\theta)E_n E_n^* A(\theta)}$$

     where:
     - E_n is the eigenvector matrix of the noise subspace.
     - $A(\theta)$ is the antenna pattern response vector at bearing θ.
     - $A^*(\theta)$ is its conjugate transpose.
4. Appends the computed DOA functions to the MUSIC data of the CS object.
5. Updates the processing step to indicate completion.

Value

The updated `seasonder_cs_object` with the MUSIC DOA functions computed and appended.

References

- Paolo, T. de, Cook, T., & Terrill, E. (2007). Properties of HF RADAR Compact Antenna Arrays and Their Effect on the MUSIC Algorithm. *OCEANS 2007*, 1–10. doi:10.1109/oceans.2007.4449265.

See Also

`seasonder_compute_antenna_pattern_proyections` for computing projections.

`seasonder_MUSICComputePropDualSols`
Compute the Proportion of Dual Solutions in MUSIC Data

Description

This function calculates the proportion of "dual" solutions in the MUSIC data associated with a given `SeaSondeRCS` object. It updates the object with the computed proportion as a new attribute.

Usage

```
seasonder_MUSICComputePropDualSols(seasonder_cs_object)
```

Arguments

`seasonder_cs_object`
: A `SeaSondeRCS` object containing MUSIC data and other related attributes.

Details

The function performs the following steps:

1. Extracts the MUSIC data from the provided `SeaSondeRCS` object.
2. Computes the proportion of entries in the `retained_solution` column of the MUSIC data that are labeled as "dual".
3. Updates the `SeaSondeRCS` object by adding the computed proportion as an attribute using `seasonder_setSeaSondeRCS_MUSIC_dual_solutions_proportion`.

Value

A `SeaSondeRCS` object with the calculated proportion of "dual" solutions stored as an attribute. This attribute can be accessed using a relevant getter function.

See Also

`seasonder_getSeaSondeRCS_MUSIC` to retrieve the MUSIC data. `seasonder_setSeaSondeRCS_MUSIC_dual_solutions_p` to set the computed proportion.

`seasonder_MUSICComputeSignalPowerMatrix` *Compute Signal Power Matrix for MUSIC Algorithm*

Description

This function computes the signal power matrix for each direction of arrival (DOA) solution obtained from the MUSIC algorithm. It updates the MUSIC data in the provided SeaSondeRCS object with the computed power matrices.

Usage

```
seasonder_MUSICComputeSignalPowerMatrix(seasonder_cs_object)
```

Arguments

`seasonder_cs_object`
: A SeaSondeRCS object containing MUSIC data, including eigenvalues, eigenvectors, and DOA solutions.

Details

The function performs the following steps:

1. Retrieves the MUSIC data from the SeaSondeRCS object.
2. Defines an internal function to update the DOA solutions with computed power matrices:
   - For dual steering vectors (`DOA_sol$dual$a`), computes the power matrix using `seasonder_computePowerMatrix` and updates `DOA_sol$dual$P`.
   - For single steering vectors (`DOA_sol$single$a`), computes the power matrix using `seasonder_computePowerMat` and updates `DOA_sol$single$P`.
3. Iterates through the MUSIC data, applying the update function to each set of eigenvalues and DOA solutions.
4. Updates the SeaSondeRCS object with the modified MUSIC data.

Value

The updated SeaSondeRCS object with the MUSIC data containing the computed power matrices for both dual and single solutions.

See Also

`seasonder_computePowerMatrix`

`seasonder_MUSICCovDecomposition` *Eigen Decomposition of the MUSIC Covariance Matrix*

Description

Performs the eigen decomposition of a MUSIC covariance matrix to obtain the eigenvalues and eigenvectors. This decomposition is a critical step in the MUSIC algorithm for spectral estimation and direction finding, as it enables the identification of the signal and noise subspaces.

Usage

```
seasonder_MUSICCovDecomposition(seasonder_cs_object)
```

Arguments

`seasonder_cs_object`
: A SeaSondeRCS object containing the covariance matrices derived from cross-spectra data.

Details

The covariance matrix represents one Doppler cell of the averaged cross-spectra of three received signals. This matrix captures the summation of signals from all bearings (plus noise) received by the antennas. To estimate the direction of arrival (DOA), the covariance matrix is subjected to eigenvalue decomposition (diagonalization) to estimate the signal and noise subspaces.

In practical HF radar systems, there are two primary sources of noise:

1. *System (thermal) noise*: Generated by the receiving equipment and assumed to be uncorrelated between antennas.
2. *Spatial noise field*: Includes wind-wave noise and current noise, modeled as Gaussian, which introduces correlation.

The eigenvalue decomposition produces:

- Three eigenvalues, ordered from largest to smallest.
- Three corresponding eigenvectors forming a 3-dimensional orthonormal basis.

Based on the largest eigenvalues:

- If there is one signal present, the first eigenvector defines a 1-dimensional signal subspace, and the remaining eigenvectors represent a 2-dimensional noise subspace.
- If two signals are present, the first two eigenvectors form a 2-dimensional signal subspace, while the remaining eigenvector represents a 1-dimensional noise subspace.

The signal and noise subspaces are orthogonal. This decomposition facilitates identifying the signal's direction by finding the antenna manifold that best fits the signal subspace.

Value

An updated SeaSondeRCS object where each Doppler cell includes the eigenvalues and eigenvectors of its covariance matrix. The eigenvalues are sorted in descending order, and the eigenvectors are aligned accordingly. The updates include:

- `eigen$values`: A numeric vector containing the sorted eigenvalues for each Doppler cell.
- `eigen$vectors`: A 3x3 matrix of the corresponding eigenvectors for each Doppler cell, aligned with the eigenvalues.

References

Paolo, T. de, Cook, T. & Terrill, E. Properties of HF RADAR Compact Antenna Arrays and Their Effect on the MUSIC Algorithm. OCEANS 2007 1–10 (2007) doi:10.1109/oceans.2007.4449265.

See Also

`seasonder_MUSICComputeCov` for computing the covariance matrix.

---

seasonder_MUSICExtractDOASolutions

Extract Direction of Arrival (DOA) Solutions Using the MUSIC Algorithm

---

Description

This function processes a set of MUSIC projection data to extract Direction of Arrival (DOA) solutions for radar signals. It implements the approach described in Paolo and Terril (2007) for HF radar analysis by first reversing the projection distances to enhance peak visibility, then detecting peaks for both single and dual solution cases. Finally, it maps the detected peak locations back to bearing values.

Usage

```
seasonder_MUSICExtractDOASolutions(
  projections,
  valid_bearings,
  seasonder_apm_obj
)
```

Arguments

`projections`	A numeric matrix of projection data where each column represents a set of MUSIC spectra for single and dual solutions. The matrix must have an attribute named `"bearings"` that contains the corresponding bearing angles (in degrees) for each column.
`valid_bearings`	A numeric vector of valid bearing values (in degrees) that are acceptable. Detected bearing peaks falling outside this set will be disregarded.

`seasonder_apm_obj`
: A matrix or similar object representing the Antenna Pattern Matrix (APM). The columns of `seasonder_apm_obj` correspond to bearings and are used to extract antenna response information for the detected peaks.

Details

The function proceeds as follows:

1. It retrieves the bearing angles from the attribute `"bearings"` of the `projections` matrix.
2. It computes the inverse of the absolute projection values for both 'single' and 'dual' solution modes to enhance peaks.
3. For single solutions, it detects the highest peak using `findpeaks`, then checks if the corresponding bearing is within the set of valid bearings.
4. If a valid single peak is found, it calculates the response in dB and determines the peak width by finding the indices where the response exceeds the (peak response - 3 dB) threshold.
5. For dual solutions, it similarly detects up to two peaks, filters them by valid bearings, and computes the response and peak width for each.
6. Finally, the function populates and returns a DOA solutions structure containing both single and dual solution fields.

Value

A list with two components corresponding to single and dual DOA solutions. Each component is a list containing:

- `bearing`: The detected bearing(s) for the solution (in degrees).
- `a`: A subset of the APM data (columns) corresponding to the detected peak.
- `peak_resp`: The peak response value(s) at the detected peak(s), expressed in dB.
- `peak_width`: The width of the peak(s) calculated from the 3 dB limit, in degrees.

References

Paolo, S., & Terril, E. (2007). Detection and characterization of signals in HF radar cross-spectra using the MUSIC algorithm. *Journal of Atmospheric and Oceanic Technology*.

See Also

`seasonder_MUSICExtractPeaks`, `findpeaks`

`seasonder_MUSICExtractPeaks` *Extract and Validate DOA Peaks Using MUSIC Algorithm*

Description

This function processes a `SeaSondeRCS` object to extract Direction of Arrival (DOA) solutions using the MUSIC algorithm and validates the retained solutions based on the extracted peaks.

Usage

```
seasonder_MUSICExtractPeaks(seasonder_cs_object)
```

Arguments

`seasonder_cs_object`
: An object of class `SeaSondeRCS` containing cross-spectra data processed with the MUSIC algorithm.

Details

The function performs the following operations:

1. Initializes the peak extraction process and logs the start.
2. Extracts DOA solutions for each set of projections using `seasonder_MUSICExtractDOASolutions`.
3. Validates and adjusts the retained solution types using `seasonder_MUSICExtractPeaksCheckRetainedSolution`.
4. Updates the `SeaSondeRCS` object with the extracted and validated solutions.
5. Logs the completion of the peak extraction process.

The MUSIC algorithm's implementation follows the theoretical framework outlined by Paolo and Terril (2007), emphasizing the identification of signal directions in HF radar cross-spectra.

Value

An updated `SeaSondeRCS` object with the following fields modified:

- `MUSIC`: Contains the extracted DOA solutions.
- `ProcessingSteps`: Includes a log of the peak extraction process.

References

Paolo, S., & Terril, E. (2007). Detection and characterization of signals in HF radar cross-spectra using the MUSIC algorithm. *Journal of Atmospheric and Oceanic Technology*.

See Also

`seasonder_MUSICExtractDOASolutions`, `seasonder_MUSICExtractPeaksCheckRetainedSolution`

`seasonder_MUSICExtractPeaksCheckRetainedSolution` *Validate Retained Solution in MUSIC Algorithm Peak Extraction*

Description

This function verifies and adjusts the retained solution type ("single" or "dual") based on the Direction of Arrival (DOA) solutions extracted using the MUSIC algorithm.

Usage

```
seasonder_MUSICExtractPeaksCheckRetainedSolution(ret_sol, DOA_sol)
```

Arguments

`ret_sol`	A character string specifying the initial solution type to retain. Valid values are `"single"` or `"dual"`.
`DOA_sol`	A list containing extracted DOA solutions, as returned by `seasonder_MUSICExtractDOASolutions`.

Details

The function performs the following checks:

1. If the retained solution is "dual" but no valid dual solution bearings exist, it defaults to "single" if valid.
2. If the retained solution is "single" but no valid single solution bearings exist, it defaults to "none".

This validation ensures the output solutions are consistent with the detected peaks, addressing potential discrepancies in the initial assumptions about the solution type.

Value

A character string indicating the validated solution type:

- `"single"`: If only one single solution bearing is valid.
- `"dual"`: If valid dual solution bearings are detected.
- `"none"`: If no valid bearings are found.

See Also

`seasonder_MUSICExtractPeaks`, `seasonder_MUSICExtractDOASolutions`

seasonder_MUSICInitCov

Initialize Covariance Matrix for MUSIC Algorithm

Description

This function initializes a covariance matrix for use in the MUSIC algorithm.

Usage

```
seasonder_MUSICInitCov()
```

Details

The covariance matrix is initialized as a 3 x 3 matrix filled with complex `NA` values. This structure is specifically designed for three-channel antenna configurations commonly used in SeaSondeR applications.

Value

A 3 x 3 matrix of complex values, each initialized to `NA_complex_`.

See Also

`seasonder_defaultMUSIC_parameters` for default MUSIC parameters.

seasonder_MUSICInitDOASolutions

Initialize Direction of Arrival (DOA) Solutions for MUSIC Algorithm

Description

This function initializes the data structure for storing Direction of Arrival (DOA) solutions calculated by the MUSIC algorithm.

Usage

```
seasonder_MUSICInitDOASolutions()
```

Details

The function returns a list containing two sub-lists, one for `"single"` solutions and another for `"dual"` solutions:

- `"single"`: Contains placeholders for single DOA solutions:
  - `bearing`: The bearing angle (`NA_real_` by default).
  - `a`: The complex steering vector (`NA_complex_` by default).
  - `P`: The power spectrum value (`NA_complex_` by default).
- `"dual"`: Contains placeholders for dual DOA solutions:
  - `bearing`: The bearing angle (`NA_real_` by default).
  - `a`: The complex steering vector (`NA_complex_` by default).
  - `P`: A 2 x 2 complex matrix initialized to `NA_complex_`.

Value

A list with initialized placeholders for `"single"` and `"dual"` DOA solutions.

See Also

`seasonder_MUSICInitCov` for initializing covariance matrices. `seasonder_MUSICInitProjections` for initializing projection matrices.

`seasonder_MUSICInitEigenDecomp` *Initialize Eigenvalue Decomposition Structure for MUSIC Algorithm*

Description

This function initializes the data structure for storing the eigenvalue decomposition results used in the MUSIC algorithm.

Usage

```
seasonder_MUSICInitEigenDecomp()
```

Details

The function returns a list with the following components:

- `values`: A vector of length 3, initialized with `NA_complex_`, to hold the eigenvalues.
- `vectors`: A 3 x 3 matrix, initialized with `NA_complex_`, to hold the eigenvectors.

This structure is designed to support three-channel antenna configurations typical in SeaSondeR applications.

Value

A list with two elements:

- `values`: Eigenvalues as a complex vector.
- `vectors`: Eigenvectors as a complex matrix.

See Also

`seasonder_MUSICInitCov` for initializing covariance matrices.

`seasonder_MUSICInitInterpolatedData` *Initialize Interpolated Data for MUSIC Algorithm*

Description

This function initializes the data structure for storing interpolated cross-spectral data to be used in the MUSIC algorithm.

Usage

```
seasonder_MUSICInitInterpolatedData(seasonder_cs_object)
```

Arguments

`seasonder_cs_object`
: A SeaSondeR cross-spectral object containing metadata about the number of Doppler cells and range cells.

Details

The function retrieves the number of Doppler cells and range cells from the provided cross-spectral object and uses this information to initialize the interpolated data structure. The resulting structure is compatible with the dimensions of the cross-spectral data used in SeaSondeR.

The data structure is initialized using `seasonder_initCSDataStructure`, ensuring it contains placeholders for components such as `SSA1`, `SSA2`, `SSA3`, `CS12`, `CS13`, `CS23`, and `QC`.

Value

A list containing the initialized interpolated data structure with placeholders for cross-spectral components.

See Also

`seasonder_initCSDataStructure` for details on the cross-spectral data structure.

`seasonder_MUSICInitProjections` *Initialize Projection Matrix for MUSIC Algorithm*

Description

This function initializes a projection matrix for use in the MUSIC algorithm.

Usage

```
seasonder_MUSICInitProjections(bearings = 0)
```

Arguments

`bearings`	A numeric vector representing the bearings (in degrees) for which projections are initialized. Defaults to `0`.

Details

The function creates a 2 x n complex matrix, where n is the number of bearings. The matrix rows are labeled:

- `"single"`: For single projections.
- `"dual"`: For dual projections.

An attribute `"bearings"` is attached to the matrix, storing the input bearings vector.

Value

A 2 x n matrix of complex values, each initialized to `NA_complex_`, with row names `"single"` and `"dual"`. The input bearings are stored as an attribute.

See Also

`seasonder_MUSICInitCov` for initializing covariance matrices.

`seasonder_MUSICLonLat` *Map MUSIC Bearings to Geographic Coordinates*

Description

This function calculates geographic coordinates (latitude and longitude) for each MUSIC detection based on the range and direction of arrival (DOA) bearings from a `SeaSondeRCS` object.

Usage

```
seasonder_MUSICLonLat(seasonder_cs_object)
```

Arguments

`seasonder_cs_object`	A `SeaSondeRCS` object containing MUSIC detection data.

Details

This function performs the following operations:

1. Retrieves MUSIC data and original geographic coordinates (latitude and longitude) from the `seasonder_cs_object`. If these coordinates are not available, the origin is derived from the associated Antenna Pattern (APM) data.
2. Converts DOA bearings from MUSIC detections into geographic bearings using the APM object.
3. Computes latitude and longitude for each MUSIC detection based on the range and geographic bearings using `seasonder_computeLonLatFromOriginDistBearing`
4. Updates the `seasonder_cs_object` with the newly computed coordinates.

Value

A `SeaSondeRCS` object with updated MUSIC data, including geographic coordinates for each detection.

See Also

- `seasonder_getSeaSondeRCS_MUSIC`
- `seasonder_getSeaSondeRCS_APM`
- `seasonder_MUSICBearing2GeographicalBearing`
- `seasonder_computeLonLatFromOriginDistBearing`

Examples

```
# Create a SeaSondeRCS object for MUSIC example
cs_file <- system.file("css_data/CSS_TORA_24_04_04_0700.cs", package = "SeaSondeR")
apm_file <- system.file("css_data/MeasPattern.txt", package = "SeaSondeR")
apm_obj  <- seasonder_readSeaSondeRAPMFile(apm_file)
cs_obj   <- seasonder_createSeaSondeRCS(cs_file, seasonder_apm_object = apm_obj)
cs_obj <- seasonder_initMUSICData(
 cs_obj,
 range_cells = c(rep(5,11), rep(4,11)),
 doppler_bins = c(c(669:679),c(674:684))
)
cs_obj <- seasonder_runMUSIC(cs_obj)
updated_obj <- seasonder_MUSICLonLat(cs_obj)
print(updated_obj)
```

---

seasonder_MUSICSelectDOA

Select Direction of Arrival (DOA) from MUSIC Algorithm Results

---

Description

This function processes the results of the MUSIC algorithm, selects the relevant Direction of Arrival (DOA) based on the specified retained solution, and updates the corresponding SeaSondeRCS object with the selected DOA and updated processing steps.

Usage

```
seasonder_MUSICSelectDOA(seasonder_cs_object)
```

Arguments

seasonder_cs_object	A SeaSondeRCS object containing the results of the MUSIC algorithm and associated metadata.

Details

The function performs the following steps:

1. Updates the processing steps to indicate the start of the DOA selection process.
2. Retrieves the MUSIC algorithm results from the SeaSondeRCS object.
3. Maps the retained solution index to the corresponding DOA solution for each entry in the MUSIC results.
4. Stores the updated MUSIC results, including the selected DOA, back into the SeaSondeRCS object.
5. Updates the processing steps to indicate the end of the DOA selection process.

Value

An updated SeaSondeRCS object with the selected DOA stored in the MUSIC results and updated processing steps.

Processing Steps

The function appends the following processing steps to the ProcessingSteps attribute of the SeaSondeRCS object:

- Start of DOA selection.
- End of DOA selection.

See Also

seasonder_setSeaSondeRCS_ProcessingSteps to manage processing steps. seasonder_getSeaSondeRCS_MUSIC to retrieve MUSIC results. seasonder_setSeaSondeRCS_MUSIC to update MUSIC results.

---

seasonder_MUSICTestDualSolutions

Test Dual-Bearing Solutions Using MUSIC Algorithm

Description

This function applies a sequence of tests (P1, P2, and P3) to validate dual-bearing solutions derived using the MUSIC algorithm. The tests evaluate the quality of solutions based on eigenvalue ratios, signal power ratios, and covariance matrix power ratios.

Usage

```
seasonder_MUSICTestDualSolutions(seasonder_cs_object)
```

Arguments

seasonder_cs_object	A SeaSondeRCS object containing MUSIC solutions and related data.

Details

The function applies the following sequence of tests:

1. **P1: Eigenvalue Ratio Test**:
   - Evaluates the ratio between the largest and second-largest eigenvalues.
2. **P2: Signal Power Ratio Test**:
   - Validates the ratio of signal powers for dual-bearing solutions.
3. **P3: Signal Matrix Power Ratio Test**:
   - Checks the ratio of diagonal to off-diagonal powers in the covariance matrix.

Each test updates the MUSIC solutions in the input object, marking solutions that fail the tests as "single." The function also logs the start and end of the testing process as part of the object's processing steps.

Value

The updated SeaSondeRCS object with validated dual-bearing solutions and recorded processing steps.

See Also

seasonder_MUSICCheckEigenValueRatio, seasonder_MUSICCheckSignalPowers, seasonder_MUSICCheckSignalMatr
seasonder_setSeaSondeRCS_ProcessingSteps

---

seasonder_MUSIC_Bins2DopplerFreq

Map Doppler Bins to Doppler Frequencies

Description

This function retrieves the Doppler frequencies corresponding to specified Doppler bins for a given SeaSonde cross-spectral object.

Usage

```
seasonder_MUSIC_Bins2DopplerFreq(seasonder_cs_object, bins)
```

Arguments

seasonder_cs_object	A SeaSondeRCS object representing the cross-spectral data structure. It contains metadata and configuration for Doppler frequency and bin mapping.
bins	A numeric or integer vector of bin indices for which Doppler frequencies are needed.

