



Remotior Sensus Documentation

Release 0.0.35

Luca Congedo

Jun 03, 2023

CONTENTS

1 Documentation	3
1.1 Introduction	3
1.1.1 Management of Raster Bands	3
1.1.2 Performance	4
1.1.3 Machine Learning	6
1.1.4 Source Code	6
1.1.5 How to cite	6
1.1.6 Official site	6
1.2 Installation	6
1.2.1 Dependencies	6
1.2.2 Installation with Conda	6
1.2.3 Installation in Linux	7
1.2.4 Installation in OS X	7
1.2.5 Installation in Windows	7
1.2.6 Package installation	7
1.3 Quickstart	7
1.4 Tools	7
1.4.1 remotior_sensus.tools package	8
1.5 Core Modules	39
1.5.1 remotior_sensus.core package	39
1.6 API Reference	84
1.6.1 remotior_sensus package	84
1.7 Basic Tutorials	84
1.7.1 Download Sentinel-2 Data and Calculate NDVI	84
Python Module Index	85
Index	87



Remotior Sensus (which is Latin for “a more remote sense”) is a Python package that allows for the processing of remote sensing images and GIS data.

Source code: https://github.com/semiautomaticgit/remotior_sensus

Bug reports: https://github.com/semiautomaticgit/remotior_sensus/issues

1.1 Introduction



Remotior Sensus, developed by Luca Congedo, is a Python package that allows for the processing of remote sensing images and GIS data.

The main objective is to simplify the processing of remote sensing data through practical and integrated APIs that span from the download and preprocessing of satellite images to the postprocessing of classifications and GIS data. Basic dependencies are [NumPy](#), [SciPy](#) for calculations, and [GDAL](#) for managing spatial data.

The main features are:

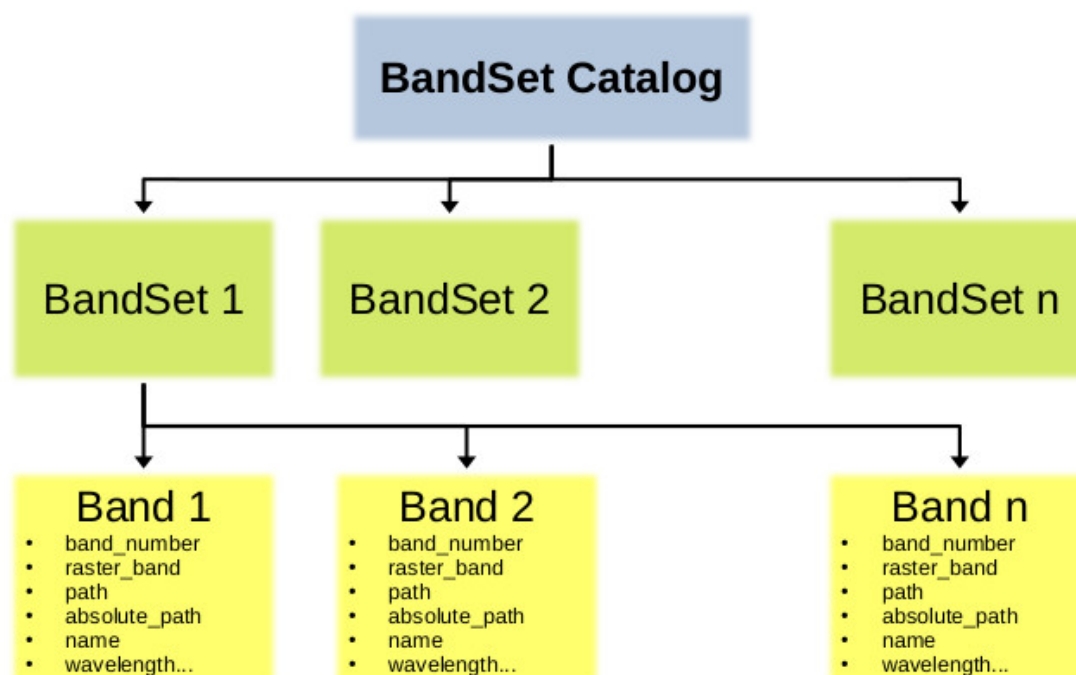
- **Search and Download** of remote sensing data such as Landsat and Sentinel-2.
- **Preprocessing** of several products such as Landsat and Sentinel-2 images.
- **Processing and postprocessing** tools to perform image classification through machine learning, manage GIS data and perform spatial analyses.
- **Parallel processing** available for most processing tools.

WARNING: Remotior Sensus is still in early development; new tools are going to be added, tools and APIs may change, and one may encounter issues and bugs using Remotior Sensus.

1.1.1 Management of Raster Bands

Most tools accept raster bands as input, defined through the file path.

In addition, raster bands can be managed through a catalog of BandSets (see [bandset_catalog\(\)](#) (page 39)), where each BandSet is an object that includes information about single bands (from the file path to the spatial and spectral characteristics). Bands in a BandSet can be referenced by the properties thereof, such as order number or center wavelength.

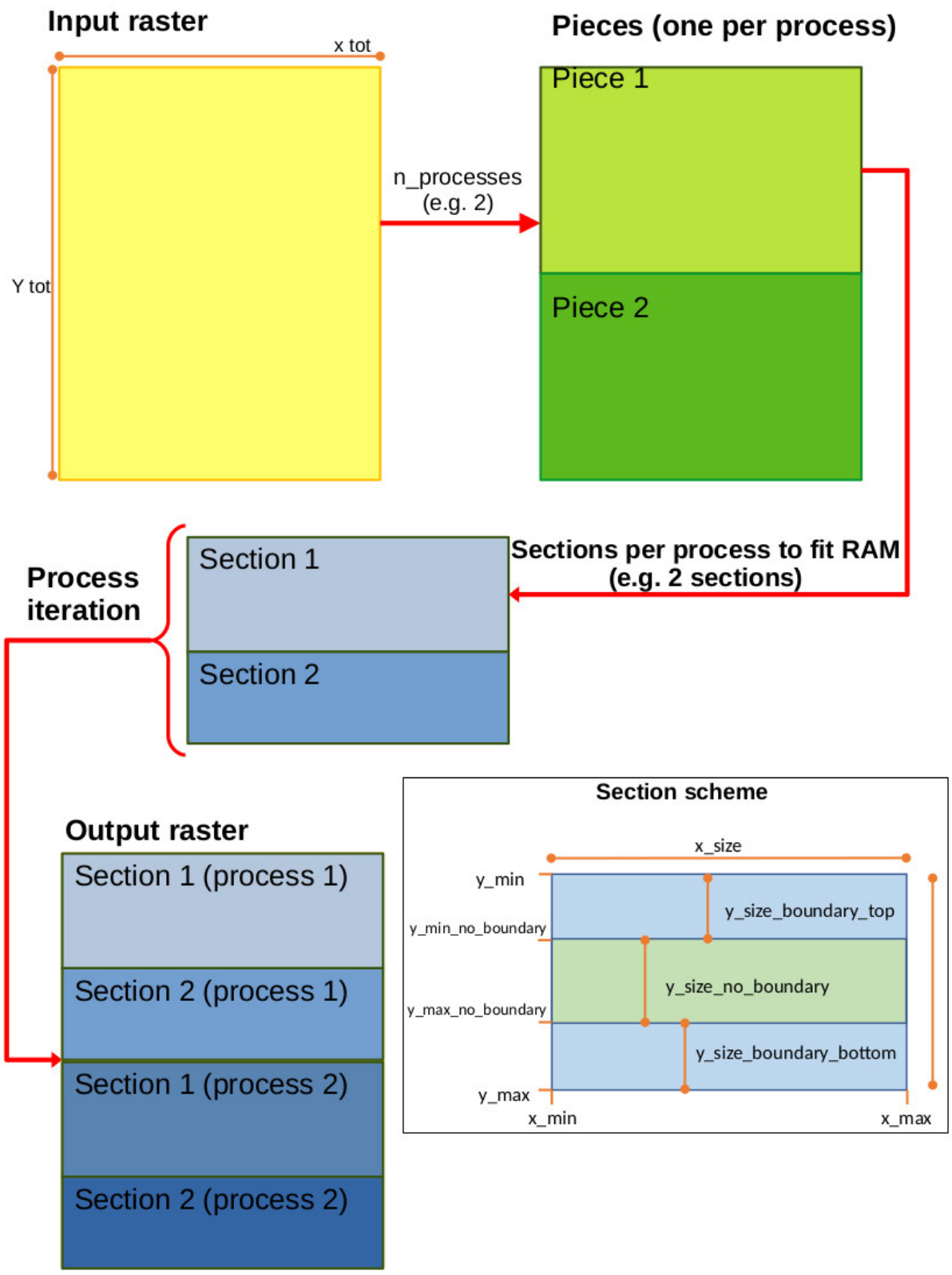


Multiple BandSets can be defined and identified by their reference number. Therefore, BandSets can be used as input for operations on multiple bands such as Principal Components Analysis, classification, mosaic, or band calculation.

In band calculations (see *band_calc()* (page 8)) name alias of bands based on center wavelength (e.g. blue, red) can be used to simplify the structure of calculation expression.

1.1.2 Performance

Most tools are designed to run in parallel processes, through a simple and effective parallelization approach based on dividing the raster input in sections that are distributed to available threads, maximizing the use of available RAM. This allows even complex algorithms to run in parallel. Optionally, the output file can be a virtual raster collecting the output rasters (corresponding to the sections) written independently by parallel processes; this avoids the time required to produce a unique raster output. Most tools allow for on the fly reprojection of input data.



1.1.3 Machine Learning

Remotior Sensus optional dependencies are `PyTorch` and `scikit-learn`, which are integrated in the tool `band_classification()` (page 10) to allow for land cover classification through machine learning. The aim is to simplify the training process and development of the model.

1.1.4 Source Code

The source code of Remotior Sensus is available at https://github.com/semiautomaticgit/remotior_sensus .

To report issues please visit https://github.com/semiautomaticgit/remotior_sensus/issues .

License of Remotior Sensus

Remotior Sensus is free software: you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation, either version 3 of the License, or (at your option) any later version. Remotior Sensus is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License for more details. You should have received a copy of the GNU General Public License along with Remotior Sensus. If not, see <https://www.gnu.org/licenses/>.

1.1.5 How to cite

Congedo, Luca, (2023). Remotior Sensus. https://github.com/semiautomaticgit/remotior_sensus

1.1.6 Official site

For more information and tutorials visit the official site



From GIS to Remote Sensing

1.2 Installation

This section describes Remotior Sensus installation along with the required dependencies.