Details

The function retrieves the full set of unnormalized Doppler bin frequencies using seasonder_getSeaSondeRCS_MUSIC_Dopp
and returns the frequencies corresponding to the provided bin indices. This is useful for translating bin-domain indices into physical Doppler frequency values for analysis or visualization.

Value

A numeric vector of Doppler frequencies corresponding to the input bin indices.

See Also

seasonder_getSeaSondeRCS_MUSIC_DopplerBinsFrequency

seasonder_MUSIC_DopplerFreq2Bins *Map Doppler Frequencies to Doppler Bins*

Description

This function maps specified Doppler frequency values to the corresponding Doppler bins for a given SeaSonde cross-spectral object.

Usage

```
seasonder_MUSIC_DopplerFreq2Bins(seasonder_cs_object, doppler_values)
```

Arguments

seasonder_cs_object	A SeaSondeRCS object representing the cross-spectral data structure. It contains metadata and configuration for Doppler frequency and bin mapping.
doppler_values	A numeric vector of Doppler frequency values to be mapped to Doppler bins.

Details

The function performs the following steps:

- Retrieves the unnormalized Doppler bin frequencies using seasonder_getSeaSondeRCS_MUSIC_DopplerBinsFrequer
- Retrieves the Doppler spectrum resolution using seasonder_getSeaSondeRCS_MUSIC_DopplerSpectrumResolution.
- Retrieves the total number of Doppler cells using seasonder_getSeaSondeRCS_MUSIC_nDopplerCells.
- Computes the Doppler bin indices corresponding to the input Doppler frequency values using seasonder_computeDopplerFreq2Bins.

This mapping is essential for translating frequency-domain values into bin indices used in further data processing or visualization.

Value

A numeric vector of Doppler bins corresponding to the input Doppler frequency values.

See Also

seasonder_getSeaSondeRCS_MUSIC_DopplerBinsFrequency, seasonder_getSeaSondeRCS_MUSIC_DopplerSpectrumR
seasonder_getSeaSondeRCS_MUSIC_nDopplerCells, seasonder_computeDopplerFreq2Bins

`seasonder_NormalizedDopplerFreq2Bins` *Convert Normalized Doppler Frequencies to Doppler Bins*

Description

This function converts a set of normalized Doppler frequencies into their corresponding Doppler bin indices within a `SeaSondeR` object.

Usage

```
seasonder_NormalizedDopplerFreq2Bins(seasonder_cs_object, doppler_values)
```

Arguments

`seasonder_cs_object`	A `SeaSondeR` cross-spectral object containing metadata about the Doppler bins.
`doppler_values`	A numeric vector specifying the normalized Doppler frequencies to be converted into bin indices.

Details

This function first retrieves the list of normalized Doppler frequencies from the given `SeaSondeR` object using `seasonder_getDopplerBinsFrequency`. The bin boundaries are computed using the first-order difference of these frequencies.

The function then applies `findInterval` to determine the corresponding bin index for each input Doppler frequency. The search process is affected by the following options:

- `rightmost.closed = TRUE`: The last bin interval is closed on the right, ensuring that the maximum normalized frequency is included in the last bin.
- `all.inside = FALSE`: Values that fall outside the range of the computed boundaries are assigned values below 1 or above the maximum bin index.
- `left.open = TRUE`: The left interval is open, meaning that values exactly equal to a boundary are assigned to the higher bin.

After `findInterval` determines the bin indices, values that are out of range (`bins < 1` or `bins > nDoppler`) are set to `NA`.

Value

An integer vector indicating the Doppler bin indices corresponding to the input normalized Doppler frequencies. Values that fall outside the valid bin range are assigned `NA`.

See Also

`seasonder_Bins2NormalizedDopplerFreq` for the inverse operation.

`seasonder_NormalizedDopplerFreq2DopplerFreq` *Convert Normalized Doppler Frequencies to Doppler Frequencies*

Description

This function converts normalized Doppler frequencies into their corresponding Doppler frequencies (in Hz) within a `SeaSondeR` object.

Usage

```
seasonder_NormalizedDopplerFreq2DopplerFreq(
  seasonder_cs_object,
  doppler_values
)
```

Arguments

`seasonder_cs_object`	A `SeaSondeR` cross-spectral object containing metadata about the Doppler bins.
`doppler_values`	A numeric vector specifying the normalized Doppler frequencies to be converted into Doppler frequencies (Hz).

Details

The function follows these steps:

1. Calls `seasonder_NormalizedDopplerFreq2Bins` to convert the input normalized Doppler frequencies into Doppler bin indices.
2. Calls `seasonder_Bins2DopplerFreq` to obtain the corresponding Doppler frequencies in Hz.

The relationship between the normalized and absolute Doppler frequencies is defined as:

$$f_{doppler} = f_{norm} \times f_{bragg}$$

where:

- $f_{doppler}$ is the Doppler frequency in Hz,
- f_{norm} is the normalized Doppler frequency,
- f_{bragg} is the Bragg frequency, computed based on radar wavelength.

Value

A numeric vector of Doppler frequencies (in Hz) corresponding to the input normalized Doppler frequencies.

See Also

`seasonder_NormalizedDopplerFreq2Bins` for converting normalized Doppler frequencies to bin indices. `seasonder_Bins2DopplerFreq` for converting bin indices to Doppler frequencies in Hz.

`seasonder_NULLSeaSondeRCS_MUSIC` *Initialize NULL Data Structure for SeaSondeR MUSIC Analysis*

Description

This function initializes a NULL data structure for storing results of the MUSIC analysis in SeaSondeR. The structure is designed as a tibble with pre-defined columns for range cells, Doppler bins, and various MUSIC-related parameters.

Usage

```
seasonder_NULLSeaSondeRCS_MUSIC()
```

Details

The initialized tibble contains the following columns:

- `range_cell`: Numeric vector representing range cell indices.
- `doppler_bin`: Numeric vector for Doppler bin indices.
- `range`: Numeric vector for range values.
- `freq`: Numeric vector for frequencies.
- `radial_v`: Numeric vector for radial velocities.
- `cov`: A list to store covariance matrices.
- `eigen`: A list to store eigenvalue decompositions.
- `projections`: A list to store projection matrices.
- `DOA_solutions`: A list to store Direction of Arrival (DOA) solutions.
- `eigen_values_ratio`: Numeric vector for the ratio of eigenvalues.
- `P1_check`: Logical vector indicating if the P1 criterion is satisfied.
- `retained_solution`: Character vector for the type of retained solution (`"single"` or `"dual"`).
- `DOA`: A list to store final DOA results.
- `lonlat`: A list containing a data frame with longitude (`lon`) and latitude (`lat`) values.

Value

A tibble with pre-defined columns and empty values, ready to be populated with MUSIC analysis results.

See Also

`seasonder_MUSICInitCov` for initializing covariance matrices. `seasonder_MUSICInitEigenDecomp` for initializing eigenvalue decompositions. `seasonder_MUSICInitProjections` for initializing projection matrices. `seasonder_MUSICInitDOASolutions` for initializing DOA solutions.

seasonder_plotAPMLoops *Plot APM Loops in a Polar Coordinate System*

Description

This function generates a polar plot of the antenna pattern loops from a SeaSonde RAPM object.

Usage

```
seasonder_plotAPMLoops(seasonder_apm_obj)
```

Arguments

seasonder_apm_obj	A SeaSonde RAPM object containing the antenna pattern data.

Value

A ggplot object displaying the magnitude of the two loops as a function of bearings.

Examples

```
# Plot loops from a test SeaSondeRAPM object
apm_file <- system.file("css_data/MeasPattern.txt", package = "SeaSondeR")
obj <- seasonder_readSeaSondeRAPMFile(apm_file)
plot <- seasonder_plotAPMLoops(obj)
```

seasonder_raw_to_int *Convert a Raw Vector to a 64-bit Integer*

Description

This function converts a raw vector to a 64-bit integer, handling both signed and unsigned conversions.

Usage

```
seasonder_raw_to_int(r, signed = FALSE)
```

Arguments

r	A raw vector to be converted.
signed	Logical, indicating whether the conversion should consider the value as signed (default is FALSE for unsigned).

Value

A 64-bit integer representation of the raw vector.

`seasonder_readCSField` *Read a CSField from a Binary Connection*

Description

This function reads specific data types from a binary connection, supporting various types including integer, float, double, complex, and strings.

Usage

```
seasonder_readCSField(con, type, endian = "big")
```

Arguments

`con`	A connection object to a binary file.
`type`	A character string identifying the type of data to read.
`endian`	A character string indicating the byte order. Options are "big" and "little" (default is "big").

Value

The value obtained from reading the CSField according to the specified type.

Supported Data Types

This function provides support for reading a variety of data types from a binary connection. The following data types are recognized and can be used for the `type` argument:

`CharN` Reads N characters from the connection where N is a positive integer. For example, `Char5` would read five characters.

`UInt8` Reads an 8-bit unsigned integer.

`SInt8` Reads an 8-bit signed integer.

`UInt16` Reads a 16-bit unsigned integer.

`SInt16` Reads a 16-bit signed integer.

`UInt32` Reads a 32-bit unsigned integer.

`SInt32` Reads a 32-bit signed integer.

`Float` Reads a single-precision floating-point number.

`Double` Reads a double-precision floating-point number.

`UInt64` Reads a 64-bit unsigned integer.

`SInt64` Reads a 64-bit signed integer.

`Complex` Reads a complex number by separately reading the real and imaginary parts, which are each represented as double-precision floating-point numbers.

`String` Reads a null-terminated string.

If the provided `type` does not match any of the supported data types, the function raises an error.

Condition Management

This function utilizes the `rlang` package to manage conditions and provide detailed and structured condition messages:

Condition Classes:

- `seasonder_cs_field_reading_error`: General error related to reading a CSField from the binary connection.
- `seasonder_cs_field_skipped`: Condition that indicates a CSField was skipped due to a reading error.

Condition Cases:

- Connection is not open.
- Error while reading value from connection.
- Read value of length 0 from connection (likely reached end of file).
- Unrecognized data type specified.

Restart Options: This function provides a structured mechanism to recover from errors during its execution using the `rlang::withRestarts` function. The following restart option is available:

`seasonder_skip_cs_field(cond, value)` This allows for the graceful handling of reading errors. If this restart is invoked, the function will log an error message indicating that a specific CSField reading was skipped and will return the value specified. The restart takes two arguments: `cond` (the condition or error that occurred) and `value` (the value to return if this CSField reading is skipped). To invoke this restart during a condition or error, you can use the helper function `seasonder_skip_cs_field(cond, value)`.

- **Usage**: In a custom condition handler, you can call `seasonder_skip_cs_field(cond, yourDesiredReturnValue)` to trigger this restart and skip the current CSField reading.
- **Effect**: If invoked, the function logs an error message detailing the reason for skipping, and then returns the value specified in the restart function call.

See Also

`seasonder_skip_cs_field`, `seasonder_raw_to_int`

seasonder_readCSSWBody

Read CSSW Body

Description

Reads the body section of a CSSW file, processing each cell block until the designated endpoint.

Usage

```
seasonder_readCSSWBody(
  connection,
  specs,
  size,
  dbRef,
  endian = "big",
  specs_key_size = NULL
)
```

Arguments

connection	A binary connection from which the body is read.
specs	A list specifying the body keys and formats.
size	The total number of bytes to read for the body section.
dbRef	Numeric decibel reference used for scaling.
endian	A character specifying byte order.
specs_key_size	Optional specification for the key size block.

Value

A list of processed body cells with applied sign corrections.

seasonder_readCSSWBodyRangeCell

Read a Body Range Cell and Apply Scaling if Required

Description

This function processes a block of keys from a binary connection according to a provided specification ('specs'). Each key is interpreted by reading it with `seasonder_readSeaSondeCSFileBlock` and processing it based on its key name. The key processing follows these rules:

Usage

```
seasonder_readCSSWBodyRangeCell(
  connection,
  specs,
  dbRef,
  endian = "big",
  specs_key_size = NULL
)
```

Arguments

connection	A binary connection from which keys and data are read.
specs	A list defining the expected keys and their formats.
dbRef	A numeric value providing the dB reference used in scaling.
endian	A string specifying the byte order ("big" or "little"). Defaults to "big".
specs_key_size	Optional specification for the key size block.

Details

- **Scaling Block ('scal')**: Reads scaling parameters (fmax, fmin, fscale, dbRef) using `seasonder_readCSSWFields` and stores them for later use.
- **Reduced Data Blocks (e.g., 'cs1a', 'cs2a', 'cs3a', 'c13m', 'c13a', etc.)**: Reads the block using `seasonder_read_reduced_encoded_data`. If scaling parameters were set by a preceding 'scal' block, the raw data is converted to voltage values using `seasonder_SeaSondeRCSSWApplyScaling`; otherwise, the raw data is returned.
- **Other Keys (e.g., 'csgn' and 'asgn')**: These keys invoke their specialized read functions for processing.

The function continues reading keys until it detects the 'END ' marker or a repeated 'indx' key, which signals the end of the block.

Value

A list with elements named after the keys read. For reduced data blocks, each element contains either the raw decoded data or the scaled voltage values if a 'scal' block had been applied.

---

seasonder_readCSSWFields

Read CSSW Fields

Description

Processes a block of keys from the binary connection according to provided specifications.

Usage

```
seasonder_readCSSWFields(connection, specs, endian, parent_key = NULL)
```

Arguments

connection	A binary connection.
specs	A list specifying the expected keys.
endian	A character indicating byte order.
parent_key	Optional parent key information.

Value

A named list as returned by seasonder_readSeaSondeCSFileBlock consistent with the provided specifications.

---

seasonder_readCSSWHeader

Read CSSW File Header

Description

This function reads the header section of a CSSW file from a binary connection. The CSSW file header contains a set of key blocks formatted according to the SeaSonde CSSW specification. The header section is processed recursively and terminates when one of the following conditions is met:

- A key with name "BODY" is encountered. In this case, the connection is rewound by 8 bytes to allow subsequent processing of the body.
- A key that is not defined in `current_specs` but is already present in the `keys_so_far` vector is encountered (indicative of repeated keys), which triggers termination.

Usage

```
seasonder_readCSSWHeader(
  connection,
  current_specs,
  endian = "big",
  parent_key = NULL,
  keys_so_far = c("CSSW", "HEAD"),
  specs_key_size = NULL
)
```

Arguments

connection	A binary connection from which to read the CSSW file header.
current_specs	A list representing the specification for the header; may contain nested subkeys.
endian	A character string indicating the byte order for reading numeric values ("big" or "little").
parent_key	(Optional) A list with information from the parent key block, used when processing nested keys.

keys_so_far	A character vector of keys already processed, used to avoid recursive loops. Defaults to c("CSSW", "HEAD").
specs_key_size	A specification for reading the key size block, often obtained from YAML specs.

Details

When no subkeys are specified in current_specs (i.e. current_specs comprises only simple field definitions), the function delegates the processing to seasonder_readCSSWFields.

The function processes the CSSW header recursively:

- If current_specs contains only field definitions, seasonder_readCSSWFields is called.
- When a key named "BODY" is encountered, it signifies the beginning of the body section; the function rewinds the connection 8 bytes and stops processing further keys.
- If a key is encountered that is not defined in current_specs but is already present in keys_so_far, the function also rewinds the connection 8 bytes and terminates header reading.
- Otherwise, the function updates keys_so_far, handles special cases (e.g., key "cs4h"), and calls itself recursively to process nested keys.

Value

A list containing the parsed CSSW header information. The returned list may be empty if a termination condition is encountered.

---

seasonder_readCSSWLims

Read CSSW Limits

Description

Reads a specified number of 32-bit unsigned integers from a binary connection and reshapes them into a matrix representing CSSW limits.

Usage

```
seasonder_readCSSWLims(connection, n_values, endian = "big")
```

Arguments

connection	A binary connection.
n_values	The number of 32-bit unsigned integers to read.
endian	A string specifying byte order ("big" or "little").

Value

A numeric matrix with four columns: LeftBraggLeftLimit, LeftBraggRightLimit, RightBraggLeftLimit, and RightBraggRightLimit.

`seasonder_readCSSYBodyRangeCell` *Read a Body Range Cell and Apply Scaling if Required*

Description

This function processes a block of keys from a binary connection according to a provided specification ('specs'). Each key is interpreted by reading it with `seasonder_readSeaSondeCSFileBlock` and processing it based on its key name. The key processing follows these rules:

Usage

```
seasonder_readCSSYBodyRangeCell(
  connection,
  specs,
  dbRef,
  endian = "big",
  specs_key_size = NULL
)
```

Arguments

`connection`	A binary connection from which keys and data are read.
`specs`	A list defining the expected keys and their formats.
`dbRef`	A numeric value providing the dB reference used in scaling.
`endian`	A string specifying the byte order ("big" or "little"). Defaults to "big".
`specs_key_size`	Optional specification for the key size block.

Details

- **Scaling Block ('scal')**: Reads scaling parameters (fmax, fmin, fscale, dbRef) using `seasonder_readCSSYFields` and stores them for later use.
- **Reduced Data Blocks (e.g., 'cs1a', 'cs2a', 'cs3a', 'c13r', 'c13i', etc.)**: Reads the block using `seasonder_read_reduced_encoded_data`. If scaling parameters were set by a preceding 'scal' block, the raw data is converted to voltage values using `seasonder_SeaSondeRCSSYApplyScaling`; otherwise, the raw data is returned.
- **Other Keys (e.g., 'csgn' and 'asgn')**: These keys invoke their specialized read functions for processing.

The function continues reading keys until it detects the 'END ' marker or a repeated 'indx' key, which signals the end of the block.

Value

A list representing a cell in the CSSY body.

seasonder_readCSSYHeader *Read CSSY File Header*

Description

This function reads the header section of a CSSY file from a binary connection. The CSSY file header contains a set of key blocks formatted according to the SeaSonde CSSY specification. The header section is processed recursively and terminates when one of the following conditions is met:

- A key with name "BODY" is encountered. In this case, the connection is rewound by 8 bytes to allow subsequent processing of the body.
- A key that is not defined in `current_specs` but is already present in the `keys_so_far` vector is encountered (indicative of repeated keys), which triggers termination.

Usage

```
seasonder_readCSSYHeader(
  connection,
  current_specs,
  endian = "big",
  parent_key = NULL,
  keys_so_far = c("CSSY", "HEAD"),
  specs_key_size = NULL
)
```

Arguments

`connection`	A binary connection from which to read the CSSY file header.
`current_specs`	A list representing the specification for the header; may contain nested subkeys.
`endian`	A character string indicating the byte order for reading numeric values ("big" or "little").
`parent_key`	(Optional) A list with information from the parent key block, used when processing nested keys.
`keys_so_far`	A character vector of keys already processed, used to avoid recursive loops. Defaults to c("CSSY", "HEAD").
`specs_key_size`	A specification for reading the key size block, often obtained from YAML specs.

Details

When no subkeys are specified in `current_specs` (i.e. `current_specs` comprises only simple field definitions), the function delegates the processing to `seasonder_readCSSYFields`.

The function processes the CSSY header recursively:

- If `current_specs` contains only field definitions, `seasonder_readCSSYFields` is called.

- When a key named "BODY" is encountered, it signifies the beginning of the body section; the function rewinds the connection 8 bytes and stops processing further keys.
- If a key is encountered that is not defined in `current_specs` but is already present in `keys_so_far`, the function also rewinds the connection 8 bytes and terminates header reading.
- Otherwise, the function updates `keys_so_far`, handles special cases (e.g., key "cs4h"), and calls itself recursively to process nested keys.

Value

A list containing the parsed CSSY header information. The returned list may be empty if a termination condition is encountered.

`seasonder_readPhaseFile` *Read Phase Correction File*

Description

This function reads a phase correction file and extracts phase correction values.

Usage

```
seasonder_readPhaseFile(file_path)
```

Arguments

`file_path`	The path to the phase correction file.

Value

A numeric vector with two elements: phase corrections for the two channels.

Examples

```
# Read phase corrections from sample file
phase_file <- system.file("css_data/Phases.txt", package = "SeaSondeR")
phase_corrections <- seasonder_readPhaseFile(phase_file)
```

`seasonder_readSeaSondeCSFile` *Read SeaSonde Cross Spectra (CS) File*

Description

This function reads and processes a SeaSonde CS file, extracting both its header and data.

Usage

```
seasonder_readSeaSondeCSFile(filepath, specs_path, endian = "big")
```

Arguments

`filepath`	A character string specifying the path to the SeaSonde CS file.
`specs_path`	A character string specifying the path to the YAML specifications for the CS file.
`endian`	Character string indicating the byte order. Options are "big" (default) or "little".

Details

The function starts by establishing a connection to the CS file specified by `filepath`. It then reads the necessary metadata and header specifications from the `specs_path`. Based on the CS file version determined from its header, it applies specific adjustments to the header data. After processing the header, the function validates the CS file data using `seasonder_validateCSFileData` and then reads the data itself via `seasonder_readSeaSondeCSFileData`.

Value

A list containing two components:

- `header`: A list containing the processed header information of the CS file.
- `data`: A list containing the processed data of the CS file. The structure of this list depends on the content of the CS file and can contain components such as `SSA*`, `CSxy`, and `QC`.