1.2.1 Dependencies

Remotior Sensus requires `GDAL`, `NumPy` and `SciPy` for most functionalities. Optionally, `scikit-learn` and `PyTorch` are required for machine learning. Python ≥ 3.10 is recommended.

1.2.2 Installation with Conda

Before installing Remotior Sensus please install the dependencies using a `Conda` environment (if you don't know `Conda` please read <https://conda-forge.org/docs>). For instance, you can use `Miniforge` to create a `Conda` environment.

```
$ conda create -c conda-forge --name environment python=3.10
Proceed ([y]/n)? y
$ conda activate environment
```

Install Remotior Sensus using `Conda` (the fundamental dependencies are also installed):

```
$ conda install -c conda-forge remotior-sensus
```

For machine learning functionalities run:

```
$ conda install -c conda-forge remotior-sensus scikit-learn pytorch
```

1.2.3 Installation in Linux

The suggested way to install Remotior Sensus is using *Conda* (see *Installation with Conda* (page 6)).

Depending on the system, one could install the required dependencies as:

```
$ sudo apt-get install python3-numpy python3-scipy gdal-bin
```

For Remotior Sensus package installation use *pip*:

```
$ pip install -U remotior-sensus
```

1.2.4 Installation in OS X

The suggested way to install Remotior Sensus is using *Conda* (see *Installation with Conda* (page 6)).

1.2.5 Installation in Windows

The suggested way to install Remotior Sensus is using *Conda* (see *Installation with Conda* (page 6)).

1.2.6 Package installation

Given that dependencies are installed, for Remotior Sensus package installation use *pip*:

```
$ pip install -U remotior-sensus
```

1.3 Quickstart

Link to the guide: <https://colab.research.google.com/gist/semiautomaticgit/0fdb7c2c5c0b3b990cf342d226b8ee43/quickstart.ipynb>

1.4 Tools

The following are the available tools.

1.4.1 remotior_sensus.tools package

Submodules

remotior_sensus.tools.band_calc module

Band calc.

This tool allows for mathematical calculations (pixel by pixel) between bands or single band rasters. A new raster file is created as result of calculation.

Typical usage example:

```
>>> # import Remotior Sensus and start the session
>>> import remotior_sensus
>>> rs = remotior_sensus.Session()
>>> # start the process
>>> output = rs.band_calc(
... input_raster_list=['file1.tif', 'file2.tif'], output_path='output.tif',
... expression_string='"file1 + file2"', input_name_list=['file1', 'file2']
... )
```

`remotior_sensus.tools.band_calc.band_calc`(*expression_string*: str, *output_path*: None | str = None, *input_raster_list*: list | None = None, *input_name_list*: list | None = None, *n_processes*: None | int = None, *available_ram*: None | int = None, *align_raster*: None | str = None, *extent_raster*: None | str = None, *extent_list*: list | None = None, *extent_intersection*: bool | None = True, *xy_resolution_list*: list | None = None, *input_nodata_as_value*: bool | None = None, *use_value_as_nodata*: None | int = None, *output_nodata*: None | int = None, *output_datatype*: None | str = None, *use_scale*: None | float = None, *use_offset*: None | float = None, *calc_datatype*: None | str = None, *any_nodata_mask*: bool | None = False, *bandset_catalog*: [BandSetCatalog](#) (page 50) | None = None, *bandset_number*: None | int = None, *input_bands*: [BandSet](#) (page 39) | None = None) → [OutputManager](#) (page 69)

Performs band calculation.

Calculation is defined by an expression string using variable names that corresponds to input bands or rasters. Expression can use band alias such as “bandset1b1”, spectral band alias such as “#NIR#” for Near-Infrared and “#RED#” for Red, or expression alias such as “#NDVI#”. Multiple expression lines can be entered for serial calculation. Several iteration functions are available for band sets. NumPy functions can also be used if the output is a single band array with the same size as input raster. Input rasters can have different projection definitions, as the tool will try to reproject input rasters on the fly based on a reference raster (first input raster or `align_raster`).

Parameters

- **input_raster_list** – list of input raster paths or list of lists [path, name] ignoring `input_name_list`.
- **output_path** – path of output file for single expression or path to a directory for multiple expression outputs.
- **expression_string** – expression string used for calculation; multiple expressions can be entered separated by new line.

- **input_name_list** – list of input raster names used in expressions (if name not defined in `input_raster_list`).
- **input_bands** – input `BandSet` for direct expression; in expressions, bands can be referred as “b1”, “b2”, etc.; also, spectral band alias such as “#RED#” or “#NIR#”; also “b*” for using all the bands.
- **n_processes** – number of threads for calculation.
- **available_ram** – number of megabytes of RAM available to processes.
- **align_raster** – string path of raster used for aligning output pixels and projections.
- **extent_raster** – string path of raster used for extent reference.
- **extent_list** – list of coordinates for defining calculation extent [left, top, right, bottom] in the same coordinates as the reference raster.
- **extent_intersection** – if True the output extent is geometric intersection of input raster extents, if False the output extent is the maximum extent from union of input raster extents.
- **xy_resolution_list** – list of [x, y] pixel resolution.
- **input_nodata_as_value** – if True then unmask the value of nodata pixels in calculations, if False then mask nodata pixels in calculations.
- **use_value_as_nodata** – use integer value as nodata in calculation.
- **output_nodata** – integer value used as nodata in output raster.
- **output_datatype** – string of data type for output raster such as `Float64`, `Float32`, `Int32`, `UInt32`, `Int16`, `UInt16`, or `Byte`.
- **use_scale** – float number used for scale for output.
- **use_offset** – float number used for offset for output.
- **calc_datatype** – data type used during calculation, which may differ from output datatype, such as `Float64`, `Float32`, `Int32`, `UInt32`, `Int16`, `UInt16`, or `Byte`.
- **any_nodata_mask** – if True then output nodata where any input is nodata, if False then output nodata where all the inputs are nodata, if None then do not apply nodata to output.
- **bandset_catalog** – `BandSetCatalog` object for using band sets in calculations.
- **bandset_number** – number of `BandSet` defined as current one.