Condition Management

This function utilizes the `rlang` package to manage conditions and provide detailed and structured condition messages:

Condition Classes:

- `seasonder_read_cs_file_error`: An error class that indicates a general problem when attempting to read the SeaSonde CS file.
- `seasonder_cs_file_skipped`: Condition indicating that the processing of a CS file was skipped due to an error.

Condition Cases:

- Failure to open a connection to the file.
- Unsupported version found in the specs file.
- Any other error that can arise from dependent functions such as `seasonder_readSeaSondeCSFileHeader` and `seasonder_readSeaSondeCSFileData`.

Restart Options: This function provides a structured mechanism to recover from errors during its execution using the `rlang::withRestarts` function. The following restart option is available:

`seasonder_skip_cs_file(cond)` This allows for the graceful handling of file reading errors. If this restart is invoked, the function will log an error message indicating that the processing of a specific CS file was skipped and will return a list with `header = NULL` and `data = NULL`. The restart takes one argument: `cond` (the condition or error that occurred).

- **Usage**: In a custom condition handler, you can call `seasonder_skip_cs_file(cond)` to trigger this restart and skip the processing of the current CS file.
- **Effect**: If invoked, the function logs an error message detailing the reason for skipping the file and then returns a list with both the header and data set to NULL.

References

Cross Spectra File Format Version 6. CODAR. 2016

See Also

`seasonder_skip_cs_file`, `seasonder_validateCSFileData`, `seasonder_readSeaSondeCSFileHeader`, `seasonder_readSeaSondeCSFileData`, `seasonder_readYAMLSpecs`

Examples

```
spec_file <- seasonder_defaultSpecsFilePath("CS")
cs_file <- system.file("css_data/CSS_TORA_24_04_04_0700.cs", package = "SeaSondeR")
cs <- seasonder_readSeaSondeCSFile(cs_file, spec_file, endian = "big")
str(cs)
```

---

seasonder_readSeaSondeCSFileBlock

Read and Apply Quality Control to a Block of Fields

Description

Reads a block of fields from a binary file based on provided specifications. Each field is read and then processed with a specified quality control function.

Usage

```
seasonder_readSeaSondeCSFileBlock(spec, connection, endian = "big")
```

Arguments

spec	A named list of specifications for fields to read. Each specification should be in the form: list(type = "data_type", qc_fun = "qc_function_name", qc_params = list(param1 = value1, ...)) Where: • type: is the data type to read, which will be passed to seasonder_readCSField. • qc_fun: is the name of a quality control function. This function should be present in the shared environment seasonder_the and must accept field_value as its first argument, followed by any other arguments specified in qc_params. • qc_params: is a list of additional parameters to pass to the quality control function.
connection	A connection to the binary file.
endian	A character string indicating the byte order. Options are "big" and "little" (default is "big").

Details

The quality control (QC) functions (qc_fun) specified within spec play a pivotal role in ensuring the reliability of the data that's read. Here's the expected behavior of these QC functions:

- **Input**:
  - field_value: Value of the field that has been read from the binary file using the seasonder_readCSField function.
  - ...: Additional parameters specified in qc_params that are passed to qc_fun for quality control.
- **Functioning**: The QC function receives a read value and performs checks or transformations based on defined rules or parameters.
  - **On QC failure**:
    - The QC function itself is responsible for determining the action to take. It can log an error, return a default value, impute the value, and more.
    - For critical errors, the QC function could halt the execution. However, note that logging is managed by the QC function and won't necessarily halt execution in every case.
  - **On success**: The QC function will return the value (either unchanged or transformed).
- **Output**: Value that has been validated or transformed based on quality control rules.
- **Additional Notes**:
  - The action on QC failure is directly implemented within the QC function.
  - Reading errors are managed by the seasonder_readCSField function, which returns NULL in the case of an error. It is up to the QC function to decide what to do if it receives a NULL.

Value

A named list where each entry corresponds to a field that has been read. Each key is the field name, and its associated value is the data for that field after quality control.

See Also

read_and_qc_field

Examples

```
spec <- list(field1 = list(type = "UInt8", qc_fun = "qc_check_unsigned", qc_params = list()))
con <- rawConnection(as.raw(c(0x01)))
block <- seasonder_readSeaSondeCSFileBlock(spec, con, endian = "big")
print(block)
close(con)
```

---

seasonder_readSeaSondeCSFileData

Read SeaSonde Cross Spectra (CS) File Data

Description

This function reads the SeaSonde CS file data based on the provided header information. The CS file data includes the antenna voltage squared self spectra (SSA*) and the antenna cross spectra (CSxy). Additionally, a quality matrix (QC) is read when the header's nCsKind is greater than or equal to 2.

Usage

```
seasonder_readSeaSondeCSFileData(connection, header, endian = "big")
```

Arguments

connection	A connection object to the CS file.
header	A list containing the header information. This is typically the output of the seasonder_readSeaSondeCSFileHeader function.
endian	Character string indicating the byte order. Options are "big" (default) or "little".

Details

- SSA*: Represents the Antenna * voltage squared self spectra. These are matrices where each row corresponds to a range and each column to a Doppler cell.
- CSxy: Represents the cross spectra between two antennas x and y. These are complex matrices.
- QC: Quality matrix with values ranging from zero to one. A value less than one indicates that the SpectraAverager skipped some data during averaging.

Value

A list containing the processed CS file data including matrices for SSA*, CSxy, and QC (if applicable).

Condition Management

This function utilizes the `rlang` package to manage errors and conditions, providing detailed and structured messages:

Error Classes:

- `"seasonder_cs_data_reading_error"`: This error is thrown when there is a problem reading the CS file data. This could be due to issues with the connection object or the file itself.
- `"seasonder_cs_missing_header_info_error"`: Thrown if essential header information such as `nRangeCells`, `nDopplerCells`, or `nCsKind` is missing or invalid.

Error Cases:

- Connection object is not properly opened or is invalid.
- Header information is incomplete or improperly formatted.
- File read operations fail due to incorrect data size, type, or unexpected end of file.
- Non-numeric values encountered where numeric spectra data is expected.

Examples

```
con <- rawConnection(as.raw(rep(0, 300)))
header <- list(nRangeCells = 1, nDopplerCells = 5, nCsKind = 2)
data <- seasonder_readSeaSondeCSFileData(con, header, endian = "big")
print(data)
close(con)
```

`seasonder_readSeaSondeCSFileHeader`

Read the SeaSonde CS File Header

Description

This function reads and processes the header of a SeaSonde CS file. It initially reads the general header (Version 1) to determine the file version. Subsequent headers are processed based on the file version.

Usage

```
seasonder_readSeaSondeCSFileHeader(specs, connection, endian = "big")
```

Arguments

`specs`	List of header specifications for each version.
`connection`	The file connection.
`endian`	Character string indicating the byte order, either "big" (default) or "little".

Value

A combined list of all processed headers up to the file version.

See Also

seasonder_readSeaSondeCSFileHeaderV1 process_version_header

`seasonder_readSeaSondeCSFileHeaderV1` *Read SeaSonde File Header (Version 1)*

Description

Reads the header of a SeaSonde file (Version 1) based on the provided specifications. Transforms the date-time fields and returns the results.

Usage

```
seasonder_readSeaSondeCSFileHeaderV1(
  specs,
  connection,
  endian = "big",
  prev_data = NULL
)
```

Arguments

specs	A list containing specifications for reading the file.
connection	Connection object to the file.
endian	Character string specifying the endianness. Default is "big".
prev_data	previous header data

Value

A list with the read and transformed results.

See Also

seasonder_check_specs seasonder_readSeaSondeCSFileBlock

`seasonder_readSeaSondeCSFileHeaderV2` *Read SeaSonde File Header (Version 2)*

Description

Reads the header of a SeaSonde file (Version 2) based on the provided specifications.

Usage

```
seasonder_readSeaSondeCSFileHeaderV2(
  specs,
  connection,
  endian = "big",
  prev_data = NULL
)
```

Arguments

`specs`	A list containing specifications for reading the file.
`connection`	Connection object to the file.
`endian`	Character string specifying the endianness. Default is "big".
`prev_data`	previous header data

Value

A list with the read results.

See Also

`seasonder_check_specs` `seasonder_readSeaSondeCSFileBlock`

`seasonder_readSeaSondeCSFileHeaderV3` *Read SeaSonde File Header (Version 3)*

Description

Reads the header of a SeaSonde file (Version 3) based on the provided specifications. Adds nRangeCells, nDopplerCells, and nFirstRangeCell as constant values to the results.

Usage

```
seasonder_readSeaSondeCSFileHeaderV3(
  specs,
  connection,
  endian = "big",
  prev_data = NULL
)
```

Arguments

specs	A list containing specifications for reading the file.
connection	Connection object to the file.
endian	Character string specifying the endianness. Default is "big".
prev_data	previous header data

Value

A list with the read results.

See Also

seasonder_check_specs seasonder_readSeaSondeCSFileBlock

---

seasonder_readSeaSondeCSFileHeaderV4

Read SeaSonde File Header (Version 4)

Description

Reads the header of a SeaSonde file (Version 4) based on the provided specifications. Transforms the CenterFreq field and returns the results.

Usage

```
seasonder_readSeaSondeCSFileHeaderV4(
  specs,
  connection,
  endian = "big",
  prev_data = NULL
)
```

Arguments

specs	A list containing specifications for reading the file.
connection	Connection object to the file.
endian	Character string specifying the endianness. Default is "big".
prev_data	previous header data

Value

A list with the read and transformed results.

See Also

seasonder_check_specs seasonder_readSeaSondeCSFileBlock

`seasonder_readSeaSondeCSFileHeaderV5`
Read SeaSonde File Header (Version 5)

Description

Reads the header of a SeaSonde file (Version 5) based on the provided specifications. Performs applicable transformations and returns the results.

Usage

```
seasonder_readSeaSondeCSFileHeaderV5(
  specs,
  connection,
  endian = "big",
  prev_data = NULL
)
```

Arguments

specs	A list containing specifications for reading the file.
connection	Connection object to the file.
endian	Character string specifying the endianness. Default is "big".
prev_data	previous header data

Value

A list with the read and transformed results.

See Also

seasonder_check_specs seasonder_readSeaSondeCSFileBlock

`seasonder_readSeaSondeCSFileHeaderV6` *Read SeaSonde CS File Header V6*

Description

This function reads the header of a SeaSonde CS File Version 6. It sequentially reads blocks based on the provided specifications and returns the read data.

Usage

```
seasonder_readSeaSondeCSFileHeaderV6(
  specs,
  connection,
  endian = "big",
  prev_data = NULL
)
```

Arguments

`specs`	A list of specifications for reading the file header. It should contain three main elements: `nCS6ByteSize`, `block_spec`, and `blocks`, each containing further specifications for reading various parts of the header.
`connection`	A connection object to the SeaSonde CS file.
`endian`	The byte order for reading the file. Default is "big".
`prev_data`	Previous data, if any, that might affect the current reading. Default is NULL.

Value

A list containing the read data, organized based on the block keys.

Condition Management

This function utilizes the `rlang` package to manage conditions and provide detailed and structured condition messages:

Condition Classes:

- `seasonder_v6_block_transformacion_skipped`: Triggered when a transformation for a specific block is skipped.
- `seasonder_v6_transform_function_error`: Triggered when there's an error while applying the transformation function for a V6 header block.
- `seasonder_v6_skip_block_error`: Triggered when there's an error while skipping a block.

Condition Cases:

The following are the scenarios when errors or conditions are raised:

- Transformation Failure: If there's a recognized block key and the transformation function associated with it fails.
- Error in Transformation Function Application: If there's an error while applying the transformation function for a recognized V6 header block.
- Error in Skipping Block: If there's an error while skipping a block when the block key is not recognized.

Restart Options:

The function provides the following restart option:

`seasonder_v6_skip_transformation`: This restart allows users to skip the transformation for a specific block and instead return the provided value.

Effects of Restart Options:

Using the `seasonder_v6_skip_transformation` restart:

- The error message gets logged.
- The transformation that caused the error gets skipped.
- The provided value for that block is returned.

Proper error management ensures the integrity of the reading process and provides detailed feedback to users regarding issues and potential resolutions.

See Also

`seasonder_check_specs` `seasonder_readSeaSondeCSFileBlock` `readV6BlockData` `seasonder_v6_skip_transformat`

`seasonder_readSeaSondeRAPMFile`

Read and Parse a SeaSonde APM File

Description

This function reads a SeaSonde APM file and returns a SeaSondeRAPM object containing the parsed data.

Usage

```
seasonder_readSeaSondeRAPMFile(
  file_path,
  override_antenna_bearing = NULL,
  override_phase_corrections = NULL,
  override_amplitude_factors = NULL,
  override_SiteOrigin = NULL,
  ...
)
```

Arguments

`file_path`	The path to the SeaSonde APM file to read.
`override_antenna_bearing`	If not NULL, overrides the Antenna Bearing data in the file.
`override_phase_corrections`	If not NULL, overrides the phase corrections in the file.
`override_amplitude_factors`	If not NULL, overrides the amplitude factors in the file.
`override_SiteOrigin`	If not NULL, overrides the SiteOrigin attribute.
`...`	Additional arguments passed to the object creation function.

Value

A SeaSondeRAPM object containing the parsed data.

See Also

`seasonder_createSeaSondeRAPM`

`seasonder_validateAttributesSeaSondeRAPM`

Examples

```
# Read a test SeaSondeRAPM object from sample file
apm_file <- system.file("css_data/MeasPattern.txt", package = "SeaSondeR")
obj <- seasonder_readSeaSondeRAPMFile(apm_file)
```

---

seasonder_readSeaSondeRCSSWFile

Read SeaSonde RCSSW File and Create SeaSondeRCS Object

Description

This function reads a SeaSonde RCSSW file from a specified file path and parses its content into a SeaSondeRCS object. The file is processed by reading its header and body sections using CSSW specifications provided via a YAML file.

Usage

```
seasonder_readSeaSondeRCSSWFile(
  filepath,
  specs_path = seasonder_defaultSpecsFilePath("CSSW"),
  endian = "big"
)
```

Arguments

`filepath`	A character string specifying the path to the SeaSonde RCSSW file.
`specs_path`	A character string specifying the path to the YAML file containing CSSW specifications. Defaults to the output of `seasonder_defaultSpecsFilePath("CSSW")`.
`endian`	A character string indicating the byte order used in the file. Defaults to `"big"`.

Details

The function executes the following steps:

1. Sets up error handling parameters specific to the function.
2. Retrieves YAML specifications for the key size block from the CSSW spec file.
3. Attempts to open the file in binary mode ("rb") with warnings suppressed.
4. Reads the file key and uses it to extract file specs.
5. Reads the header key, retrieves header specs, and parses the CSSW header.
6. Converts the CSSW header into a valid SeaSondeRCS header.
7. Reads the body key, retrieves body specs, and parses the CSSW body.
8. Transforms the CSSW body into a SeaSondeRCS data structure.
9. Combines the header and data into a SeaSondeRCS object.

Value

A SeaSondeRCS object containing the parsed header and data.

`seasonder_readSeaSondeRCSSYFile` *Read SeaSonde RCSSY File and Create SeaSondeRCS Object*

Description

This function reads a SeaSonde RCSSY file from a specified file path and parses its content into a SeaSondeRCS object. The file is processed by reading its header and body sections using CSSY specifications provided via a YAML file.

Usage

```
seasonder_readSeaSondeRCSSYFile(
  filepath,
  specs_path = seasonder_defaultSpecsFilePath("CSSY"),
  endian = "big"
)
```

Arguments

filepath	A character string specifying the path to the SeaSonde RCSSY file.
specs_path	A character string specifying the path to the YAML file containing CSSY specifications. Defaults to the output of `seasonder_defaultSpecsFilePath("CSSY")`.
endian	A character string indicating the byte order used in the file. Defaults to `"big"`.

Details

The function executes the following steps:

1. Sets up error handling parameters specific to the function.
2. Retrieves YAML specifications for the key size block from the CSSY spec file.
3. Attempts to open the file in binary mode ("rb") with warnings suppressed.
4. Reads the file key and uses it to extract file specs.
5. Reads the header key, retrieves header specs, and parses the CSSY header.
6. Converts the CSSY header into a valid SeaSondeRCS header.
7. Reads the body key, retrieves body specs, and parses the CSSY body.
8. Transforms the CSSY body into a SeaSondeRCS data structure.
9. Combines the header and data into a SeaSondeRCS object.

Value

A SeaSondeRCS object containing the parsed header and data.

`seasonder_readYAMLSpecs`

Read Specifications from a YAML File

Description

This function reads a YAML file containing specifications, handles potential reading errors, and extracts specific information based on a provided path.

Usage

```
seasonder_readYAMLSpecs(file_path, path = rlang::zap())
```

Arguments

file_path	A string. The path to the YAML file.
path	A character vector. Represents the path within the YAML file to access the desired information. For example, to access fields of version V2 of the header, the path would be `c("header", "versions", "V2")`.

Details

This function provides built-in error handling which aborts execution and logs detailed error messages in case of:

- File not found.
- Error in reading the YAML content.
- The read YAML content is not a list.
- No data found for the provided path in the YAML content.

Errors generated are of class `"seasonder_read_yaml_file_error"`. For logging and aborting, this function uses `seasonder_logAndAbort`.

Value

A list. The information extracted from the YAML file based on the provided path.

See Also

`read_yaml` for the underlying YAML reading.

`pluck` for the data extraction mechanism used.

Examples

```
# Example: Read the CS header specifications (version V1) from the default specs file
specs_path <- seasonder_defaultSpecsFilePath("CS")
result <- seasonder_readYAMLSpecs(specs_path, c("header", "V1"))
str(result)
```

---

seasonder_read_reduced_encoded_data

Read Reduced Encoded Data from a Binary Connection

Description

This function reads an array of numbers from a binary connection using a custom command-based protocol. A block of data is processed according to its size specified in `key$size`. Within the block, the first byte read is a command byte that determines how the subsequent bytes are interpreted. The function updates a running "tracking value" based on the commands encountered and returns a vector of decoded numbers. The supported commands are:

Usage

```
seasonder_read_reduced_encoded_data(connection, key, endian = "big")
```

Arguments

connection	A binary connection from which the encoded data is read.
key	A list containing a field size that indicates how many bytes of data to process.
endian	A character string specifying the byte order; either "big" or "little". The default is "big".

Details

0x9C Read 4 bytes as an unsigned 32-bit integer.

0x94 Read one count byte, then (count+1) unsigned 32-bit integers.

0xAC Read 3 bytes as a 24-bit signed integer; add its value to the current tracking value.

0xA4 Read one count byte, then (count+1) 24-bit signed integers; sequentially add each to the tracking value.

0x89 Read 1 byte as a signed 8-bit integer; add it to the tracking value.

0x8A Read 2 bytes as a signed 16-bit integer; add it to the tracking value.

0x82 Read one count byte, then (count+1) signed 16-bit integers; sequentially add each to the tracking value.

0x81 Read one count byte, then (count+1) signed 8-bit integers; sequentially add each to the tracking value.

A 24-bit signed integer is computed by reading 3 bytes and then adjusting the value by subtracting 16777216 if the computed value is greater than or equal to 8388608 to account for the two's complement representation.

Value

An integer vector containing the decoded numbers.

---

seasonder_rejectDistantBragg

Apply Distant Bragg Peak Rejection to All Range Cells

Description

This function applies a proximity-based rejection test to all detected First Order Region (FOR) peaks in a SeaSondeRCS object. Peaks that are too far from their corresponding Bragg indices are removed, ensuring that only valid Bragg signals are retained.

Usage

```
seasonder_rejectDistantBragg(seasonder_cs_object)
```

Arguments

seasonder_cs_object	A SeaSondeRCS object containing the spectral data and FOR parameters.

Details

Reason for the Test: The function filters out peaks in the FOR that are not closely associated with expected Bragg indices. These distant peaks could result from interference, noise, or other non-Bragg sources, potentially leading to erroneous current velocity vectors. This ensures that only physically valid Bragg signals are used in the processing pipeline.

Steps:

1. **Retrieve Current FOR Data:**
   - Retrieves the detected FOR Doppler bin indices for all range cells using `seasonder_getSeaSondeRCS_FOR`.
2. **Apply Rejection Test:**
   - Iterates over each range cell and each Bragg region (`positive_FOR` and `negative_FOR`).
   - Calls `seasonder_rejectDistantBraggPeakTest` for each peak to evaluate its proximity to the Bragg indices.
   - Peaks that fail the test are removed (replaced with an empty vector).
3. **Store Updated FOR Data:**
   - Updates the `SeaSondeRCS` object with the filtered FOR results using `seasonder_setSeaSondeRCS_FOR`.

This function is part of the FOR processing workflow and should be applied after the initial detection of Bragg peaks.

Value

The updated `SeaSondeRCS` object with the filtered FOR bins.

References

COS. SpectraPlotterMap 12 User Guide. CODAR Ocean Sensors (COS), Mountain View, CA, USA, 2016.

See Also

- `seasonder_rejectDistantBraggPeakTest` for the peak rejection logic.
- `seasonder_getSeaSondeRCS_FOR` for retrieving detected FOR bins.
- `seasonder_setSeaSondeRCS_FOR` for updating FOR data.

`seasonder_rejectDistantBraggPeakTest`

Reject Bragg Peaks Far from Expected Bragg Index

Description

This function evaluates Bragg peaks based on their proximity to expected Bragg index bins. If the boundaries of a peak are farther from all Bragg indices than the width of the peak itself, the peak is rejected by returning an empty integer vector.

Usage

```
seasonder_rejectDistantBraggPeakTest(
  seasonder_cs_object,
  peak,
  range = NA,
  peak_name = ""
)
```

Arguments

seasonder_cs_object	A SeaSondeRCS object containing the spectral data and Bragg indices.
peak	A numeric vector indicating the Doppler bin positions of the peak under evaluation.
range	Optional; A numeric or integer value representing the range cell corresponding to the peak. Defaults to NA.
peak_name	Optional; A character string representing the name or identifier of the peak (e.g., "positive_FOR" or "negative_FOR"). Defaults to an empty string.

Details

Reason for the Test: The test ensures that peaks identified as part of the first-order Bragg region are reasonably close to the expected Bragg index. Peaks that are distant from the Bragg index are likely caused by noise, interference, or other sources unrelated to first-order Bragg scatter. These invalid peaks, if included, can lead to erroneous current velocity vectors and degrade the quality of radar-derived measurements.

Specifically, the test rejects peaks when:

- The distance from the left or right boundary of the peak to the nearest Bragg index exceeds the width of the peak.
- This condition ensures that peaks with excessively large offsets from the Bragg index are excluded.

Steps:

1. If the peak is empty, the function does nothing.
2. Calculates the width of the peak as the difference between its maximum and minimum Doppler bin indices.
3. Computes the left and right boundaries of the peak.
4. Calculates the distance from each boundary to all Bragg indices.
5. Rejects the peak if both boundary distances exceed the peak width.
6. Logs the rejection information if applicable.

This test is particularly important in scenarios where strong non-Bragg signals, such as those from ships or other high-intensity noise sources, might otherwise be misclassified as first-order Bragg.

Value

A possibly modified version of the peak argument, where a rejected peak is returned as `integer(0)`, indicating that it does not pass the proximity test.

References

COS. SpectraPlotterMap 12 User Guide. CODAR Ocean Sensors (COS), Mountain View, CA, USA, 2016.

See Also

- `seasonder_rejectDistantBragg` for applying this test to all range cells and Bragg regions.
- `seasonder_getBraggLineBins` for retrieving Bragg index bins.

`seasonder_rejectNoiseIonospheric` *Apply Noise/Ionospheric Contamination Test to All Bragg Peaks*

Description

This function evaluates and filters the First Order Region (FOR) detections across all range cells by applying the noise/ionospheric contamination rejection test to both positive and negative Bragg regions.

Usage

```
seasonder_rejectNoiseIonospheric(seasonder_cs_object)
```

Arguments

`seasonder_cs_object` A `SeaSondeRCS` object containing the spectral data and FOR parameters.

Details

Reason for the Test: This function ensures that Bragg peaks contaminated by excessive noise or ionospheric interference are removed from the detected First Order Region (FOR). Peaks where the power in the surrounding non-Bragg region exceeds the power in the Bragg region by a specified threshold are deemed invalid and are rejected. This step is critical for maintaining the accuracy of radar-derived measurements.

Steps:

1. **Retrieve Current FOR Data:**
   - Retrieves the detected FOR Doppler bin indices for all range cells using `seasonder_getSeaSondeRCS_FOR`.
2. **Apply Noise/Ionospheric Rejection Test:**
   - Iterates over each range cell and evaluates both positive and negative Bragg regions.

- Calls seasonder_rejectNoiseIonosphericTest to check each peak against the noise/ionospheric criterion.
- Peaks that fail the test are removed (replaced with an empty vector).

3. **Store Updated FOR Data:**
   - Updates the SeaSondeRCS object with the filtered FOR results using seasonder_setSeaSondeRCS_FOR.

Use Case: This function is particularly useful in environments where noise or ionospheric effects are prevalent, ensuring that only valid first-order Bragg peaks are retained for further processing.

Value

The updated SeaSondeRCS object with the filtered FOR bins.

See Also

- seasonder_rejectNoiseIonosphericTest for the noise/ionospheric rejection logic.
- seasonder_getSeaSondeRCS_FOR for retrieving detected FOR bins.
- seasonder_setSeaSondeRCS_FOR for updating FOR data.

---

seasonder_rejectNoiseIonosphericTest

Reject Bragg Peaks Due to Noise/Ionospheric Contamination

Description

This function evaluates Bragg peaks based on the power ratio between the Bragg region and the surrounding non-Bragg region. If the non-Bragg power exceeds the Bragg power by a specified threshold, the peak is rejected.

Usage

```
seasonder_rejectNoiseIonosphericTest(
  seasonder_cs_object,
  peak,
  range = NA,
  peak_name = ""
)
```

Arguments

seasonder_cs_object	A SeaSondeRCS object containing spectral data and FOR parameters.
peak	A numeric vector indicating the Doppler bin positions of the peak under evaluation.
range	Optional; A numeric or integer value representing the range cell corresponding to the peak. Defaults to NA.
peak_name	Optional; A character string representing the name or identifier of the peak (e.g., "positive_FOR" or "negative_FOR"). Defaults to an empty string.

Details

Reason for the Test: This test ensures that Bragg peaks are not contaminated by excessive noise or ionospheric interference. Bragg regions with significantly higher non-Bragg power levels are likely to be invalid and are rejected.

Steps:

1. **Threshold Retrieval:**
   - Retrieves the `reject_noise_ionospheric_threshold` parameter, which defines the power difference (in dB) allowed between the Bragg and non-Bragg regions.
2. **Peak Region Determination:**
   - Determines whether the peak is in the positive or negative Bragg region based on its location relative to the central Doppler bin.
3. **Non-Bragg Region Extraction:**
   - Identifies the non-Bragg region by excluding the bins corresponding to the peak.
4. **Power Calculations:**
   - Calculates the total power for the Bragg and non-Bragg regions and converts them to decibels (dB).
5. **Rejection Criterion:**
   - If the non-Bragg power exceeds the Bragg power by more than the threshold, the peak is rejected.
   - Logs a message detailing the rejection.

Use Case: This function is particularly useful in environments where noise or ionospheric effects are prevalent, ensuring that only valid first-order Bragg peaks are retained.

Value

A possibly modified version of the `peak` argument, where a rejected peak is returned as `integer(0)`, indicating that it does not pass the noise/ionospheric test.

See Also

- `seasonder_rejectDistantBraggPeakTest` for evaluating peaks based on proximity to Bragg indices.
- `seasonder_getSeaSondeRCS_FOR_SS_Smoothed` for retrieving smoothed spectra.
- `seasonder_SelfSpectra2dB` for power conversion to dB.

`seasonder_rerun_qc_with_fun` *Structured Restart for Quality Control*

Description

Provides a structured restart mechanism to rerun the quality control (QC) function with an alternative function during the execution of `read_and_qc_field`. This allows for a flexible error recovery strategy when the initial QC function fails or is deemed inadequate.

Usage

```
seasonder_rerun_qc_with_fun(cond, qc_fun)
```

Arguments

`cond`	The condition object captured during the execution of the `read_and_qc_field` function.
`qc_fun`	An alternate quality control function to apply. This function should accept the value from the field as its sole argument and return a QC-applied value.

Details

This function is meant to be used within custom condition handlers for the `read_and_qc_field` function.

Value

The value returned by the alternate quality control function.

Examples

```
# Example (expected to error due to missing restart):
val <- try(
  seasonder_rerun_qc_with_fun(
    list(seasonder_value = 42),
    function(x) x * 2
  ),
  silent = TRUE
)
print(val)
```

`seasonder_runMUSIC` *Execute the MUSIC Algorithm on a SeaSondeRCS Object*

Description

This function performs the MUSIC (MUltiple SIgnal Classification) algorithm on a given SeaSondeRCS object, executing a series of processing steps to extract direction-of-arrival (DOA) information and other related metrics from the radar cross-spectrum data.

Usage

```
seasonder_runMUSIC(seasonder_cs_object)
```

Arguments

`seasonder_cs_object`
: A `SeaSondeRCS` object that contains the radar cross-spectrum data and metadata. This object is modified in place to include the results of the MUSIC algorithm.

Details

The MUSIC algorithm is executed in a series of sequential steps:

1. Log the start of the MUSIC algorithm.
2. Update the processing steps of the `SeaSondeRCS` object to include the MUSIC start text.
3. Perform the following computations:
   - Compute the covariance matrix from the cross-spectrum data.
   - Perform eigen decomposition on the covariance matrix.
   - Compute the DOA functions using MUSIC-specific methods.
   - Extract peaks from the DOA functions, corresponding to possible signal directions.
   - Calculate the signal power matrix.
   - Test for dual solutions and compute their proportions.
   - Select the final set of DOAs from the computed data.
   - Convert the selected DOAs to geographical coordinates (latitude and longitude).
4. Log the completion of the MUSIC algorithm.

Value

A `SeaSondeRCS` object with updated MUSIC-related attributes. Specifically:

- Processing steps annotated with the MUSIC start and end points.
- Updated attributes and fields for covariance matrix computations, DOA estimations, and other MUSIC-related metrics.

See Also

seasonder_MUSICComputeCov: Compute the covariance matrix. seasonder_MUSICCovDecomposition: Perform eigen decomposition of the covariance matrix. seasonder_MUSICComputeDOAProjections: Compute the direction-of-arrival functions. seasonder_MUSICExtractPeaks: Extract peaks from the DOA functions. seasonder_MUSICComputeSignalPowerMatrix: Calculate the signal power matrix. seasonder_MUSICTestDualSolutions: Test and analyze dual solutions in the DOA. seasonder_MUSICComputePropDualSols: Compute proportions for dual solutions. seasonder_MUSICSelectDOA: Select final DOA estimations. seasonder_MUSIC_LonLat: Convert DOA estimations to geographical coordinates.

Examples

```
# Prepare a SeaSondeRCS object with MUSIC data
apm_file <- system.file("css_data/MeasPattern.txt", package = "SeaSondeR")
apm_obj <- seasonder_readSeaSondeRAPMFile(apm_file)
cs_file <- system.file("css_data/CSS_TORA_24_04_04_0700.cs", package = "SeaSondeR")
cs_obj <- seasonder_createSeaSondeRCS(cs_file, seasonder_apm_object = apm_obj)
cs_obj <- seasonder_initMUSICData(cs_obj)
cs_obj <- seasonder_initMUSICData(
 cs_obj,
 range_cells = c(rep(5,11), rep(4,11)),
 doppler_bins = c(c(669:679),c(674:684))
)
cs_obj <- seasonder_SeaSondeRCSMUSICInterpolateDoppler(cs_obj)
# Run the MUSIC algorithm
cs_obj <- seasonder_runMUSIC(cs_obj)
# Check the updated processing steps
print(seasonder_getSeaSondeRCS_ProcessingSteps(cs_obj))
```

---

seasonder_runMUSICInFOR

Run MUSIC Algorithm on FOR Data

Description

This function integrates the MUSIC (Multiple Signal Classification) algorithm into a SeaSondeRCS object that has First Order Regions (FOR) initialized. It first applies Doppler interpolation to the cross-spectra data, then extracts the FOR boundaries for each range cell by transforming the negative and positive FOR Doppler bins into frequency values and subsequently mapping these frequencies back to Doppler bins. Finally, the function initializes the MUSIC data structure and invokes the full MUSIC algorithm to update the SeaSondeRCS object.

Usage

```
seasonder_runMUSICInFOR(seasonder_cs_object)
```

Arguments

`seasonder_cs_object`
: A SeaSondeRCS object containing cross-spectra data and FOR information. (Note: Although this parameter is specified as an argument in the documentation, the actual Doppler interpolation factor is retrieved from the MUSIC options stored in the object.)

Details

This function performs the following sequence of operations:

1. It retrieves the Doppler interpolation factor from the MUSIC options of the input object.
2. It obtains the FOR data from the object using `seasonder_getSeaSondeRCS_FOR`.
3. For each range cell in the FOR data:
   (a) It processes the negative FOR bins by:
      i. Determining the frequency range corresponding to the negative bins via `seasonder_Bins2DopplerFreq`.
      ii. Mapping these frequencies to new Doppler bin indices with `seasonder_MUSIC_DopplerFreq2Bins` and adjusting the indices based on the interpolation factor.
   (b) It processes the positive FOR bins in an analogous manner.
   (c) If valid Doppler bin indices are obtained, a data frame is created recording the range cell and Doppler bin information.
4. The function compiles the extracted FOR information from all range cells into a single data frame.
5. It initializes the MUSIC data structure for the specified range cells and Doppler bins using `seasonder_initMUSICData`.
6. Finally, it calls `seasonder_runMUSIC` to execute the MUSIC algorithm on the updated object.

Value

A SeaSondeRCS object with its MUSIC data updated after applying Doppler interpolation, FOR extraction, and the complete MUSIC processing.

Examples

```
# Prepare a SeaSondeRCS object with MUSIC data (including FOR segments)
apm_file <- system.file("css_data/MeasPattern.txt", package = "SeaSondeR")
apm_obj <- seasonder_readSeaSondeRAPMFile(apm_file)
cs_file <- system.file("css_data/CSS_TORA_24_04_04_0700.cs", package = "SeaSondeR")
cs_obj <- seasonder_createSeaSondeRCS(cs_file, seasonder_apm_object = apm_obj)
FOR <- seasonder_getSeaSondeRCS_FOR(cs_obj)
cs_obj <- seasonder_setSeaSondeRCS_FOR(cs_obj,FOR[4:5])
# Run MUSIC algorithm in FOR context
result <- seasonder_runMUSICInFOR(cs_obj)
# View processing steps
print(seasonder_getSeaSondeRCS_ProcessingSteps(result))
```

`seasonder_SeaSondeRCSExportFORBoundaries` *Export First Order Region (FOR) Boundaries*

Description

This function exports the boundaries of the First Order Region (FOR) for each range cell from a `SeaSondeRCS` object, providing the first and last Doppler bins for both negative and positive Bragg regions.

Usage

```
seasonder_SeaSondeRCSExportFORBoundaries(seasonder_cs_object)
```

Arguments

`seasonder_cs_object`
: A `SeaSondeRCS` object containing the computed FOR data.

Details

Purpose: This function retrieves the computed FOR data from the `SeaSondeRCS` object and extracts the boundary Doppler bins for each range cell. The result is a data frame with the following columns:

- `range_cell`: The index of the range cell.
- `first_neg_doppler_cell`: The first Doppler bin in the negative Bragg region.
- `last_neg_doppler_cell`: The last Doppler bin in the negative Bragg region.
- `first_pos_doppler_cell`: The first Doppler bin in the positive Bragg region.
- `last_pos_doppler_cell`: The last Doppler bin in the positive Bragg region.

Steps:

1. Retrieve the FOR data using `seasonder_getSeaSondeRCS_FOR`.
2. Iterate through each range cell and extract the Doppler bins for both negative and positive Bragg regions.
3. Determine the range (first and last bins) for each region.
4. Combine the results into a single data frame, omitting empty entries.

Use Case: This function is useful for exporting the computed FOR boundaries to a format that can be further analyzed or visualized.

Value

A data frame with the boundaries of the FOR for each range cell.

See Also

- seasonder_getSeaSondeRCS_FOR for retrieving the FOR data.

Examples

```
# Set sample file paths
cs_file <- system.file("css_data/CSS_TORA_24_04_04_0700.cs", package = "SeaSondeR")
apm_file <- system.file("css_data/MeasPattern.txt", package = "SeaSondeR")
# Read the antenna pattern file to create a SeaSondeRAPM object
apm_obj <- seasonder_readSeaSondeRAPMFile(apm_file)
# Create a SeaSondeRCS object from a spectral file
cs_obj <- seasonder_createSeaSondeRCS(cs_file, seasonder_apm_object = apm_obj)
boundaries <- seasonder_SeaSondeRCSExportFORBoundaries(cs_obj)
head(boundaries)
```

---

seasonder_SeaSondeRCSMUSICInterpolateDoppler

Perform Doppler Interpolation for SeaSonde Cross-Spectra Data

---

Description

This function performs Doppler interpolation on the cross-spectra data of a SeaSondeRCS object, preparing the data for MUSIC processing. Interpolation is achieved by inserting additional Doppler bins using linear interpolation, potentially increasing the number of detected vectors while possibly smoothing the radials. The function tries to mimic CODAR's AnalyzeSpectra tool interpolation, including the addition of a wraparound Doppler cell before interpolation.

Usage

```
seasonder_SeaSondeRCSMUSICInterpolateDoppler(seasonder_cs_object)
```

Arguments

`seasonder_cs_object`
: A SeaSondeRCS object containing cross-spectra data and metadata for processing.

Details

Doppler interpolation increases the number of Doppler bins by a factor of 2, 3, or 4 before radial processing. This is accomplished by linearly interpolating between existing bins, increasing the number of radial vectors by approximately 15% for a 2x interpolation, and yielding smoother radials. The interpolation factor is configurable via the SeaSondeRCS object's `doppler_interpolation` attribute and it's setter `seasonder_setSeaSondeRCS_MUSIC_doppler_interpolation`. The number of Doppler bins after interpolation should not exceed 2048; exceeding this limit will result in an error.

The interpolation process is as follows:

1. A wraparound Doppler cell is added to the right of the data.
2. For non-quality-control (QC) matrices, linear interpolation is applied to fill in the newly added Doppler bins.
3. QC matrices are updated with a default value (-1) for interpolated bins.

Value

A `SeaSondeRCS` object with updated interpolated cross-spectra data and metadata.

Note

- CODAR's SeaSonde R8 Radial Config Setup documentation advises against using 3x or 4x interpolation.
- The function ensures the number of Doppler bins after interpolation does not exceed 2048.
- Doppler interpolation is a preprocessing step typically performed by CODAR's AnalyzeSpectra tool before MUSIC processing.

See Also

`seasonder_setSeaSondeRCS_MUSIC_interpolated_data` for setting interpolated data, `seasonder_getSeaSondeRCS_MUS` for retrieving the interpolation factor, `seasonder_setSeaSondeRCS_MUSIC_doppler_interpolation` for setting the interpolation factor, `seasonder_initCSDataStructure` for initializing the interpolated data structure.

Examples

```
# Doppler interpolation
# Create a SeaSondeRCS object for interpolation example
cs_file <- system.file("css_data/CSS_TORA_24_04_04_0700.cs", package = "SeaSondeR")
apm_file <- system.file("css_data/MeasPattern.txt", package = "SeaSondeR")
apm_obj <- seasonder_readSeaSondeRAPMFile(apm_file)
cs_obj <- seasonder_createSeaSondeRCS(cs_file, seasonder_apm_object = apm_obj)
# Perform Doppler interpolation
out <- seasonder_SeaSondeRCSMUSICInterpolateDoppler(cs_obj)
```

`seasonder_SeaSondeRCSSWApplyScaling`

Apply Scaling to SeaSondeRCSSW Data

Description

This function applies scaling to each vector of integer values contained in the list `values` by converting them to floating point voltage values using a specified scaling procedure. For each integer value:

- If the value equals 0xFFFFFFFF, it returns NaN;
- Otherwise, it computes an intermediate value using the formula: intermediate = value * (fmax - fmin) / fscale + fmin and then converts it to a voltage via: voltage = 10^((intermediate + dbRef) / 10)

Usage

```
seasonder_SeaSondeRCSSWApplyScaling(
  values,
  fmax,
  fmin,
  fscale,
  dbRef,
  computeVoltage = TRUE
)
```

Arguments

values	A list of numeric vectors containing integer values to be scaled. Each vector is expected to contain values read from a binary CSSW values block.
fmax	A numeric value representing the maximum scaling value. Used to compute the linear scaling factor.
fmin	A numeric value representing the minimum scaling value. Acts as an offset for the scaling.
fscale	A numeric value representing the scaling factor. Must not be zero as it determines the divisor in the scaling formula.
dbRef	A numeric value representing the decibel reference to be added before the voltage conversion step.
computeVoltage	A logical value indicating whether to compute the voltage from the scaled values. If FALSE, it returns the intermediate scaled values.

Details

The function processes each vector in the input list and returns a new list having the same structure, but with each value converted into its corresponding voltage value. It also performs several validations regarding input types and values.

The scaling process performs the following steps for each input value:

1. Checks whether the value equals 0xFFFFFFFF. If so, it returns NaN immediately because this value indicates a missing or invalid measurement.
2. Otherwise, it computes the intermediate scaled value by applying a linear transformation: intermediate = value * (fmax - fmin) / fscale + fmin
3. Finally, it converts the intermediate value to a voltage using: voltage = 10^((intermediate + dbRef) / 10)

The function includes input validation to ensure that `values` is a list, and that `fmax`, `fmin`, `fscale`, and `dbRef` are numeric. It also checks that no element in `values` is non-numeric and that `fscale` is non-zero to prevent division errors.

Value

A list with the same structure as `values`, where each numeric vector has been transformed to a vector of floating point voltage values. Special integer values equal to 0xFFFFFFFF are converted to NaN.