Returns

`OutputManager()` (page 69) object with

- `paths = [output raster paths]`

Examples

Sum of two raster files

```
>>> output_object = band_calc(
...   input_raster_list=['file1.tif', 'file2.tif'], output_path='output.tif',
...   expression_string="file1 + file2", input_name_list=['file1', 'file2']
... )
>>> # for instance display the output path
>>> print(output_object.paths)
['output.tif']
```

Calculation setting output datatype

```
>>> output_object = band_calc(
... input_raster_list=['file1.tif', 'file2.tif'], output_path='output.tif',
... expression_string='"file1 + file2"', input_name_list=['file1', 'file2'],
... output_datatype='Int32'
... )
```

Calculation setting the output extent as the maximum extent from union of input raster extents

```
>>> output_object = band_calc(
... input_raster_list=['file1.tif', 'file2.tif'], output_path='output.tif',
... expression_string='"file1 + file2"', input_name_list=['file1', 'file2'],
... extent_intersection=False
... )
```

remotior_sensus.tools.band_classification module

Band classification.

This tool allows for the classification of remote sensing images, providing several algorithms such as Minimum Distance, Maximum Likelihood, Spectral Angle Mapping. Also, machine learning algorithms are provided through `PyTorch` (`pytorch_multi_layer_perceptron`) and `scikit-learn` (`random_forest`, `random_forest_ovr`, `support_vector_machine`, `multi_layer_perceptron`). This module includes tools for training the algorithms using Regions of Interest (ROIs) or spectral signatures.

Typical usage example:

```
>>> # import Remotior Sensus and start the session
>>> import remotior_sensus
>>> rs = remotior_sensus.Session()
>>> # start the process
>>> classification = rs.band_classification(
... input_bands=['file1.tif', 'file2.tif'], output_path='output.tif',
... algorithm_name=cfg.maximum_likelihood
... )
```

```
class remotior_sensus.tools.band_classification.Classifier(algorithm_name,
                                                         spectral_signatures,
                                                         covariance_matrices,
                                                         model_classifier,
                                                         input_normalization,
                                                         normalization_values)
```

Bases: object

Manages classifiers.

A classifier is an object which includes the required parameters to perform a classification, including the tools to perform the training.

algorithm_name

algorithm name selected form `cfg.classification_algorithms`.

spectral_signatures

a `SpectralSignaturesCatalog` containing spectral signatures.

covariance_matrices

dictionary of previously calculated covariance matrices (used in `maximum_likelihood`).

model_classifier

classifier object.

input_normalization

perform input normalization; options are `z_score` or `linear_scaling`.

normalization_values

list of normalization parameters defined for each variable [normalization expressions, mean values, standard deviation values, minimum values, maximum values].

framework_name

name of framework such as `classification_framework`, `scikit_framework`, or `pytorch_framework`.

classification_function

the actual classification function.

function_argument = a dictionary including arguments for the classification function

such as `model_classifier`, `covariance_matrices`, `normalization_values`, `spectral_signatures_catalog`.

Examples**Fit a classifier**

```
>>> # Start a session
>>> import remotior_sensus
>>> rs = remotior_sensus.Session()
>>> # create a BandSet
>>> catalog = rs.bandset_catalog()
>>> file_list = ['file1.tif', 'file2.tif', 'file3.tif']
>>> catalog.create_bandset(file_list, wavelengths=['Landsat 8'])
>>> # set a BandSet reference in signature catalog
>>> signature_catalog = rs.spectral_signatures_catalog(
>>>     bandset=catalog.get(1))
>>> # import vector in signature catalog
>>> signature_catalog.import_vector(
>>>     file_path='file.gpkg', macroclass_field='macroclass', class_field=
↪ 'class',
>>>     macroclass_name_field='macroclass', class_name_field='class',
>>>     calculate_signature=True)
>>> # train a minimum distance classifier
>>> classifier = Classifier.train(
>>>     spectral_signatures=signature_catalog,
>>>     algorithm_name='minimum distance'
>>> )
```

__init__(algorithm_name, spectral_signatures, covariance_matrices, model_classifier, input_normalization, normalization_values)

Initializes a classifier.

A classifier is an object which includes the

Parameters

- **algorithm_name** – algorithm name selected from `cfg.classification_algorithms`.
- **spectral_signatures** – a `SpectralSignaturesCatalog` containing spectral signatures.
- **covariance_matrices** – dictionary of previously calculated covariance matrices (used in `maximum_likelihood`).

- **model_classifier** – classifier object.
- **input_normalization** – perform input normalization; options are `z_score` or `linear_scaling`.
- **normalization_values** – list of normalization parameters defined for each variable [normalization expressions, mean values, standard deviation values, minimum values, maximum values].

classmethod load_classifier(*algorithm_name=None, spectral_signatures=None, covariance_matrices=None, model_classifier=None, input_normalization=None, normalization_values=None*)

Loads a classifier.

Creates a classifier from loading.

Parameters

- **algorithm_name** – algorithm name selected from `cfg.classification_algorithms`.
- **spectral_signatures** – a `SpectralSignaturesCatalog` containing spectral signatures.
- **covariance_matrices** – dictionary of previously calculated covariance matrices (used in `maximum_likelihood`).
- **model_classifier** – classifier object.
- **input_normalization** – perform input normalization; options are `z_score` or `linear_scaling`.
- **normalization_values** – list of normalization parameters defined for each variable [normalization expressions, mean values, standard deviation values, minimum values, maximum values].

Returns

Classifier object.

Examples

Load a classifier

```
>>> classifier = Classifier.load_classifier(  
>>> algorithm_name=algorithm_name, spectral_signatures=spectral_  
↪ signatures,  
>>> covariance_matrices=covariance_matrices, model_classifier=model_  
↪ classifier,  
>>> input_normalization=input_normalization, normalization_  
↪ values=normalization_values)
```

run_prediction(*input_raster_list, output_raster_path, n_processes: None | int = None, available_ram: None | int = None, macroclass: bool | None = True, threshold: bool | None = False, signature_raster: bool | None = False, classification_confidence: bool | None = False, virtual_raster: bool | None = None, min_progress: int | None = 1, max_progress: int | None = 100*)

Runs prediction.

Performs multiprocessing classification using a trained classifier using input bands.

Parameters

- **input_raster_list** – list of paths of input rasters.
- **output_raster_path** – path of output file.

- **n_processes** – number of parallel processes.
- **available_ram** – number of megabytes of RAM available to processes.
- **macroclass** – if True, use macroclass ID from ROIs or spectral signatures; if False use class ID.
- **threshold** – if False, classification without threshold; if True, use single threshold for each signature; if float, use this value as threshold for all the signature.
- **classification_confidence** – if True, write also additional classification confidence rasters as output.
- **signature_raster** – if True, write additional rasters for each spectral signature as output.
- **virtual_raster** – if True, create virtual raster output.
- **min_progress** – minimum progress value for *Progress()* (page 74).
- **max_progress** – maximum progress value for *Progress()* (page 74).

Returns

OutputManager object with

- path = [output path]

Examples

Save a trained classifier

```
>>> classifier = Classifier()
>>> # after the training
>>> prediction = classifier.run_prediction(
... input_raster_list = 'file1.tif', 'file2.tif', 'file3.tif',
... output_raster_path = 'file_path')
```

save_model(*output_path: str*) → *OutputManager* (page 69)

Saves classifier model.

Saves a classifier model to file to be loaded later.

Parameters

output_path – path of output file.

Returns

OutputManager object with

- path = [output path]

Examples

Save a trained classifier

```
>>> classifier = Classifier()
>>> # after the training
>>> saved_model = classifier.save_model(output_path=output_path)
```

```
classmethod train(spectral_signatures=None, algorithm_name=None, covariance_matrices=None,
    svc_classification_confidence=None, n_processes: int | None = None,
    available_ram: int | None = None, cross_validation=True, x_matrix=None,
    y=None, class_weight=None, input_normalization=None,
    normalization_values=None, find_best_estimator=False, rf_max_features=None,
    rf_number_trees=100, rf_min_samples_split=None, svm_c=None,
    svm_gamma=None, svm_kernel=None, pytorch_model=None,
    pytorch_optimizer=None, mlp_training_portion=None,
    pytorch_loss_function=None, mlp_hidden_layer_sizes=None, mlp_alpha=None,
    mlp_learning_rate_init=None, mlp_max_iter=None, mlp_batch_size=None,
    mlp_activation=None, pytorch_optimization_n_iter_no_change=None,
    pytorch_optimization_tol=None, pytorch_device=None, min_progress=1,
    max_progress=100)
```

Trains a classifier.

Trains a classifier using ROIs or spectral signatures.

Parameters

- **spectral_signatures** – a SpectralSignaturesCatalog containing spectral signatures.
- **algorithm_name** – algorithm name selected from `cfg.classification_algorithms`; if None, minimum distance is used.
- **n_processes** – number of parallel processes.
- **available_ram** – number of megabytes of RAM available to processes.
- **cross_validation** – if True, perform cross validation for algorithms provided through scikit-learn (random_forest, random_forest_ovr, support_vector_machine, multi_layer_perceptron).
- **x_matrix** – optional previously saved x matrix.
- **y** – optional previously saved y matrix.
- **covariance_matrices** – dictionary of previously calculated covariance matrices (used in maximum_likelihood).
- **svc_classification_confidence** – if True, write also additional classification confidence rasters as output; required information for support_vector_machine.
- **input_normalization** – perform input normalization; options are `z_score` or `linear_scaling`.
- **normalization_values** – list of normalization parameters defined for each variable [normalization expressions, mean values, standard deviation values, minimum values, maximum values].
- **class_weight** – specific for random forest and support_vector_machine, if None each class has equal weight 1, if 'balanced' weight is computed inversely proportional to class frequency.
- **find_best_estimator** – specific for scikit classifiers, if True, find automatically the best parameters and fit the model, if integer the greater the value the more are the tested combinations.
- **rf_max_features** – specific for random forest, if None all features are considered in node splitting, available options are 'sqrt' as square root of all the features, an integer number, or a float number for a fraction of all the features.
- **rf_number_trees** – specific for random forest, number of trees in the forest.
- **rf_min_samples_split** – specific for random forest through scikit, sets the minimum number of samples required to split an internal node; default = 2.