`seasonder_SeaSondeRCSSYApplyScaling` *Apply Scaling to SeaSondeRCSSY Data*

Description

This function applies scaling to each vector of integer values contained in the list `values` by converting them to floating point voltage values using a specified scaling procedure. For each integer value:

- If the value equals 0xFFFFFFFF, it returns NaN;
- Otherwise, it computes an intermediate value using the formula: intermediate = value * (fmax - fmin) / fscale + fmin and then converts it to a voltage via: voltage = 10^((intermediate + dbRef) / 10)

Usage

```
seasonder_SeaSondeRCSSYApplyScaling(values, fmax, fmin, fscale, dbRef)
```

Arguments

`values`	A list of numeric vectors containing integer values to be scaled. Each vector is expected to contain values read from a binary CSSY values block.
`fmax`	A numeric value representing the maximum scaling value. Used to compute the linear scaling factor.
`fmin`	A numeric value representing the minimum scaling value. Acts as an offset for the scaling.
`fscale`	A numeric value representing the scaling factor. Must not be zero as it determines the divisor in the scaling formula.
`dbRef`	A numeric value representing the decibel reference to be added before the voltage conversion step.

Details

The function processes each vector in the input list and returns a new list having the same structure, but with each value converted into its corresponding voltage value. It also performs several validations regarding input types and values.

The scaling process performs the following steps for each input value:

1. Checks whether the value equals 0xFFFFFFFF. If so, it returns NaN immediately because this value indicates a missing or invalid measurement.
2. Otherwise, it computes the intermediate scaled value by applying a linear transformation: intermediate = value * (fmax - fmin) / fscale + fmin
3. Finally, it converts the intermediate value to a voltage using: voltage = 10^((intermediate + dbRef) / 10)

The function includes input validation to ensure that `values` is a list, and that `fmax`, `fmin`, `fscale`, and `dbRef` are numeric. It also checks that no element in `values` is non-numeric and that `fscale` is non-zero to prevent division errors.

Value

A list with the same structure as `values`, where each numeric vector has been transformed to a vector of floating point voltage values. Special integer values equal to 0xFFFFFFFF are converted to NaN.

`seasonder_SeaSondeRCS_plotSelfSpectrum`
Plot Self-Spectrum for a SeaSondeRCS Object

Description

This function generates a plot of the self-spectrum (in dB) for a specified antenna and range cell from a SeaSondeRCS object. The Doppler frequencies are converted to the desired units before plotting. Optionally, it overlays additional elements such as smoothed self-spectrum lines, first-order region (FOR) vertical lines, and noise level lines.

Usage

```
seasonder_SeaSondeRCS_plotSelfSpectrum(
  seasonder_cs_object,
  antenna,
  range_cell,
  doppler_units = "normalized doppler frequency",
  plot_FORs = FALSE
)
```

Arguments

`seasonder_cs_object`	A SeaSondeRCS object containing spectral and metadata.
`antenna`	An integer or vector specifying the antenna(s) to extract the self-spectrum from.
`range_cell`	An integer indicating the range cell to extract the spectrum.
`doppler_units`	A character string specifying the desired Doppler units for the plot. Commonly "normalized doppler frequency" or "doppler frequency" (Hz). Default is "normalized doppler frequency".
`plot_FORs`	Logical. If `TRUE`, the function overlays elements related to the first order region (FOR) such as vertical lines at the FOR boundaries and the smoothed self-spectrum. Default is `FALSE`.

Details

The function performs the following steps:

1. Retrieves the self-spectrum data for the given antenna and range cell using `seasonder_getSeaSondeRCS_SelfSpectra`
2. Converts the Doppler bin frequencies to the specified units using `seasonder_SwapDopplerUnits`.
3. Converts the self-spectrum to dB using `seasonder_SelfSpectra2dB` and combines it with the Doppler values.
4. Retrieves the Bragg Doppler angular frequency for plotting a reference vertical line.
5. If `plot_FORs` is `TRUE`, overlays:
   - An orange line for the smoothed self-spectrum.
   - Blue vertical lines for FOR boundaries.
   - Red lines indicating the noise level across the Doppler spectrum.
6. Finally, returns the ggplot object.

Value

A ggplot object representing the self-spectrum plot.

Examples

```
# Prepare a SeaSondeRCS object for plotting self-spectrum
apm_file <- system.file("css_data/MeasPattern.txt", package = "SeaSondeR")
apm_obj <- seasonder_readSeaSondeRAPMFile(apm_file)
cs_file <- system.file("css_data/CSS_TORA_24_04_04_0700.cs", package = "SeaSondeR")
cs_obj <- seasonder_createSeaSondeRCS(cs_file, seasonder_apm_object = apm_obj)
# Plot self-spectrum for antenna 1, range cell 5
p <- seasonder_SeaSondeRCS_plotSelfSpectrum(cs_obj, antenna = 1, range_cell = 5)
print(p)
```

`seasonder_SelfSpectra2dB` *Convert Self-Spectra to dB Using a SeaSondeR Object*

Description

This function transforms self-spectra power values into decibels (dB) by retrieving the receiver gain from a given `SeaSondeR` object.

Usage

```
seasonder_SelfSpectra2dB(seasonder_cs_object, spectrum_values)
```

Arguments

`seasonder_cs_object`
: A SeaSondeR cross-spectral object.

`spectrum_values`
: A numeric vector. The power values in linear scale.

Details

This function first extracts the receiver gain in decibels from the `seasonder_cs_object` using `seasonder_getReceiverGain_dB` and then applies the conversion using:

$$dB = 10 \log_{10}(|P|) - G$$

where:

- \(dB \) is the power in decibels,
- \(P \) is the self-spectra power in linear scale,
- \(G \) is the receiver gain in decibels.

This function ensures consistency by obtaining the receiver gain directly from the SeaSondeR object.

Value

A numeric vector of power values in decibels (dB).

See Also

`self_spectra_to_dB` for a generic power-to-dB transformation.

`seasonder_setFORParameter` *Set a Specific FOR Parameter for a SeaSondeRCS Object*

Description

This function updates a single First Order Region (FOR) parameter in the SeaSondeRCS object. It creates a single-parameter list and passes it to `seasonder_setFOR_parameters()`.

Usage

```
seasonder_setFORParameter(seasonder_cs_object, FOR_parameter, value)
```

Arguments

`seasonder_cs_object`
: A SeaSondeRCS object containing FOR-related metadata.

`FOR_parameter`
: A character string specifying the name of the parameter to set.

`value`
: The value to assign to the specified FOR parameter.

Value

The updated SeaSondeRCS object with the new parameter value.

Examples

```
# Set sample file paths
cs_file <- system.file("css_data/CSS_TORA_24_04_04_0700.cs", package = "SeaSondeR")
apm_file <- system.file("css_data/MeasPattern.txt", package = "SeaSondeR")
# Read the antenna pattern file to create a SeaSondeRAPM object
apm_obj <- seasonder_readSeaSondeRAPMFile(apm_file)
# Create a SeaSondeRCS object from a spectral file
cs_obj <- seasonder_createSeaSondeRCS(cs_file, seasonder_apm_object = apm_obj)
cs_obj <- seasonder_setFORParameter(cs_obj, "nsm", 4)
```

---

seasonder_setFOR_currmax

Set FOR Maximum Velocity (currmax)

Description

This function sets the maximum radial velocity (`currmax`) allowed in the first-order region for the SeaSondeRCS object.

Usage

```
seasonder_setFOR_currmax(seasonder_cs_object, currmax)
```

Arguments

`seasonder_cs_object`	A SeaSondeRCS object with FOR parameters.
`currmax`	A numeric value specifying the new maximum radial velocity.

Value

The updated SeaSondeRCS object with the new `currmax` value.

Examples

```
# Set sample file paths
cs_file <- system.file("css_data/CSS_TORA_24_04_04_0700.cs", package = "SeaSondeR")
apm_file <- system.file("css_data/MeasPattern.txt", package = "SeaSondeR")
# Read the antenna pattern file to create a SeaSondeRAPM object
apm_obj <- seasonder_readSeaSondeRAPMFile(apm_file)
# Create a SeaSondeRCS object from a spectral file
cs_obj <- seasonder_createSeaSondeRCS(cs_file, seasonder_apm_object = apm_obj)
cs_obj <- seasonder_setFOR_currmax(cs_obj, 2.5)
```

seasonder_setFOR_fdown *Set FOR Dropoff Threshold (fdown)*

Description

This function sets the power dropoff threshold (`fdown`) used to define the lower limit of the first-order region.

Usage

```
seasonder_setFOR_fdown(seasonder_cs_object, fdown)
```

Arguments

`seasonder_cs_object`	A SeaSondeRCS object with FOR parameters.
`fdown`	A numeric value specifying the new dropoff threshold.

Value

The updated SeaSondeRCS object with the new `fdown` value.

Examples

```
# Set sample file paths
cs_file <- system.file("css_data/CSS_TORA_24_04_04_0700.cs", package = "SeaSondeR")
apm_file <- system.file("css_data/MeasPattern.txt", package = "SeaSondeR")
# Read the antenna pattern file to create a SeaSondeRAPM object
apm_obj <- seasonder_readSeaSondeRAPMFile(apm_file)
# Create a SeaSondeRCS object from a spectral file
cs_obj <- seasonder_createSeaSondeRCS(cs_file, seasonder_apm_object = apm_obj)
cs_obj <- seasonder_setFOR_fdown(cs_obj, 12)
```

seasonder_setFOR_flim *Set FOR Null Limit (flim)*

Description

This function sets the null limit (`flim`) used for defining the first-order region in the SeaSondeRCS object.

Usage

```
seasonder_setFOR_flim(seasonder_cs_object, flim)
```

Arguments

seasonder_cs_object	A SeaSondeRCS object with FOR parameters.
flim	A numeric value specifying the new null limit.

Value

The updated SeaSondeRCS object with the new flim value.

Examples

```
# Set sample file paths
cs_file <- system.file("css_data/CSS_TORA_24_04_04_0700.cs", package = "SeaSondeR")
apm_file <- system.file("css_data/MeasPattern.txt", package = "SeaSondeR")
# Read the antenna pattern file to create a SeaSondeRAPM object
apm_obj <- seasonder_readSeaSondeRAPMFile(apm_file)
# Create a SeaSondeRCS object from a spectral file
cs_obj <- seasonder_createSeaSondeRCS(cs_file, seasonder_apm_object = apm_obj)
cs_obj <- seasonder_setFOR_flim(cs_obj, 100)
```

---

seasonder_setFOR_noisefact

Set FOR Noise Factor (noisefact)

Description

This function sets the noise factor (noisefact) used in FOR processing for the given SeaSondeRCS object.

Usage

```
seasonder_setFOR_noisefact(seasonder_cs_object, noisefact)
```

Arguments

seasonder_cs_object	A SeaSondeRCS object with FOR parameters.
noisefact	A numeric value that specifies the noise factor.

Value

The updated SeaSondeRCS object with the new noisefact value.

Examples

```
# Set sample file paths
cs_file <- system.file("css_data/CSS_TORA_24_04_04_0700.cs", package = "SeaSondeR")
apm_file <- system.file("css_data/MeasPattern.txt", package = "SeaSondeR")
# Read the antenna pattern file to create a SeaSondeRAPM object
apm_obj <- seasonder_readSeaSondeRAPMFile(apm_file)
# Create a SeaSondeRCS object from a spectral file
cs_obj <- seasonder_createSeaSondeRCS(cs_file, seasonder_apm_object = apm_obj)
cs_obj <- seasonder_setFOR_noisefact(cs_obj, 4)
```

---

seasonder_setFOR_nsm *Set FOR Doppler Smoothing Factor (nsm)*

Description

This function sets the Doppler smoothing factor (nsm) in the SeaSondeRCS object for FOR processing.

Usage

```
seasonder_setFOR_nsm(seasonder_cs_object, nsm)
```

Arguments

seasonder_cs_object	A SeaSondeRCS object containing FOR-related information.
nsm	A numeric value specifying the new smoothing factor.

Value

The updated SeaSondeRCS object with the new nsm value.

Examples

```
# Set sample file paths
cs_file <- system.file("css_data/CSS_TORA_24_04_04_0700.cs", package = "SeaSondeR")
apm_file <- system.file("css_data/MeasPattern.txt", package = "SeaSondeR")
# Read the antenna pattern file to create a SeaSondeRAPM object
apm_obj <- seasonder_readSeaSondeRAPMFile(apm_file)
# Create a SeaSondeRCS object from a spectral file
cs_obj <- seasonder_createSeaSondeRCS(cs_file, seasonder_apm_object = apm_obj)
cs_obj <- seasonder_setFOR_nsm(cs_obj, 3)
```

`seasonder_setFOR_parameters` *Set First Order Region (FOR) Parameters for a SeaSondeRCS Object*

Description

This function validates and sets the FOR parameters in the SeaSondeRCS object. It updates the "FOR_data" attribute with validated parameters and, if the noise factor has changed, recomputes the noise level for all three antennas.

Usage

```
seasonder_setFOR_parameters(seasonder_cs_object, FOR_parameters)
```

Arguments

`seasonder_cs_object`	A SeaSondeRCS object containing FOR-related metadata.
`FOR_parameters`	A named list of parameters for first-order region detection.

Details

The provided parameters are validated using `seasonder_validateFOR_parameters()`, and then stored in the object's "FOR_data" attribute under `FOR_parameters`. If the `noisefact` parameter changes, the noise level is recomputed for antennas 1, 2, and 3.

Value

The updated SeaSondeRCS object with the new FOR parameters.

Examples

```
# Set sample file paths
cs_file <- system.file("css_data/CSS_TORA_24_04_04_0700.cs", package = "SeaSondeR")
apm_file <- system.file("css_data/MeasPattern.txt", package = "SeaSondeR")
# Read the antenna pattern file to create a SeaSondeRAPM object
apm_obj <- seasonder_readSeaSondeRAPMFile(apm_file)
# Create a SeaSondeRCS object from a spectral file
cs_obj <- seasonder_createSeaSondeRCS(cs_file, seasonder_apm_object = apm_obj)
new_params <- list(nsm = 3, noisefact = 4, fdown = 12, flim = 80, currmax = 2.5)
cs_obj <- seasonder_setFOR_parameters(cs_obj, new_params)
```

seasonder_setMUSICOption *Set a Specific MUSIC Option for a SeaSondeRCS Object*

Description

This function updates a single MUSIC option in the MUSIC data of a SeaSondeRCS object. It verifies that the provided option name is valid (i.e. exists in the default options), then updates that field with the new value.

Usage

```
seasonder_setMUSICOption(seasonder_cs_object, option_name, option_value)
```

Arguments

seasonder_cs_object	A SeaSondeRCS object that contains the MUSIC data as an attribute.
option_name	A character string specifying the name of the MUSIC option to update.
option_value	The new value to assign to the specified MUSIC option.

Details

The function first checks if `option_name` is one of the valid options as provided by `seasonder_defaultMUSICOptions()`. If not, it calls `seasonder_logAndAbort` to log an error. Then, the current MUSIC options are retrieved, updated with the new value, and stored back into the object.

Value

The updated SeaSondeRCS object with the specified MUSIC option modified.

Examples

```
# Example: set a specific MUSIC option on a minimal CS object
header <- list(nRangeCells = 1, nDopplerCells = 1)
data <- list(
  SSA1 = matrix(0,1,1), SSA2 = matrix(0,1,1), SSA3 = matrix(0,1,1),
  CS12 = matrix(complex(real=0,imaginary=0),1,1),
  CS13 = matrix(complex(real=0,imaginary=0),1,1),
  CS23 = matrix(complex(real=0,imaginary=0),1,1), QC = matrix(0,1,1)
)
apm_file <- system.file("css_data/MeasPattern.txt", package = "SeaSondeR")
apm_obj <- seasonder_readSeaSondeRAPMFile(apm_file)
cs_obj <- seasonder_createSeaSondeRCS(list(header = header, data = data),
seasonder_apm_object = apm_obj)
cs_obj <- seasonder_setMUSICOption(cs_obj, "smoothNoiseLevel", TRUE)
opts <- seasonder_getMUSICOptions(cs_obj)
print(opts$smoothNoiseLevel)
```

`seasonder_setMUSICOptions` *Set MUSIC Options for a SeaSondeRCS Object*

Description

This function updates the MUSIC options stored in a SeaSondeRCS object's MUSIC data attribute. It merges the provided options with the default MUSIC options, ensuring that any missing fields are filled with the defaults.

Usage

```
seasonder_setMUSICOptions(
  seasonder_cs_object,
  MUSIC_options = seasonder_defaultMUSICOptions()
)
```

Arguments

`seasonder_cs_object`	A SeaSondeRCS object that contains the MUSIC data as an attribute.
`MUSIC_options`	A named list of MUSIC options. Defaults to the output of `seasonder_defaultMUSICOptions()`.

Details

The function uses `modifyList` to merge the default MUSIC options with any user-specified options. This ensures that the resulting options list contains all required fields.

Value

The updated SeaSondeRCS object with the MUSIC options stored in its MUSIC data attribute.

Examples

```
# Example: update MUSIC options on a minimal CS object
header <- list(nRangeCells = 1, nDopplerCells = 1)
data <- list(
  SSA1 = matrix(0,1,1), SSA2 = matrix(0,1,1), SSA3 = matrix(0,1,1),
  CS12 = matrix(complex(real=0,imaginary=0),1,1),
  CS13 = matrix(complex(real=0,imaginary=0),1,1),
  CS23 = matrix(complex(real=0,imaginary=0),1,1), QC = matrix(0,1,1)
)
apm_file <- system.file("css_data/MeasPattern.txt", package = "SeaSondeR")
apm_obj <- seasonder_readSeaSondeRAPMFile(apm_file)
cs_obj <- seasonder_createSeaSondeRCS(list(header = header, data = data),
seasonder_apm_object = apm_obj)
cs_obj <- seasonder_setMUSICOptions(cs_obj, list(doppler_interpolation = 3))
opts <- seasonder_getMUSICOptions(cs_obj)
print(opts)
```

`seasonder_setNoiseLevelEstimationInterval` *Set Noise Level Estimation Interval for a SeaSondeRCS Object*

Description

This function sets the noise level estimation interval for a SeaSondeRCS object by updating the object's attribute and recalculating the reference noise normalized limits. It then updates the FOR parameters with the new noise limits.

Usage

```
seasonder_setNoiseLevelEstimationInterval(seasonder_cs_object, interval_value)
```

Arguments

`seasonder_cs_object`
: A SeaSondeRCS object.

`interval_value`
: A list containing the noise level estimation interval with two elements:
  - `low_limit`: A numeric value between 0 and 1 representing the lower limit.
  - `high_limit`: A numeric value between 0 and 1 representing the upper limit.

  The `low_limit` should be less than `high_limit`.

Details

The function updates the attribute `"reference_noise_normalized_limits_estimation_interval"` of the SeaSondeRCS object with `interval_value`. It then computes new reference noise normalized limits by calling `seasonder_estimateReferenceNoiseNormalizedLimits` with the provided lower and upper limits. Finally, it sets the new noise limits in the FOR parameters using `seasonder_setFORParameter`.

Value

The updated SeaSondeRCS object with the new noise level estimation interval and reference noise normalized limits.

Examples

```
new_interval <- list(low_limit = 0.9, high_limit = 1.0)
# Prepare a SeaSondeRCS object with valid data
apm_file <- system.file("css_data/MeasPattern.txt", package = "SeaSondeR")
apm_obj <- seasonder_readSeaSondeRAPMFile(apm_file)
cs_file <- system.file("css_data/CSS_TORA_24_04_04_0700.cs", package = "SeaSondeR")
cs_obj <- seasonder_createSeaSondeRCS(
  cs_file,
  specs_path = seasonder_defaultSpecsFilePath("CS"),
```

```
  seasonder_apm_object = apm_obj
)
cs_obj <- seasonder_setNoiseLevelEstimationInterval(cs_obj, new_interval)
print(attr(cs_obj, "reference_noise_normalized_limits_estimation_interval"))
noise_limits <- seasonder_getFOR_parameters(cs_obj)$reference_noise_normalized_limits
print(noise_limits)
```

`seasonder_setSeaSondeRAPM_AmplitudeFactors` *Setter for AmplitudeFactors*

Description

Setter for AmplitudeFactors

Usage

```
seasonder_setSeaSondeRAPM_AmplitudeFactors(seasonde_apm_obj, new_value)
```

Arguments

`seasonde_apm_obj`	SeaSonderAPM object
`new_value`	new value

Value

The modified SeaSondeRAPM object with updated AmplitudeFactors.

Examples

```
# Minimal example for seasonder_setSeaSondeRAPM_AmplitudeFactors
apm_file <- system.file("css_data/MeasPattern.txt", package = "SeaSondeR")
apm_obj <- seasonder_readSeaSondeRAPMFile(apm_file)
new_factors <- c(1, 2)
apm_obj <- seasonder_setSeaSondeRAPM_AmplitudeFactors(apm_obj, new_factors)
print(attributes(apm_obj)$AmplitudeFactors)
```

seasonder_setSeaSondeRAPM_AntennaBearing *Setter for AntennaBearing*

Description

Setter for AntennaBearing

Usage

```
seasonder_setSeaSondeRAPM_AntennaBearing(seasonde_apm_obj, new_value)
```

Arguments

seasonde_apm_obj	SeaSonderAPM object
new_value	new value

Value

The modified SeaSondeRAPM object with updated AntennaBearing.