- **svm_c** – specific for support_vector_machine through scikit, sets the regularization parameter C; default = 1.
- **svm_gamma** – specific for support_vector_machine through scikit, sets the kernel coefficient; default = scale.
- **svm_kernel** – specific for support_vector_machine through scikit, sets the kernel; default = rbf.
- **mlp_training_portion** – specific for multi_layer_perceptron and pytorch_multi_layer_perceptron, the proportion of data to be used as training (default = 0.9) and the remaining part as test (default = 0.1).
- **mlp_hidden_layer_sizes** – specific for multi_layer_perceptron and pytorch_multi_layer_perceptron, list of values where each value defines the number of neurons in a hidden layer (e.g., [200, 100] for two hidden layers of 200 and 100 neurons respectively); default = [100].
- **mlp_alpha** – specific for multi_layer_perceptron and pytorch_multi_layer_perceptron, weight decay (also L2 regularization term) for Adam optimizer (default = 0.0001).
- **mlp_learning_rate_init** – specific for multi_layer_perceptron and pytorch_multi_layer_perceptron, sets initial learning rate (default = 0.001).
- **mlp_max_iter** – specific for multi_layer_perceptron and pytorch_multi_layer_perceptron, sets the maximum number of iterations (default = 200).
- **mlp_batch_size** – specific for multi_layer_perceptron and pytorch_multi_layer_perceptron, sets the number of samples per batch for optimizer; if “auto”, the batch is the minimum value between 200 and the number of samples (default = auto).
- **mlp_activation** – specific for multi_layer_perceptron and pytorch_multi_layer_perceptron, sets the activation function (default relu).
- **pytorch_model** – specific for pytorch_multi_layer_perceptron, custom pytorch nn.Module.
- **pytorch_optimizer** – specific for pytorch_multi_layer_perceptron, custom pytorch optimizer.
- **pytorch_loss_function** – specific for pytorch_multi_layer_perceptron, sets a custom loss function (default CrossEntropyLoss).
- **pytorch_optimization_n_iter_no_change** – specific for pytorch_multi_layer_perceptron, sets the maximum number of epochs where the loss is not improving by at least the value pytorch_optimization_tol (default 5).
- **pytorch_optimization_tol** – specific for pytorch_multi_layer_perceptron, sets the tolerance of optimization (default = 0.0001).
- **pytorch_device** – specific for pytorch_multi_layer_perceptron, processing device ‘cpu’ (default) or ‘cuda’ if available.
- **min_progress** – minimum progress value for *Progress()* (page 74).
- **max_progress** – maximum progress value for *Progress()* (page 74).

Returns

Classifier object.

Examples

Load a classifier

```
>>> import remotior_sensus
>>> rs = remotior_sensus.Session()
>>> signature_catalog = rs.spectral_signatures_catalog()
>>> classifier = Classifier.train(
>>> spectral_signatures=spectral_signatures,
>>> algorithm_name='minimum distance')
```

```

remotior_sensus.tools.band_classification.band_classification(input_bands: list | int |
    BandSet (page 39),
    output_path: str | None =
    None, overwrite: bool | None =
    False, spectral_signatures:
    SpectralSignaturesCatalog
    (page 78) | None = None,
    algorithm_name: str | None =
    None, bandset_catalog:
    BandSetCatalog (page 50) |
    None = None, macroclass:
    bool | None = True, threshold:
    bool | float | None = False,
    classification_confidence: bool
    | None = False,
    signature_raster: bool | None =
    False, n_processes: int | None
    = None, available_ram: int |
    None = None,
    cross_validation: bool | None
    = True, x_input: array | None
    = None, y_input: array | None
    = None, covariance_matrices:
    dict | None = None,
    input_normalization: str | None
    = None, load_classifier: str |
    None = None, save_classifier:
    bool | None = False, only_fit:
    bool | None = False,
    class_weight: None | str | dict
    = None,
    find_best_estimator=False,
    rf_max_features=None,
    rf_number_trees: int | None =
    100, rf_min_samples_split:
    None | int | float = None,
    svm_c: float | None = None,
    svm_gamma: float | str | None
    = None, svm_kernel: str | None
    = None, mlp_training_portion:
    None | float = None,
    mlp_hidden_layer_sizes: None
    | tuple | list = None, mlp_alpha:
    float | None = None,
    mlp_learning_rate_init: float |
    None = None, mlp_max_iter:
    float | None = None,
    mlp_batch_size: None | int | str
    = None, mlp_activation: None |
    str = None, pytorch_model:
    Optional = None,
    pytorch_optimizer: Optional =
    None, pytorch_loss_function:
    Optional = None, py-
    torch_optimization_n_iter_no_change:
    None | int = None,
    pytorch_optimization_tol:
    None | int = None,
    pytorch_device: None | str =
    None, progress_message: bool
    | None = True) → 17
    OutputManager (page 69)

```

Performs band classification.

This tool allows for classification of raster bands using the selected algorithm.

Parameters

- **input_bands** – list of input raster paths, or a BandSet number, or a previously defined BandSet.
- **output_path** – path of output file.
- **overwrite** – if True, output overwrites existing files.
- **spectral_signatures** – a SpectralSignaturesCatalog containing spectral signatures.
- **algorithm_name** – algorithm name selected from `cfg.classification_algorithms`; if None, minimum distance is used.
- **bandset_catalog** – BandSetCatalog object.
- **macroclass** – if True, use macroclass ID from ROIs or spectral signatures; if False use class ID.
- **threshold** – if False, classification without threshold; if True, use single threshold for each signature; if float, use this value as threshold for all the signature.
- **classification_confidence** – if True, write also additional classification confidence rasters as output.
- **signature_raster** – if True, write additional rasters for each spectral signature as output.
- **n_processes** – number of parallel processes.
- **available_ram** – number of megabytes of RAM available to processes.
- **cross_validation** – if True, perform cross validation for algorithms provided through scikit-learn (`random_forest`, `random_forest_ovr`, `support_vector_machine`, `multi_layer_perceptron`).
- **load_classifier** – path to a previously saved classifier.
- **x_input** – optional previously saved x matrix.
- **y_input** – optional previously saved y matrix.
- **covariance_matrices** – dictionary of previously calculated covariance matrices (used in `maximum_likelihood`).
- **input_normalization** – perform input normalization; options are `z_score` or `linear_scaling`.
- **only_fit** – perform only classifier fitting.
- **save_classifier** – save classifier to file.
- **class_weight** – specific for random forest and `support_vector_machine`, if None each class has equal weight 1, if ‘balanced’ weight is computed inversely proportional to class frequency.
- **find_best_estimator** – specific for scikit classifiers, if True, find automatically the best parameters and fit the model, if integer the greater the value the more are the tested combinations.
- **rf_max_features** – specific for random forest, if None all features are considered in node splitting, available options are ‘sqrt’ as square root of all the features, an integer number, or a float number for a fraction of all the features.
- **rf_number_trees** – specific for random forest, number of trees in the forest.

- **rf_min_samples_split** – specific for random forest through scikit, sets the minimum number of samples required to split an internal node; default = 2.
- **svm_c** – specific for support_vector_machine through scikit, sets the regularization parameter C; default = 1.
- **svm_gamma** – specific for support_vector_machine through scikit, sets the kernel coefficient; default = scale.
- **svm_kernel** – specific for support_vector_machine through scikit, sets the kernel; default = rbf.
- **mlp_training_portion** – specific for multi_layer_perceptron and pytorch_multi_layer_perceptron, the proportion of data to be used as training (default = 0.9) and the remaining part as test (default = 0.1).
- **mlp_hidden_layer_sizes** – specific for multi_layer_perceptron and pytorch_multi_layer_perceptron, list of values where each value defines the number of neurons in a hidden layer (e.g., [200, 100] for two hidden layers of 200 and 100 neurons respectively); default = [100].
- **mlp_alpha** – specific for multi_layer_perceptron and pytorch_multi_layer_perceptron, weight decay (also L2 regularization term) for Adam optimizer (default = 0.0001).
- **mlp_learning_rate_init** – specific for multi_layer_perceptron and pytorch_multi_layer_perceptron, sets initial learning rate (default = 0.001).
- **mlp_max_iter** – specific for multi_layer_perceptron and pytorch_multi_layer_perceptron, sets the maximum number of iterations (default = 200).
- **mlp_batch_size** – specific for multi_layer_perceptron and pytorch_multi_layer_perceptron, sets the number of samples per batch for optimizer; if “auto”, the batch is the minimum value between 200 and the number of samples (default = auto).
- **mlp_activation** – specific for multi_layer_perceptron and pytorch_multi_layer_perceptron, sets the activation function (default relu).
- **pytorch_model** – specific for pytorch_multi_layer_perceptron, custom pytorch nn.Module.
- **pytorch_optimizer** – specific for pytorch_multi_layer_perceptron, custom pytorch optimizer.
- **pytorch_loss_function** – specific for pytorch_multi_layer_perceptron, sets a custom loss function (default CrossEntropyLoss).
- **pytorch_optimization_n_iter_no_change** – specific for pytorch_multi_layer_perceptron, sets the maximum number of epochs where the loss is not improving by at least the value pytorch_optimization_tol (default 5).
- **pytorch_optimization_tol** – specific for pytorch_multi_layer_perceptron, sets the tolerance of optimization (default = 0.0001).
- **pytorch_device** – specific for pytorch_multi_layer_perceptron, processing device ‘cpu’ (default) or ‘cuda’ if available.
- **progress_message** – progress message.

Returns

If `only_fit` is `True` returns `OutputManager()` (page 69) object with

- `extra = {'classifier': classifier, 'model_path': output model path}`

If `only_fit` is `False` returns `OutputManager()` (page 69) object with

- `path = classification path`

- `extra = {'model_path': output model path}`

remotior_sensus.tools.band_combination module

Band combination.