Examples

```
# Minimal example for seasonder_setSeaSondeRAPM_AntennaBearing
apm_file <- system.file("css_data/MeasPattern.txt", package = "SeaSondeR")
apm_obj <- seasonder_readSeaSondeRAPMFile(apm_file)
new_bearing <- 45
apm_obj <- seasonder_setSeaSondeRAPM_AntennaBearing(apm_obj, new_bearing)
print(attributes(apm_obj)$AntennaBearing)
```

seasonder_setSeaSondeRAPM_BEAR *Setter for BEAR*

Description

Setter for BEAR

Usage

```
seasonder_setSeaSondeRAPM_BEAR(seasonde_apm_obj, new_value)
```

Arguments

seasonde_apm_obj	SeaSonderAPM object
new_value	new value

Value

The modified SeaSondeRAPM object with updated BEAR.

Examples

```
# Minimal example for seasonder_setSeaSondeRAPM_BEAR
apm_file <- system.file("css_data/MeasPattern.txt", package = "SeaSondeR")
apm_obj <- seasonder_readSeaSondeRAPMFile(apm_file)
new_bear <- attributes(apm_obj)$BEAR
apm_obj <- seasonder_setSeaSondeRAPM_BEAR(apm_obj, new_bear)
print(attributes(apm_obj)$BEAR)
```

---

seasonder_setSeaSondeRAPM_BearingResolution

Setter for BearingResolution

Description

Setter for BearingResolution

Usage

```
seasonder_setSeaSondeRAPM_BearingResolution(seasonde_apm_obj, new_value)
```

Arguments

seasonde_apm_obj	SeaSonderAPM object
new_value	new value

Value

The modified SeaSondeRAPM object with updated BearingResolution.

Examples

```
# Minimal example for seasonder_setSeaSondeRAPM_BearingResolution
apm_file <- system.file("css_data/MeasPattern.txt", package = "SeaSondeR")
apm_obj <- seasonder_readSeaSondeRAPMFile(apm_file)
new_bearing_resolution <- 1.0
apm_obj <- seasonder_setSeaSondeRAPM_BearingResolution(apm_obj, new_bearing_resolution)
print(attributes(apm_obj)$BearingResolution)
```

seasonder_setSeaSondeRAPM_CommentLine *Setter for CommentLine*

Description

Setter for CommentLine

Usage

```
seasonder_setSeaSondeRAPM_CommentLine(seasonde_apm_obj, new_value)
```

Arguments

seasonde_apm_obj	SeaSonderAPM object
new_value	new value

Value

The modified SeaSondeRAPM object with updated CommentLine.

Examples

```
apm_file <- system.file("css_data/MeasPattern.txt", package = "SeaSondeR")
apm_obj <- seasonder_readSeaSondeRAPMFile(apm_file)
new_comment_line <- "Test comment"
apm_obj <- seasonder_setSeaSondeRAPM_CommentLine(apm_obj, new_comment_line)
print(attributes(apm_obj)$CommentLine)
```

seasonder_setSeaSondeRAPM_CreateTimeStamp *Setter for CreateTimeStamp*

Description

Setter for CreateTimeStamp

Usage

```
seasonder_setSeaSondeRAPM_CreateTimeStamp(seasonde_apm_obj, new_value)
```

Arguments

seasonde_apm_obj	SeaSonderAPM object
new_value	new value

Value

The modified SeaSondeRAPM object with updated CreateTimeStamp.

Examples

```
apm_file <- system.file("css_data/MeasPattern.txt", package = "SeaSondeR")
apm_obj <- seasonder_readSeaSondeRAPMFile(apm_file)
new_create_time_stamp <- as.POSIXct("2000-01-01 00:00:00", tz = "UTC")
apm_obj <- seasonder_setSeaSondeRAPM_CreateTimeStamp(apm_obj, new_create_time_stamp)
print(attributes(apm_obj)$CreateTimeStamp)
```

---

seasonder_setSeaSondeRAPM_Creator

Setter for Creator

Description

Setter for Creator

Usage

```
seasonder_setSeaSondeRAPM_Creator(seasonde_apm_obj, new_value)
```

Arguments

seasonde_apm_obj	SeaSonderAPM object
new_value	new value

Value

The modified SeaSondeRAPM object with updated Creator.

Examples

```
# Minimal example for seasonder_setSeaSondeRAPM_Creator
apm_file <- system.file("css_data/MeasPattern.txt", package = "SeaSondeR")
apm_obj <- seasonder_readSeaSondeRAPMFile(apm_file)
new_creator <- attributes(apm_obj)$Creator
apm_obj <- seasonder_setSeaSondeRAPM_Creator(apm_obj, new_creator)
print(attributes(apm_obj)$Creator)
```

seasonder_setSeaSondeRAPM_FileID *Setter for FileID*

Description

Setter for FileID

Usage

```
seasonder_setSeaSondeRAPM_FileID(seasonde_apm_obj, new_value)
```

Arguments

seasonde_apm_obj	SeaSonderAPM object
new_value	new value

Value

The modified SeaSondeRAPM object with updated FileID.

Examples

```
apm_file <- system.file("css_data/MeasPattern.txt", package = "SeaSondeR")
apm_obj <- seasonder_readSeaSondeRAPMFile(apm_file)
new_file_id <- attributes(apm_obj)$FileID
apm_obj <- seasonder_setSeaSondeRAPM_FileID(apm_obj, new_file_id)
print(attributes(apm_obj)$FileID)
```

seasonder_setSeaSondeRAPM_FileName *Setter for FileName*

Description

Setter for FileName

Usage

```
seasonder_setSeaSondeRAPM_FileName(seasonde_apm_obj, new_value)
```

Arguments

seasonde_apm_obj	SeaSonderAPM object
new_value	new value

Value

The modified SeaSondeRAPM object with updated FileName.

Examples

```
# Minimal example for seasonder_setSeaSondeRAPM_FileName
apm_file <- system.file("css_data/MeasPattern.txt", package = "SeaSondeR")
apm_obj <- seasonder_readSeaSondeRAPMFile(apm_file)
new_file_name <- "new.txt"
apm_obj <- seasonder_setSeaSondeRAPM_FileName(apm_obj, new_file_name)
print(attributes(apm_obj)$FileName)
```

---

seasonder_setSeaSondeRAPM_PhaseCorrections

Setter for PhaseCorrections

Description

Setter for PhaseCorrections

Usage

```
seasonder_setSeaSondeRAPM_PhaseCorrections(seasonde_apm_obj, new_value)
```

Arguments

seasonde_apm_obj	SeaSonderAPM object
new_value	new value

Value

The modified SeaSondeRAPM object with updated PhaseCorrections.

Examples

```
# Minimal example for seasonder_setSeaSondeRAPM_PhaseCorrections
apm_file <- system.file("css_data/MeasPattern.txt", package = "SeaSondeR")
apm_obj <- seasonder_readSeaSondeRAPMFile(apm_file)
new_phase_corrections <- attributes(apm_obj)$PhaseCorrections
apm_obj <- seasonder_setSeaSondeRAPM_PhaseCorrections(apm_obj, new_phase_corrections)
print(attributes(apm_obj)$PhaseCorrections)
```

seasonder_setSeaSondeRAPM_ProcessingSteps *Setter for ProcessingSteps*

Description

Setter for ProcessingSteps

Usage

```
seasonder_setSeaSondeRAPM_ProcessingSteps(
  seasonde_apm_obj,
  new_value,
  append = TRUE
)
```

Arguments

seasonde_apm_obj	SeaSonderAPM object
new_value	new value
append	Append the new step to existing steps if TRUE; otherwise, replace previous steps.

Value

The modified SeaSondeRAPM object with updated ProcessingSteps.

Examples

```
# Minimal example for seasonder_setSeaSondeRAPM_ProcessingSteps
apm_file <- system.file("css_data/MeasPattern.txt", package = "SeaSondeR")
apm_obj <- seasonder_readSeaSondeRAPMFile(apm_file)
new_processing_steps <- "step1"
apm_obj <- seasonder_setSeaSondeRAPM_ProcessingSteps(apm_obj, new_processing_steps)
print(attributes(apm_obj)$ProcessingSteps)
```

seasonder_setSeaSondeRAPM_quality_matrix *Setter for quality_matrix*

Description

Setter for quality_matrix

Usage

```
seasonder_setSeaSondeRAPM_quality_matrix(seasonde_apm_obj, new_value)
```

Arguments

seasonde_apm_obj	SeaSonderAPM object
new_value	new value

Value

The modified SeaSondeRAPM object with updated quality_matrix.

Examples

```
# Create a default SeaSondeRAPM object
obj <- seasonder_createSeaSondeRAPM()
# Retrieve the existing quality_matrix
new_quality_matrix <- attributes(obj)$quality_matrix
# Update quality_matrix in the object
obj <- seasonder_setSeaSondeRAPM_quality_matrix(obj, new_quality_matrix)
```

---

seasonder_setSeaSondeRAPM_SiteName

Setter for SiteName

Description

Setter for SiteName

Usage

```
seasonder_setSeaSondeRAPM_SiteName(seasonde_apm_obj, new_value)
```

Arguments

seasonde_apm_obj	SeaSonderAPM object
new_value	new value

Value

The modified SeaSondeRAPM object with updated SiteName.

Examples

```
# Minimal example for seasonder_setSeaSondeRAPM_SiteName
apm_file <- system.file("css_data/MeasPattern.txt", package = "SeaSondeR")
apm_obj <- seasonder_readSeaSondeRAPMFile(apm_file)
new_site_name <- attributes(apm_obj)$SiteName
apm_obj <- seasonder_setSeaSondeRAPM_SiteName(apm_obj, new_site_name)
print(attributes(apm_obj)$SiteName)
```

---

seasonder_setSeaSondeRAPM_SiteOrigin

Setter for SiteOrigin

Description

Setter for SiteOrigin

Usage

```
seasonder_setSeaSondeRAPM_SiteOrigin(seasonde_apm_obj, new_value)
```

Arguments

seasonde_apm_obj	SeaSonderAPM object
new_value	new value

Value

The modified SeaSondeRAPM object with updated SiteOrigin.

Examples

```
apm_file <- system.file("css_data/MeasPattern.txt", package = "SeaSondeR")
apm_obj <- seasonder_readSeaSondeRAPMFile(apm_file)
new_site_origin <- attributes(apm_obj)$SiteOrigin
apm_obj <- seasonder_setSeaSondeRAPM_SiteOrigin(apm_obj, new_site_origin)
print(attributes(apm_obj)$SiteOrigin)
```

seasonder_setSeaSondeRAPM_Smoothing *Setter for Smoothing*

Description

Setter for Smoothing

Usage

```
seasonder_setSeaSondeRAPM_Smoothing(seasonde_apm_obj, new_value)
```

Arguments

seasonde_apm_obj	SeaSonderAPM object
new_value	new value

Value

The modified SeaSondeRAPM object with updated Smoothing.

Examples

```
# Minimal example for seasonder_setSeaSondeRAPM_Smoothing
apm_file <- system.file("css_data/MeasPattern.txt", package = "SeaSondeR")
apm_obj <- seasonder_readSeaSondeRAPMFile(apm_file)
new_smoothing <- 3
apm_obj <- seasonder_setSeaSondeRAPM_Smoothing(apm_obj, new_smoothing)
print(attributes(apm_obj)$Smoothing)
```

seasonder_setSeaSondeRAPM_StationCode *Setter for StationCode*

Description

Setter for StationCode

Usage

```
seasonder_setSeaSondeRAPM_StationCode(seasonde_apm_obj, new_value)
```

Arguments

seasonde_apm_obj	SeaSonderAPM object
new_value	new value

Value

The modified SeaSondeRAPM object with updated StationCode.

Examples

```
# Minimal example for seasonder_setSeaSondeRAPM_StationCode
apm_file <- system.file("css_data/MeasPattern.txt", package = "SeaSondeR")
apm_obj <- seasonder_readSeaSondeRAPMFile(apm_file)
new_station_code <- attr(apm_obj, "StationCode")
apm_obj <- seasonder_setSeaSondeRAPM_StationCode(apm_obj, new_station_code)
print(attributes(apm_obj)$StationCode)
```

---

seasonder_setSeaSondeRAPM_Type

Setter for Type

Description

Setter for Type

Usage

```
seasonder_setSeaSondeRAPM_Type(seasonde_apm_obj, new_value)
```

Arguments

seasonde_apm_obj	SeaSonderAPM object
new_value	new value

Value

The modified SeaSondeRAPM object with updated Type.

Examples

```
# Minimal example for seasonder_setSeaSondeRAPM_Type
apm_file <- system.file("css_data/MeasPattern.txt", package = "SeaSondeR")
apm_obj <- seasonder_readSeaSondeRAPMFile(apm_file)
new_type <- attributes(apm_obj)$Type
apm_obj <- seasonder_setSeaSondeRAPM_Type(apm_obj, new_type)
print(attributes(apm_obj)$Type)
```

seasonder_setSeaSondeRCS_APM *Set APM for a SeaSondeRCS Object*

Description

This function assigns the provided APM object to the SeaSondeRCS object by setting its "APM" attribute. (Note: Validation of the APM object is to be implemented.)

Usage

```
seasonder_setSeaSondeRCS_APM(seasonder_cs_object, seasonder_apm_object)
```

Arguments

seasonder_cs_object	A SeaSondeRCS object.
seasonder_apm_object	An object representing the APM (Antenna Pattern Matrix or similar metadata) to be assigned to the SeaSondeRCS object.

Details

The function simply sets the "APM" attribute of the provided SeaSondeRCS object to the given APM object. Further validation of the APM object should be performed (TODO).

Value

The updated SeaSondeRCS object with the new APM attribute set.

Examples

```
# Minimal example for seasonder_setSeaSondeRCS_APM
cs_file <- system.file("css_data/CSS_TORA_24_04_04_0700.cs", package = "SeaSondeR")
apm_file <- system.file("css_data/MeasPattern.txt", package = "SeaSondeR")
apm_obj <- seasonder_readSeaSondeRAPMFile(apm_file)
cs_obj <- seasonder_createSeaSondeRCS(cs_file, seasonder_apm_object = apm_obj)
cs_obj <- seasonder_setSeaSondeRCS_APM(cs_obj, apm_obj)
print(attr(cs_obj, "APM"))
```

seasonder_setSeaSondeRCS_data

Setter for data

Description

Setter for data

Usage

```
seasonder_setSeaSondeRCS_data(seasonder_cs_object, data)
```

Arguments

seasonder_cs_object	SeaSondeRCS object
data	new value

Value

A SeaSondeRCS object with updated data.

See Also

seasonder_validateCSDataStructure

Examples

```
# Minimal example for seasonder_setSeaSondeRCS_data
cs_file <- system.file("css_data/CSS_TORA_24_04_04_0700.cs", package = "SeaSondeR")
apm_file <- system.file("css_data/MeasPattern.txt", package = "SeaSondeR")
apm_obj <- seasonder_readSeaSondeRAPMFile(apm_file)
cs_obj <- seasonder_createSeaSondeRCS(cs_file, seasonder_apm_object = apm_obj)
new_data <- seasonder_getSeaSondeRCS_data(cs_obj)
cs_obj <- seasonder_setSeaSondeRCS_data(cs_obj, new_data)
str(seasonder_getSeaSondeRCS_data(cs_obj))
```

seasonder_setSeaSondeRCS_FOR

Set First Order Region Data in a SeaSondeRCS Object

Description

This function assigns First Order Region (FOR) data to a SeaSondeRCS object. The FOR data is stored within the object's attributes under the "FOR_data" element. Currently, no explicit validation is performed on the provided FOR data.

Usage

```
seasonder_setSeaSondeRCS_FOR(seasonder_cs_object, FOR)
```

Arguments

seasonder_cs_object	A SeaSondeRCS object containing spectral and metadata information.
FOR	A data structure containing the First Order Region (FOR) data. This is typically a list with elements such as negative_FOR and positive_FOR representing Doppler bin indices.

Details

This low-level setter function updates the SeaSondeRCS object by assigning the provided FOR data to the "FOR" field within the object's "FOR_data" attribute. It is intended to be used internally as part of the FOR processing workflow.

Value

The updated SeaSondeRCS object with the specified FOR data stored in its attributes.

Examples

```
# Set sample file paths
cs_file <- system.file("css_data/CSS_TORA_24_04_04_0700.cs", package = "SeaSondeR")
apm_file <- system.file("css_data/MeasPattern.txt", package = "SeaSondeR")
# Read the antenna pattern file to create a SeaSondeRAPM object
apm_obj <- seasonder_readSeaSondeRAPMFile(apm_file)
# Create a SeaSondeRCS object from a spectral file
cs_obj <- seasonder_createSeaSondeRCS(cs_file, seasonder_apm_object = apm_obj)
sample_FOR <- list(
  negative_FOR = c(1, 2, 3),
  positive_FOR = c(10, 11, 12)
)
cs_obj <- seasonder_setSeaSondeRCS_FOR(cs_obj, sample_FOR)
```

---

seasonder_setSeaSondeRCS_FOR_MAXP

Set Maximum Power (MAXP) for First Order Region (FOR)

Description

This function assigns the computed maximum power values (MAXP) for each range cell in the First Order Region (FOR) to the SeaSondeRCS object.

Usage

```
seasonder_setSeaSondeRCS_FOR_MAXP(seasonder_cs_object, FOR_MAXP)
```

Arguments

seasonder_cs_object	A SeaSondeRCS object to which the MAXP values will be assigned.
FOR_MAXP	A list containing the maximum power values for each range cell.

Details

The maximum power (MAXP) represents the highest spectral power detected in the first-order region. This value is extracted from the self-spectra and used for setting first-order boundaries.

Validation Considerations:

- The function does not currently perform explicit validation on FOR_MAXP.
- Future improvements should ensure that FOR_MAXP contains numeric values corresponding to each range cell.

Value

The updated SeaSondeRCS object with the MAXP values stored in the FOR_data attribute.

See Also

- seasonder_findFORNulls for computing MAXP.
- seasonder_setSeaSondeRCS_FOR_MAXP.bin for setting maximum power bin indices.

---

seasonder_setSeaSondeRCS_FOR_MAXP.bin

Set Maximum Power Bin Indices for First Order Region (FOR)

Description

This function assigns the Doppler bin indices corresponding to the maximum power (MAXP.bin) for each range cell in the First Order Region (FOR) to the SeaSondeRCS object.

Usage

```
seasonder_setSeaSondeRCS_FOR_MAXP.bin(seasonder_cs_object, FOR_MAXP.bin)
```

Arguments

seasonder_cs_object	A SeaSondeRCS object to which the MAXP.bin values will be assigned.
FOR_MAXP.bin	A list containing the Doppler bin indices of the maximum power for each range cell.

Details

The maximum power bin (MAXP.bin) represents the Doppler bin index at which the highest spectral power was detected in the first-order region. This information is used to refine first-order boundary detection.

Validation Considerations:

- The function does not currently validate the format of FOR_MAXP.bin.
- Future improvements should ensure that FOR_MAXP.bin consists of integer values corresponding to Doppler bins.

Value

The updated SeaSondeRCS object with the MAXP.bin values stored in the FOR_data attribute.

See Also

- seasonder_findFORNulls for computing MAXP.bin.
- seasonder_setSeaSondeRCS_FOR_MAXP for setting maximum power values.

---

seasonder_setSeaSondeRCS_FOR_method

Set First Order Region Processing Method for SeaSondeRCS Object

---

Description

This function sets the First Order Region (FOR) processing method for a SeaSondeRCS object. It validates the provided method using seasonder_validateFORMethod and assigns it to the object's FOR_data attribute under FOR_method.

Usage

```
seasonder_setSeaSondeRCS_FOR_method(seasonder_cs_object, FOR_method)
```

Arguments

seasonder_cs_object	A SeaSondeRCS object containing spectral and metadata information.
FOR_method	A character string specifying the desired FOR processing method. Currently, only "SeaSonde" is supported.

Details

The function first validates the provided method. If the method is valid, it is stored in the FOR_data attribute of the SeaSondeRCS object under the FOR_method field. This setting is later used in the processing workflow to guide FOR computation.

Value

The updated SeaSondeRCS object with the specified FOR processing method set.

`seasonder_setSeaSondeRCS_FOR_SS_Smoothed`
Set Smoothed Self-Spectra for First Order Region (FOR)

Description

This function assigns a smoothed self-spectra (SS) matrix to the First Order Region (FOR) data within a `SeaSondeRCS` object. This smoothed matrix is used in FOR processing to improve the detection of the first-order region.

Usage

```
seasonder_setSeaSondeRCS_FOR_SS_Smoothed(seasonder_cs_object, FOR_SS_Smoothed)
```

Arguments

`seasonder_cs_object`
: A `SeaSondeRCS` object to which the smoothed FOR self-spectra will be assigned.

`FOR_SS_Smoothed`
: A matrix containing the smoothed self-spectra data.

Details

The function assigns the provided smoothed self-spectra matrix to the `FOR_data` attribute of the `SeaSondeRCS` object. This matrix is typically generated using `seasonder_SmoothSS` and applied to antenna 3.

Validation Considerations:

- The function currently lacks explicit validation for `FOR_SS_Smoothed`.
- Future improvements should include checking whether `FOR_SS_Smoothed` is a matrix and ensuring its dimensions match the original self-spectra structure.

Value

The updated `SeaSondeRCS` object with the smoothed self-spectra stored in the `FOR_data` attribute.

See Also

- `seasonder_SmoothSS` for generating the smoothed self-spectra.
- `seasonder_SmoothFORSS` for applying smoothing and setting the result.

`seasonder_setSeaSondeRCS_header` *Setter for header*

Description

Setter for header

Usage

```
seasonder_setSeaSondeRCS_header(seasonder_cs_object, header)
```

Arguments

`seasonder_cs_object`	SeaSondeRCS object
`header`	new value

Value

A SeaSondeRCS object with updated header.

See Also

`seasonder_validateCSHeaderStructure`

Examples

```
# Set sample file paths and create SeaSondeRCS object
cs_file <- system.file("css_data/CSS_TORA_24_04_04_0700.cs", package = "SeaSondeR")
apm_file <- system.file("css_data/MeasPattern.txt", package = "SeaSondeR")
apm_obj <- seasonder_readSeaSondeRAPMFile(apm_file)
cs_obj <- seasonder_createSeaSondeRCS(cs_file, seasonder_apm_object = apm_obj)
# Retrieve and set header
new_header <- seasonder_getSeaSondeRCS_header(cs_obj)
cs_obj <- seasonder_setSeaSondeRCS_header(cs_obj, new_header)
print(seasonder_getSeaSondeRCS_header(cs_obj))
```

seasonder_setSeaSondeRCS_MUSIC

Set MUSIC Data in a SeaSondeRCS Object

Description

This function assigns MUSIC analysis results to a SeaSondeRCS object. The MUSIC data is stored within the object's `MUSIC_data` attribute under the field `MUSIC`. Currently, no explicit validation is performed on the provided MUSIC data.

Usage

```
seasonder_setSeaSondeRCS_MUSIC(seasonder_cs_object, MUSIC)
```

Arguments

`seasonder_cs_object`	A `SeaSondeRCS` object containing cross-spectral data and metadata.
`MUSIC`	A data structure containing the MUSIC algorithm results. This is typically a list or tibble produced during the MUSIC processing workflow.

Details

This low-level setter function updates the SeaSondeRCS object's MUSIC data by assigning the provided MUSIC results to the `MUSIC` field within the `MUSIC_data` attribute. It is intended for use during the MUSIC processing workflow.

Value

The updated `SeaSondeRCS` object with the specified MUSIC data stored in its attributes.

seasonder_setSeaSondeRCS_MUSIC_doppler_interpolation

Set the Doppler Interpolation Factor in a SeaSondeRCS Object

Description

This function validates and assigns the Doppler interpolation factor in the SeaSondeRCS object, updating the corresponding option in the `MUSIC_data` field.

Usage

```
seasonder_setSeaSondeRCS_MUSIC_doppler_interpolation(
  seasonder_cs_object,
  doppler_interpolation
)
```

Arguments

`seasonder_cs_object`
: A SeaSondeRCS object containing radar data and metadata.

`doppler_interpolation`
: An integer specifying the Doppler interpolation factor. Must be 1, 2, 3, or 4.

Details

The function performs the following operations:

1. Validates the value of `doppler_interpolation` using the function `SeaSondeRCS_MUSIC_validate_doppler_interpo`
2. Updates the attribute `MUSIC_options$doppler_interpolation` of the SeaSondeRCS object with the validated value.

Value

The SeaSondeRCS object with the updated Doppler interpolation option.

See Also

`SeaSondeRCS_MUSIC_validate_doppler_interpolation` for Doppler interpolation factor validation.

Examples

```
# Create a valid SeaSondeRCS object for examples
cs_file <- system.file("css_data/CSS_TORA_24_04_04_0700.cs", package = "SeaSondeR")
apm_file <- system.file("css_data/MeasPattern.txt", package = "SeaSondeR")
apm_obj <- seasonder_readSeaSondeRAPMFile(apm_file)
cs_object <- seasonder_createSeaSondeRCS(cs_file, seasonder_apm_object = apm_obj)
# Set the Doppler interpolation factor to 2 (internal alias)
cs_object <- seasonder_setSeaSondeRCS_MUSIC_doppler_interpolation(cs_object, 2)
```

`seasonder_setSeaSondeRCS_MUSIC_dual_solutions_proportion` *Set Dual Solutions Proportion for MUSIC Analysis*

Description

This function assigns the dual solutions proportion to the MUSIC data of a SeaSondeRCS object. The dual solutions proportion represents the fraction of solutions identified as dual in the MUSIC processing workflow. Currently, no explicit validation of the provided value is performed.

Usage

```
seasonder_setSeaSondeRCS_MUSIC_dual_solutions_proportion(
  seasonder_cs_object,
  dual_solutions_proportion
)
```

Arguments

`seasonder_cs_object`
: A `SeaSondeRCS` object containing MUSIC analysis results.

`dual_solutions_proportion`
: A numeric value representing the proportion of dual solutions.

Details

The function updates the `dual_solutions_proportion` field within the `MUSIC_data` attribute. This value is later used to assess the prevalence of dual bearing solutions in the MUSIC results.

Value

The updated `SeaSondeRCS` object with the dual solutions proportion stored in its MUSIC data.

`seasonder_setSeaSondeRCS_MUSIC_interpolated_data` *Set Interpolated MUSIC Data in a SeaSondeRCS Object*

Description

This function assigns the interpolated cross-spectral data to the MUSIC data attribute of a SeaSondeRCS object. It stores the provided interpolated data into the `interpolated_data` field of the MUSIC data. If no data is provided, it defaults to the output of `seasonder_MUSICInitInterpolatedData()`.

Usage

```
seasonder_setSeaSondeRCS_MUSIC_interpolated_data(
  seasonder_cs_object,
  interpolated_data
)
```

Arguments

`seasonder_cs_object`
: A `SeaSondeRCS` object containing cross-spectral and MUSIC data.

`interpolated_data`
: A data structure (typically a list or tibble) representing the interpolated cross-spectral data. If `NULL`, the function uses `seasonder_MUSICInitInterpolatedData()` to initialize the structure.

Details

The function assigns the provided interpolated data (or initializes a new data structure) to the `interpolated_data` field within the MUSIC data attribute. This structure is intended for use in further MUSIC processing steps, where interpolated cross-spectral data is required for refining the estimation of Doppler bins.

Value

The updated SeaSondeRCS object with the interpolated_data field set in its MUSIC data attribute.

---

seasonder_setSeaSondeRCS_MUSIC_parameters
Set MUSIC Parameters for a SeaSondeRCS Object

Description

This function updates the MUSIC algorithm parameters stored in a SeaSondeRCS object's MUSIC data attribute. The parameters are updated in the MUSIC options under the MUSIC_parameters field. Currently, no explicit validation of the provided parameters is performed.

Usage

```
seasonder_setSeaSondeRCS_MUSIC_parameters(
  seasonder_cs_object,
  MUSIC_parameters = seasonder_defaultMUSIC_parameters()
)
```

Arguments

seasonder_cs_object	A SeaSondeRCS object containing cross-spectral data and metadata.
MUSIC_parameters	A numeric vector of parameters for the MUSIC algorithm. Defaults to the result of seasonder_defaultMUSIC_parameters().

Details

The function assigns the provided MUSIC_parameters vector to the MUSIC_parameters field within the MUSIC_options list, which is stored in the object's MUSIC_data attribute. These parameters are used in various steps of the MUSIC processing workflow.

Value

The updated SeaSondeRCS object with the new MUSIC parameters stored in its MUSIC options.

seasonder_setSeaSondeRCS_NoiseLevel

Set Noise Level for SeaSondeRCS Object

Description

This function updates the noise level for a specified antenna within a SeaSondeRCS object. It retrieves the current "NoiseLevel" attribute (or initializes it with the default if missing), updates the noise level for the given antenna, and stores it back in the object.

Usage

```
seasonder_setSeaSondeRCS_NoiseLevel(
  seasonder_cs_object,
  NoiseLevel,
  antenna = 3
)
```

Arguments

seasonder_cs_object	A SeaSondeRCS object.
NoiseLevel	A numeric value representing the new noise level to be set.
antenna	An integer specifying the antenna for which the noise level is updated. Default is 3.

Value

The updated SeaSondeRCS object with the modified "NoiseLevel" attribute.

seasonder_setSeaSondeRCS_ProcessingSteps

Setter for ProcessingSteps

Description

Setter for ProcessingSteps

Usage

```
seasonder_setSeaSondeRCS_ProcessingSteps(
  seasonder_cs_object,
  processing_steps,
  append = TRUE
)
```

Arguments

seasonder_cs_object	SeaSondeRCS object
processing_steps	new value
append	append the new step or replace previous steps? Default: TRUE

Value

A SeaSondeRCS object with updated ProcessingSteps.

Examples

```
# Create a valid SeaSondeRCS object for examples
cs_file <- system.file("css_data/CSS_TORA_24_04_04_0700.cs", package = "SeaSondeR")
apm_file <- system.file("css_data/MeasPattern.txt", package = "SeaSondeR")
apm_obj <- seasonder_readSeaSondeRAPMFile(apm_file)
cs_obj <- seasonder_createSeaSondeRCS(cs_file, seasonder_apm_object = apm_obj)
# Define and append new processing steps
new_steps <- "Example processing step"
cs_obj <- seasonder_setSeaSondeRCS_ProcessingSteps(cs_obj, new_steps)
print(seasonder_getSeaSondeRCS_ProcessingSteps(cs_obj))
```

---

seasonder_skip_cs_field

Skip Reading a CSField and Return a Specified Value

Description

This function is a convenience mechanism to invoke the `seasonder_skip_cs_field` restart option. It can be used in custom condition handlers when reading a CSField from a binary connection encounters an error or condition. When called, it indicates the intention to skip reading the current CSField and return a specific value.

Usage

```
seasonder_skip_cs_field(cond, value)
```

Arguments

cond	A condition or error that occurred while reading the CSField.
value	The desired return value to use in place of the CSField reading that encountered an error.

Details

During the execution of the `seasonder_readCSField` function, errors or conditions can occur. To provide a structured mechanism to handle such cases, the function utilizes the `rlang::withRestarts` mechanism, offering a restart option named `seasonder_skip_cs_field`. This restart allows the function to gracefully handle reading errors by logging a relevant error message and returning a specified value.

The `seasonder_skip_cs_field` function provides an easy way to invoke this restart. When called within a custom condition handler, it signals the intention to skip the current CSField reading due to an error and specifies a return value.

Value

The value specified in the 'value' parameter.

Examples

```
# Example: Skip reading a CSField using a withRestarts handler to return a default value
r <- withRestarts(
  seasonder_skip_cs_field(simpleError("test error"), "default"),
  seasonder_skip_cs_field = function(cond, value) value
)
print(r)
```

`seasonder_skip_cs_file` *Skip SeaSonde Cross Spectra (CS) File Reading*

Description

This function serves as a restart for `seasonder_readSeaSondeCSFile`. When invoked, it provides a mechanism to gracefully handle file reading errors by logging an error message and skipping the current file processing.

Usage

```
seasonder_skip_cs_file(cond)
```

Arguments

`cond`	The condition or error that occurred during the file reading process. This is used to log a detailed error message indicating the reason for skipping the file.

Details

This function is meant to be used within a custom condition handler. When a problematic condition arises during the processing of a SeaSonde CS file, you can call `seasonder_skip_cs_file(cond)` to trigger this restart, which allows for a graceful degradation by logging an error message and returning a specified value.

The effect of invoking this restart is twofold:

1. An error message detailing the reason for skipping the file is logged.
2. The calling function (`seasonder_readSeaSondeCSFile`) will immediately return a list with `header = NULL` and `data = NULL`.

Value

A list with header = NULL and data = NULL.

Examples

```
# Example: Skip file reading using a withRestarts handler to return NULL header and data
result <- withRestarts(
  seasonder_skip_cs_file(simpleError("test error")),
  seasonder_skip_cs_file = function(cond) list(header = NULL, data = NULL)
)
print(result)
```

---

seasonder_smoothAPM *Smooth APM Data*

Description

This function smooths the antenna pattern data for each channel of a SeaSonde RAPM object by applying a moving average with a specified number of points.

Usage

```
seasonder_smoothAPM(seasonder_apm_object, smoothing)
```

Arguments

`seasonder_apm_object`	A SeaSonde RAPM object containing raw antenna pattern data.
`smoothing`	The number of points to use for the moving average smoothing.

Value

The SeaSonde RAPM object with smoothed antenna pattern data and an updated processing step.

Examples

```
# Smooth antenna pattern data from a test SeaSondeRAPM object
apm_file <- system.file("css_data/MeasPattern.txt", package = "SeaSondeR")
obj <- seasonder_readSeaSondeRAPMFile(apm_file)
smoothed_obj <- seasonder_smoothAPM(obj, 5)
```

---

seasonder_SmoothFORSS *Smooth Self-Spectra for First Order Region (FOR)*

Description

This function applies a smoothing operation to the self-spectra (SS) matrix of antenna 3 in a SeaSondeR cross-spectral object, specifically for First Order Region (FOR) processing. The smoothed self-spectra are stored as an attribute within the object.

Usage

```
seasonder_SmoothFORSS(seasonder_cs_object)
```

Arguments

seasonder_cs_object	A SeaSondeRCS object containing self-spectra data.

Details

The function retrieves the default Doppler smoothing factor (nsm) from seasonder_getFOR_parameters and applies the smoothing operation using seasonder_SmoothSS on the self-spectra of antenna 3.

Steps:

1. Retrieve the Doppler smoothing factor (nsm).
2. Apply the sliding mean smoothing to the self-spectra of antenna 3.
3. Store the smoothed matrix as an attribute within the SeaSondeRCS object.

The smoothing process helps stabilize the estimation of nulls between first- and second-order regions, preventing over-smoothing that could distort boundaries or under-smoothing that could introduce jagged edges.

Value

The input SeaSondeRCS object with the smoothed self-spectra stored as an attribute.

See Also

- seasonder_SmoothSS for performing the smoothing operation.
- seasonder_setSeaSondeRCS_FOR_SS_Smoothed for storing the smoothed self-spectra.
- seasonder_getFOR_parameters for retrieving default nsm values.

`seasonder_SmoothSS` *Smooth Self-Spectra Matrix Using a Sliding Window*

Description

This function applies a smoothing operation to the self-spectra (SS) matrix of a specific antenna in a SeaSondeR cross-spectral object. The smoothing is performed using a sliding mean over a specified number of Doppler bins.

Usage

```
seasonder_SmoothSS(seasonder_cs_object, antenna, smoothing = NULL)
```

Arguments

`seasonder_cs_object`	A `SeaSondeRCS` object containing self-spectra data.
`antenna`	A character or numeric identifier of the antenna whose self-spectra will be smoothed.
`smoothing`	Optional. An integer specifying the number of Doppler bins used for smoothing. If `NULL`, the function retrieves the default smoothing factor (`nsm`) from `seasonder_getFOR_parameters`.

Details

The smoothing process is performed using a centered sliding mean filter with a window of `nsm` bins. The window extends symmetrically before and after each bin, with adjustments based on whether `nsm` is even or odd:

- If `nsm` is even, the window includes `nsm/2` bins before and after the target bin.
- If `nsm` is odd, the window includes `(nsm - 1)/2` bins before and `(nsm - 1)/2 + 1` bins after.

The function utilizes `slide_mean` to apply the smoothing operation row-wise across the self-spectra matrix.

This smoothing implementation mimics the one performed by the tool AnalyzeSpectra of CODAR's Radial Suite R8.

Value

A matrix with the same dimensions as the input self-spectra matrix, but with smoothed values.

See Also

- `seasonder_getFOR_parameters` for retrieving default `nsm` values.
- `seasonder_getSeaSondeRCS_antenna_SSdata` for accessing self-spectra data.
- `slide_mean` for applying the sliding window mean operation.

`seasonder_splitLog` *Split Logs Based on Time Thresholds*

Description

The function splits the log entries into blocks based on time gaps between timestamps. The threshold for splitting can be provided or calculated based on the gaps in the log timestamps.

Usage

```
seasonder_splitLog(
  threshold = NULL,
  threshold_factor = 4,
  threshold_quantile = 0.9,
  min_threshold_secs = 10
)
```

Arguments

`threshold`	The time difference threshold for splitting the logs. If NULL, it's calculated.
`threshold_factor`	Multiplicative factor applied to the calculated threshold.
`threshold_quantile`	Quantile used for threshold calculation if `threshold` is NULL.
`min_threshold_secs`	Minimum threshold in seconds.

Value

A list of log blocks, each block being a vector of log entries.

Examples

```
# Enable logging
seasonder_enableLogs()
# Log some messages
seasonder_log("First log entry", "info")
Sys.sleep(0.1)
seasonder_log("Second log entry", "info")
# Split logs into blocks (using a 1-second threshold)
blocks <- seasonder_splitLog(threshold = as.difftime(300, units = "secs"))
str(blocks)
```

`seasonder_SwapDopplerUnits` *Convert Between Different Doppler Frequency Units*

Description

This function converts Doppler-related values between different units, including normalized Doppler frequency, Doppler bins, and absolute Doppler frequency (Hz), within a SeaSondeR object.

Usage

```
seasonder_SwapDopplerUnits(seasonder_cs_object, values, in_units, out_units)
```

Arguments

`seasonder_cs_object`	A SeaSondeR cross-spectral object containing Doppler bin metadata.
`values`	A numeric vector specifying the Doppler values to be converted.
`in_units`	A character string specifying the current unit of `values`. Must be one of: • `"normalized doppler frequency"`: Values are normalized by the Bragg frequency. • `"bins"`: Values represent Doppler bin indices. • `"doppler frequency"`: Values are in Hz.
`out_units`	A character string specifying the target unit for conversion. Must be one of the same three options as `in_units`.

Details

The function first validates that the input and output units are among the allowed options. If `in_units` and `out_units` are the same, the function returns the original values without modification.

The unit conversions follow this logic:

1. If converting from `"normalized doppler frequency"`:
   - To `"bins"`: Uses `seasonder_NormalizedDopplerFreq2Bins`.
   - To `"doppler frequency"`: Uses `seasonder_NormalizedDopplerFreq2DopplerFreq`.
2. If converting from `"bins"`:
   - To `"normalized doppler frequency"`: Uses `seasonder_Bins2NormalizedDopplerFreq`.
   - To `"doppler frequency"`: Uses `seasonder_Bins2DopplerFreq`.
3. If converting from `"doppler frequency"`:
   - To `"bins"`: Uses `seasonder_DopplerFreq2Bins`.
   - To `"normalized doppler frequency"`: Uses `seasonder_DopplerFreq2NormalizedDopplerFreq`.

Overall, the functions used for Doppler units conversion mimic the implementation of Doppler units displayed in SpectraPlotterMap 12 in Radial Suite R8

Value

A numeric vector with the converted Doppler values in the specified output unit.

References

COS. SeaSonde Radial Suite Release 8; CODAR Ocean Sensors (COS): Mountain View, CA, USA, 2016.

See Also

seasonder_NormalizedDopplerFreq2Bins, seasonder_Bins2NormalizedDopplerFreq, seasonder_DopplerFreq2Bins and related functions for unit-specific conversions.

---

seasonder_trimAPM *Trim APM Data*

Description

This function trims a specified number of points from the beginning and end of the antenna pattern data.

Usage

```
seasonder_trimAPM(seasonder_apm_object, trimming)
```

Arguments

seasonder_apm_object	A SeaSonde RAPM object containing the antenna pattern data.
trimming	The number of points to trim from each end.

Value

The SeaSonde RAPM object with trimmed antenna pattern data and updated attributes.

Examples

```
# Trim loops for a test SeaSondeRAPM object
apm_file <- system.file("css_data/MeasPattern.txt", package = "SeaSondeR")
obj <- seasonder_readSeaSondeRAPMFile(apm_file)
trimmed_obj <- seasonder_trimAPM(obj, 3)
```

`seasonder_v6_skip_transformation` *Trigger Restart for Skipping Transformation*

Description

This function provides a mechanism to invoke a restart during the reading and transformation process of the SeaSonde CS File Version 6 header. It allows users to skip transformations that may have caused errors and proceed with a provided value.

Usage

```
seasonder_v6_skip_transformation(cond, value)
```

Arguments

`cond`	The condition object that triggered the restart.
`value`	The provided value to be used when the transformation is skipped.

Details

This function specifically triggers the `seasonder_v6_skip_transformation` restart that allows for skipping a block transformation in the reading process of the SeaSonde CS File Version 6 header. When triggered, it logs an error message, skips the problematic transformation, and returns the provided value for the block.

Value

This function triggers a restart and does not return a usual value.

Integration with SeaSonde CS File Reading

The restart mechanism of this function is integrated within the `seasonder_readSeaSondeCSFileHeaderV6` function. If an error occurs during the transformation process of a specific block, the restart provides users with an option to skip the problematic transformation and proceed with a fallback value.

Examples

```
# Example: Skip transformation using a restart handler
res <- withRestarts(
  seasonder_v6_skip_transformation(simpleError("test error"), "default"),
  seasonder_v6_skip_transformation = function(cond, value) value
)
print(res)
```

`seasonder_validateAttributesSeaSondeRAPM` *Validate Attributes for a SeaSondeRAPM Object*

Description

This function validates the attributes of a given SeaSondeRAPM object to ensure they meet the required specifications.

Usage

```
seasonder_validateAttributesSeaSondeRAPM(seasonde_apm_obj)
```

Arguments

`seasonde_apm_obj` A SeaSondeRAPM object whose attributes are to be validated.

Details

The function performs validation on the following attributes of the SeaSondeRAPM object:

- quality_matrix
- BEAR
- Type
- Creator
- SiteName
- SiteOrigin
- FileName
- CreateTimeStamp
- ProcessingSteps
- AmplitudeFactors
- AntennaBearing
- StationCode
- BearingResolution
- Smoothing
- CommentLine
- FileID
- PhaseCorrections

It internally calls specific validation functions for each of these attributes. If any of the attributes are found to be invalid, the function will stop execution and display an error message.

Value

TRUE if all attributes are valid. The function will stop execution and display an error message if any of the attributes are invalid.

See Also

validate_SeaSondeRAPM_quality_matrix, validate_SeaSondeRAPM_BEAR, validate_SeaSondeRAPM_Type, validate_SeaSondeRAPM_Creator, validate_SeaSondeRAPM_SiteName, validate_SeaSondeRAPM_SiteOrigin, validate_SeaSondeRAPM_FileName, validate_SeaSondeRAPM_CreateTimeStamp, validate_SeaSondeRAPM_Processi validate_SeaSondeRAPM_AmplitudeFactors, validate_SeaSondeRAPM_AntennaBearing, validate_SeaSondeRAPM_S validate_SeaSondeRAPM_BearingResolution, validate_SeaSondeRAPM_Smoothing, validate_SeaSondeRAPM_Commer validate_SeaSondeRAPM_FileID, validate_SeaSondeRAPM_PhaseCorrections

Examples

```
# Create a test SeaSondeRAPM object by reading sample file
apm_file <- system.file("css_data/MeasPattern.txt", package = "SeaSondeR")
obj <- seasonder_readSeaSondeRAPMFile(apm_file)
valid <- seasonder_validateAttributesSeaSondeRAPM(obj)
```

---

seasonder_validateCalibrationMatrixSeaSondeRAPM

Validate Calibration Matrix for a SeaSondeRAPM Object

Description

This function validates the input calibration_matrix to ensure it meets the required specifications for use in a SeaSondeRAPM object.

Usage

```
seasonder_validateCalibrationMatrixSeaSondeRAPM(matrix)
```

Arguments

matrix	A 3 x b complex matrix for calibration, where b is the number of bearings.

Details

The function performs the following validation checks:

1. Confirms that the input is a matrix.
2. Verifies that the matrix has exactly three rows.
3. Checks that the matrix contains only complex numbers.

If any of these validation steps fail, the function will log a fatal error and stop the execution using `rlang::abort`.

Value

TRUE if the matrix is valid. The function will stop execution and display an error message if the matrix is invalid.

See Also

seasonder_createSeaSondeRAPM

Examples

```
valid <- seasonder_validateCalibrationMatrixSeaSondeRAPM(
  matrix(complex(real = 1, imaginary = 0), nrow = 3, ncol = 5)
)
```

---

seasonder_validateCSDataStructure

Validate the Data Structure of CrossSpectra Data

Description

This function checks the validity of the `data` structure for CrossSpectra (CS) data. It ensures that all required fields are present, the dimensions of the matrices are correct based on `nRanges` and `nDoppler`, and that the types of the data fields are as expected.

Usage

```
seasonder_validateCSDataStructure(data, nRanges, nDoppler)
```

Arguments

`data`	A list representing the CrossSpectra (CS) data. It should contain fields "SSA1", "SSA2", "SSA3", "CS12", "CS13", "CS23", and "QC".
`nRanges`	An integer specifying the expected number of range cells.
`nDoppler`	An integer specifying the expected number of Doppler cells.

Details

The function expects the following structure for the `data` list:

- `SSA1`, `SSA2`, `SSA3`, `QC`: Matrices with numeric values, with dimensions `nRanges` x `nDoppler`.
- `CS12`, `CS13`, `CS23`: Matrices with complex values, with dimensions `nRanges` x `nDoppler`.

Value

Invisible NULL if the data structure is valid. Otherwise, an error is thrown.

Error Management

This function utilizes the `rlang` package to manage errors and provide detailed and structured error messages:

Condition Classes:

- `seasonder_CS_data_structure_validation_error`: An error class indicating a problem with the data structure of the CrossSpectra (CS) data.

Condition Cases:

- Missing fields in the data.
- Incorrect dimensions for the matrices in the data.
- Incorrect data type for the fields in the data.

Examples

```
# Example with all required fields
data <- list(
  SSA1 = matrix(rep(NA_real_, 10 * 20), ncol = 20, byrow = TRUE),
  SSA2 = matrix(rep(NA_real_, 10 * 20), ncol = 20, byrow = TRUE),
  SSA3 = matrix(rep(NA_real_, 10 * 20), ncol = 20, byrow = TRUE),
  CS12 = matrix(complex(real = NA, imaginary = NA), nrow = 10, ncol = 20),
  CS13 = matrix(complex(real = NA, imaginary = NA), nrow = 10, ncol = 20),
  CS23 = matrix(complex(real = NA, imaginary = NA), nrow = 10, ncol = 20),
  QC   = matrix(rep(NA_real_, 10 * 20), ncol = 20, byrow = TRUE)
)
seasonder_validateCSDataStructure(data, 10, 20)
```

`seasonder_validateCSFileData` *Validate SeaSondeR CS File Data*

Description

This function performs multiple validation checks on a provided CS file in the SeaSondeR system. It checks the file for various conditions to determine if it meets the SeaSondeR standards.

Usage

```
seasonder_validateCSFileData(filepath, header)
```

Arguments

`filepath`	A character string indicating the path to the CS file to validate.
`header`	A list containing header information of the CS file.

Details

The function performs the following validation checks:

1. Verifies that the file size is greater than 10 bytes.
2. Validates the `nCsFileVersion` field in the header to ensure it's between 1 and 32.
3. Depending on the `nCsFileVersion`, verifies the appropriate file size, and the extent of various version headers (`nV1Extent`, `nV2Extent`, etc.).
4. Validates the `nRangeCells` and `nDopplerCells` fields to ensure they are within permissible ranges.
5. Depending on the `nCsKind` value, validates the file size against expected sizes based on `nRangeCells`, `nSpectraChannels`, and `nDopplerCells`.

Value

NULL invisibly. The function mainly serves to validate and will stop execution and log an error using `seasonder_logAndAbort` if any condition fails.

Condition Management

This function utilizes the `rlang` package to manage conditions and provide detailed and structured condition messages:

Condition Classes:

- `seasonder_validate_cs_file_error`: An error class that indicates a validation requirement was not met.

Condition Cases:

- Failure on any validation test.

References

Cross Spectra File Format Version 6. CODAR. 2016

`seasonder_validateCSHeaderStructure`

Validate the Header of CrossSpectra Data

Description

This function validates the structure of a header list that is expected to represent the metadata for a cross spectra file. It checks if the header is indeed a list and whether mandatory elements, such as the number of range cells and the number of Doppler cells, are present.

Usage

```
seasonder_validateCSHeaderStructure(header)
```

Arguments

header	A list representing the header metadata of a cross spectra file.

Value

Invisible NULL if the header structure is valid. Otherwise, an error is thrown.

Details

The function primarily checks for two conditions:

- Whether the provided header argument is a list.
- Whether the nRangeCells and nDopplerCells are present in the header.

Condition Management

This function utilizes the `rlang` package to manage conditions and provide detailed and structured condition messages:

Condition Classes:

- `seasonder_CS_header_is_not_a_list`: Triggered when the header parameter is not a list.
- `seasonder_CS_missing_nRange_nDoppler_error`: Triggered when either nRangeCells or nDopplerCells is missing from the header.

Examples

```
header <- list(nRangeCells = 100, nDopplerCells = 256)
seasonder_validateCSHeaderStructure(header)
```

`seasonder_validateFORMethod` *Validate First Order Region (FOR) Processing Method*

Description

This function checks whether the specified method for First Order Region (FOR) detection is supported. If an unsupported method is provided, it logs an error and aborts execution.

Usage

```
seasonder_validateFORMethod(method)
```

Arguments

method	A character string specifying the FOR processing method. Currently, only `"SeaSonde"` is supported.

Details

The function verifies that the method argument is valid. If the method is not recognized, an error is raised using seasonder_logAndAbort.

Supported Methods:

- "SeaSonde": Implements first-order region detection based on CODAR's SeaSonde methodology.

Value

The function returns the input method invisibly if it is valid.

See Also

- seasonder_validateFOR_parameters for FOR parameter validation.
- seasonder_logAndAbort for error handling and logging.

---

seasonder_validateFOR_parameters

Validate First Order Region (FOR) Parameters

Description

This function validates and assigns default values to the parameters used in defining the First Order Region (FOR) in a SeaSondeR cross-spectral object. It ensures that all necessary parameters are present and assigns appropriate defaults where values are missing.

Usage

```
seasonder_validateFOR_parameters(
  seasonder_cs_object,
  FOR_parameters,
  method = "SeaSonde"
)
```

Arguments

seasonder_cs_object	A SeaSondeRCS object containing metadata about the Doppler spectrum.
FOR_parameters	A named list containing the parameters for first-order region detection.
method	A character string specifying the validation method. Default is "SeaSonde". Currently, only "SeaSonde" is supported.

Details

The function validates FOR parameters and assigns default values where necessary. If the selected method is "SeaSonde", the function ensures that each parameter is defined and, if missing, assigns it a default value based on `seasonder_defaultFOR_parameters`.

The parameters validated include:

- **nsm** (Doppler Smoothing): Number of points used for spectral smoothing.
- **fdown** (Peak Power Dropoff): Defines how far below peak power the algorithm descends before searching for the null.
- **flim** (Null Below Peak Power): Specifies a power threshold for identifying the first-order region.
- **noisefact** (Signal to Noise): Threshold above the noise floor that a spectral bin must exceed to be considered first-order.
- **currmax** (Maximum Velocity): Maximum radial velocity allowed in the first-order region.
- **reject_distant_bragg**: Logical flag indicating whether to reject Bragg regions that are too distant from the central Bragg frequency.
- **reject_noise_ionospheric**: Logical flag indicating whether to reject Bragg regions affected by ionospheric noise.
- **reject_noise_ionospheric_threshold**: Threshold (in dB) for rejecting first-order regions based on noise contamination.
- **reference_noise_normalized_limits**: Estimated reference noise range in normalized Doppler frequency, computed using `seasonder_estimateReferenceNoiseNormalizedLimits`.

Value

A named list containing validated and completed FOR parameters.

See Also

- `seasonder_defaultFOR_parameters` for default FOR settings.
- `seasonder_estimateReferenceNoiseNormalizedLimits` for computing reference noise limits.
- `seasonder_validateFORMethod` for validating the processing method.

`self_spectra_to_dB` *Convert Self-Spectra Power to dB*

Description

This function converts self-spectra power values from a linear scale to decibels (dB). The transformation considers the receiver gain to adjust the power measurements accordingly.

Usage

```
self_spectra_to_dB(spectrum_values, receiver_gain)
```

Arguments

`spectrum_values`	A numeric vector. The power values in linear scale.
`receiver_gain`	A numeric scalar. The receiver gain in decibels (dB).

Details

The conversion follows the equation:

$$dB = 10 \log_{10}(|P|) - G$$

where:

- \(dB \) is the power in decibels,
- \(P \) is the self-spectra power in linear scale,
- \(G \) is the receiver gain in decibels.

Absolute values of power are used to ensure valid logarithmic calculations.

Value

A numeric vector of power values in decibels (dB).

See Also

`dB_to_self_spectra` for the reverse conversion.

`summary.SeaSondeRAPM` *Summarizes a SeaSondeRAPM Object*

Description

This function prints a summary of a SeaSondeRAPM object by displaying its processing steps. The processing steps provide a record of the transformations and operations applied to the object, which can be useful for debugging and understanding the data workflow.

Usage

```
## S3 method for class 'SeaSondeRAPM'
summary(object, ...)
```

Arguments

`object`	An object of class "SeaSondeRAPM". This object should be created using the seasonder_createSeaSondeRAPM() function and must include a calibration matrix, a quality matrix, the BEAR attribute, and a StationCode.
`...`	Additional arguments that might be passed to other methods; currently not used.

Details

The function first verifies that the provided object inherits from the "SeaSondeRAPM" class. It then retrieves the processing steps associated with the object via the seasonder_getSeaSondeRAPM_ProcessingSteps() function. These steps are concatenated into a single string, which is printed alongside a header indicating that they represent the processing steps. This method is primarily used for diagnostic purposes and for verifying that the object has undergone the intended series of operations.

Value

Invisibly returns the input SeaSondeRAPM object. This allows the function to be used in a sequence of operations (e.g., chaining) without printing the object again after the summary is displayed.

Examples

```
# Summarize a test SeaSondeRAPM object
apm_file <- system.file("css_data/MeasPattern.txt", package = "SeaSondeR")
obj <- seasonder_readSeaSondeRAPMFile(apm_file)
summary(obj)
```

---

summary.SeaSondeRCS *Summary Method for SeaSondeRCS Object*

Description

Provides a concise summary of a SeaSondeRCS object by printing the processing steps that have been applied to the data contained in the object.

Usage

```
## S3 method for class 'SeaSondeRCS'
summary(object, ...)
```

Arguments

object	An object of class "SeaSondeRCS". This object should contain at least a header list with metadata (such as station name, date/time, and cell counts) and processing step information retrieved by the function seasonder_getSeaSondeRCS_ProcessingSteps.
...	Additional arguments. Currently not used, but supplied for compatibility with generic summary methods.

Details

This method first validates that the input object inherits from the "SeaSondeRCS" class. It then retrieves the processing steps applied to the data using seasonder_getSeaSondeRCS_ProcessingSteps, formats them into a readable string, and outputs the result via the cat function. The function is designed for interactive use, and its output facilitates quick inspection of the object.

Value

Invisibly returns the original SeaSondeRCS object.

Examples

```
obj <- list(header = list(nSiteCodeName = "Station1",
                          nDateTime = Sys.time(),
                          nDopplerCells = 256,
                          nRangeCells = 100))
class(obj) <- "SeaSondeRCS"
summary(obj)
```

---

validate_SeaSondeRAPM_AmplitudeFactors

Validate AmplitudeFactors Attribute for a SeaSondeRAPM Object

---

Description

This function validates if the provided AmplitudeFactors is a numeric vector of length 2.

Usage

```
validate_SeaSondeRAPM_AmplitudeFactors(factors)
```

Arguments

factors	The numeric vector to be validated.

Value

Returns TRUE if the validation passes.

---

validate_SeaSondeRAPM_AntennaBearing

Validate AntennaBearing Attribute for a SeaSondeRAPM Object

Description

This function validates if the provided AntennaBearing is a numeric value.

Usage

```
validate_SeaSondeRAPM_AntennaBearing(bearing)
```

Arguments

bearing	The numeric value to be validated.

Value

Returns TRUE if the validation passes.

`validate_SeaSondeRAPM_BEAR`
Validate BEAR Attribute for a SeaSondeRAPM Object

Description

This function validates if the provided BEAR is a numeric vector and if its length matches the number of columns in the calibration_matrix of the given SeaSondeRAPM object. It also validates that the bearings are between -180 and 180 degrees.

Usage

```
validate_SeaSondeRAPM_BEAR(vector, seasonde_apm_obj)
```

Arguments

`vector`	The numeric vector to be validated.
`seasonde_apm_obj`	The SeaSondeRAPM object for compatibility check.

Value

Returns TRUE if the validation passes.

`validate_SeaSondeRAPM_BearingResolution`
Validate BearingResolution Attribute for a SeaSondeRAPM Object

---

Description

This function validates if the provided BearingResolution is a numeric value.

Usage

```
validate_SeaSondeRAPM_BearingResolution(resolution)
```

Arguments

`resolution`	The numeric value to be validated.

Value

Returns TRUE if the validation passes.

validate_SeaSondeRAPM_CommentLine

Validate CommentLine Attribute for a SeaSondeRAPM Object

Description

This function validates if the provided CommentLine is a character string.

Usage

```
validate_SeaSondeRAPM_CommentLine(comment)
```

Arguments

comment	The character string to be validated.

Value

Returns TRUE if the validation passes.

validate_SeaSondeRAPM_CreateTimeStamp

Validate CreateTimeStamp Attribute for a SeaSondeRAPM Object

Description

This function validates if the provided CreateTimeStamp is a POSIXct Date object.

Usage

```
validate_SeaSondeRAPM_CreateTimeStamp(timestamp)
```

Arguments

timestamp	The Date object to be validated.

Value

Returns TRUE if the validation passes.

validate_SeaSondeRAPM_Creator

Validate Creator Attribute for a SeaSondeRAPM Object

Description

This function validates if the provided Creator is a character string.

Usage

```
validate_SeaSondeRAPM_Creator(creator)
```

Arguments

creator	The character string to be validated.

Value

Returns TRUE if the validation passes.

validate_SeaSondeRAPM_FileID

Validate FileID Attribute for a SeaSondeRAPM Object

Description

This function validates if the provided FileID is a unique character string.

Usage

```
validate_SeaSondeRAPM_FileID(id)
```

Arguments

id	The unique character string to be validated.

Value

Returns TRUE if the validation passes.

validate_SeaSondeRAPM_FileName

Validate FileName Attribute for a SeaSondeRAPM Object

Description

This function validates if the provided FileName is a character string.

Usage

```
validate_SeaSondeRAPM_FileName(file_name)
```

Arguments

file_name	The character string to be validated.

Value

Returns TRUE if the validation passes.

validate_SeaSondeRAPM_PhaseCorrections

Validate PhaseCorrections Attribute for a SeaSondeRAPM Object

Description

This function validates if the provided PhaseCorrections attribute is a numeric vector of length 2.

Usage

```
validate_SeaSondeRAPM_PhaseCorrections(corrections)
```

Arguments

corrections	The numeric vector to be validated.

Value

Returns TRUE if the validation passes.

validate_SeaSondeRAPM_ProcessingSteps

Validate ProcessingSteps Attribute for a SeaSondeRAPM Object

Description

This function validates if the provided ProcessingSteps is a character vector.

Usage

```
validate_SeaSondeRAPM_ProcessingSteps(steps)
```

Arguments

steps	The character vector to be validated.

Value

Returns TRUE if the validation passes.

validate_SeaSondeRAPM_quality_matrix

Validate quality_matrix Attribute for a SeaSondeRAPM Object

Description

This function validates if the provided quality_matrix is a 3-row complex matrix. It also checks if the number of columns matches that of the calibration_matrix in the given SeaSondeRAPM object.

Usage

```
validate_SeaSondeRAPM_quality_matrix(matrix, seasonde_apm_obj)
```

Arguments

matrix	The matrix to be validated.
seasonde_apm_obj	The SeaSondeRAPM object for compatibility check.

Value

Returns TRUE if the validation passes.

validate_SeaSondeRAPM_SiteName

Validate SiteName Attribute for a SeaSondeRAPM Object

Description

This function validates if the provided SiteName is a character string.

Usage

```
validate_SeaSondeRAPM_SiteName(site_name)
```

Arguments

site_name	The character string to be validated.

Value

Returns TRUE if the validation passes.

validate_SeaSondeRAPM_SiteOrigin

Validate SiteOrigin Attribute for a SeaSondeRAPM Object

Description

This function validates if the provided SiteOrigin is a numeric vector of length 2.

Usage

```
validate_SeaSondeRAPM_SiteOrigin(site_origin)
```

Arguments

site_origin	The numeric vector to be validated.

Value

Returns TRUE if the validation passes.

validate_SeaSondeRAPM_Smoothing

Validate Smoothing Attribute for a SeaSondeRAPM Object

Description

This function validates if the provided Smoothing is a numeric value.

Usage

```
validate_SeaSondeRAPM_Smoothing(smoothing)
```

Arguments

smoothing	The numeric value to be validated.

Value

Returns TRUE if the validation passes.

validate_SeaSondeRAPM_StationCode

Validate StationCode Attribute for a SeaSondeRAPM Object

Description

This function validates if the provided StationCode is an empty character string or a 4-character string of length 1.

Usage

```
validate_SeaSondeRAPM_StationCode(code)
```

Arguments

code	The character string to be validated.

Value

Returns TRUE if the validation passes.

validate_SeaSondeRAPM_Type

Validate Type Attribute for a SeaSondeRAPM Object

Description

This function validates if the provided Type is a character string.

Usage

```
validate_SeaSondeRAPM_Type(type)
```

Arguments

type	The character string to be validated.

Value

Returns TRUE if the validation passes.

validate_SeaSondeRCS_ProcessingSteps

Validate ProcessingSteps Attribute for a SeaSondeRCS Object

Description

This function validates if the provided ProcessingSteps is a character vector.

Usage

```
validate_SeaSondeRCS_ProcessingSteps(steps)
```

Arguments

steps	The character vector to be validated.

Value

Returns TRUE if the validation passes.

Index