This tool is intended for combining classifications in order to get a raster where each value corresponds to a combination of class values. A unique value is assigned to each combination of values. The output is a raster made of unique values corresponding to combinations of values. An output text file describes the correspondence between unique values and combinations, as well as the statistics of each combination.

Typical usage example:

```
>>> # import Remotior Sensus and start the session
>>> import remotior_sensus
>>> rs = remotior_sensus.Session()
>>> # start the process
>>> combination = rs.band_combination(input_bands=['path_1', 'path_2'],
... output_path='output_path')
```

`remotior_sensus.tools.band_combination.band_combination`(*input_bands*: list | int | BandSet (page 39), *output_path*: None | str = None, *nodata_value*: None | int = None, *overwrite*: bool | None = False, *n_processes*: None | int = None, *available_ram*: None | int = None, *bandset_catalog*: BandSetCatalog (page 50) | None = None, *extent_list*: list | None = None, *column_name_list*: list | None = None, *output_table*: bool | None = True, *progress_message*: bool | None = True) → *OutputManager* (page 69)

Calculation of band combination.

This tool allows for the combination of rasters or bands loaded in a BandSet. This tool is intended for combining classifications in order to get a raster where each value corresponds to a combination of class values. Input raster values must be integer type. The output is a combination raster and a text file reporting the statistics of each combination.

Parameters

- **input_bands** – list of paths of input rasters, or number of BandSet, or BandSet object.
- **output_path** – path of the output raster.
- **overwrite** – if True, output overwrites existing files.
- **nodata_value** – input value to be considered as nodata.
- **n_processes** – number of parallel processes.
- **available_ram** – number of megabytes of RAM available to processes.
- **bandset_catalog** – BandSetCatalog object required if `input_bands` is a BandSet number.
- **extent_list** – list of boundary coordinates left top right bottom.
- **column_name_list** – list of strings corresponding to input bands used as column names in output table, if None then column names are extracted for input band names.
- **output_table** – if True then calculate output table; if False then calculate only array of combinations and sum.

- **progress_message** – if True then start progress message; if False does not start the progress message (useful if launched from other tools).

Returns

If **output_table** is True returns the *OutputManager()* (page 69) object with

- paths = [output raster path, output table path]

If **output_table** is False returns the *OutputManager()* (page 69) object with

- paths = [virtual raster path]
- extra = { 'combinations': array of combinations, 'sums': array of the sums of values }

Examples

Combination using two rasters having paths path_1 and path_2

```
>>> combination = band_combination(input_bands=['path_1', 'path_2'], output_
↳ path='output_path')
```

The combination raster and the table are finally created; the paths can be retrieved from the output that is an *OutputManager()* (page 69) object

```
>>> raster_path, table_path = combination.paths
>>> print(raster_path)
output_path
```

Combination using a virtual raster as output file

```
>>> combination = band_combination(input_bands=['path_1', 'path_2'], output_
↳ path='output_path.vrt')
>>> raster_path, table_path = combination.paths
>>> print(raster_path)
output_path.vrt
```

Using input BandSet number

```
>>> catalog = BandSetCatalog()
>>> combination = band_combination(input_bands=1, output_path='output_path',
↳ bandset_catalog=catalog)
```

Using input BandSet

```
>>> catalog = BandSetCatalog()
>>> combination = band_combination(input_bands=catalog.get_bandset(1), output_
↳ path='output_path')
```

remotior_sensus.tools.band_dilation module

Band dilation.

This tool allows for the spatial dilation, through a moving window, of band pixels selected by values.

Typical usage example:

```
>>> # import Remotior Sensus and start the session
>>> import remotior_sensus
>>> rs = remotior_sensus.Session()
>>> # start the process
>>> dilation = rs.band_dilation(input_bands=['file1.tif', 'file2.tif'],
... value_list=[1], size=3, output_path='directory_path',
... circular_structure=True, prefix='dilation_')
```

`remotior_sensus.tools.band_dilation.band_dilation`(*input_bands*: list | int | [BandSet](#) (page 39), *value_list*: list, *size*: int, *output_path*: list | str | None = None, *overwrite*: bool | None = False, *circular_structure*: bool | None = None, *prefix*: str | None = "", *extent_list*: list | None = None, *n_processes*: None | int = None, *available_ram*: None | int = None, *bandset_catalog*: [BandSetCatalog](#) (page 50) | None = None, *virtual_output*: bool | None = None) → [OutputManager](#) (page 69)

Perform dilation of band pixels.

This tool performs the dilation of pixels identified by a list of values. A new raster is created for each input band.

Parameters

- **input_bands** – input of type [BandSet](#) or list of paths or integer number of [BandSet](#).
- **output_path** – string of output path directory or list of paths.
- **overwrite** – if True, output overwrites existing files.
- **value_list** – list of values for dilation.
- **size** – size of dilation in pixels.
- **virtual_output** – if True (and *output_path* is directory), save output as virtual raster of multiprocessing parts
- **circular_structure** – if True, use circular structure; if False, square structure.
- **prefix** – optional string for output name prefix.
- **extent_list** – list of boundary coordinates left top right bottom.
- **n_processes** – number of parallel processes.
- **available_ram** – number of megabytes of RAM available to processes.
- **bandset_catalog** – optional type [BandSetCatalog](#) for [BandSet](#) number

Returns

Object [OutputManager\(\)](#) (page 69) with

- *paths* = output list

Examples

Perform the dilation of size 5 for value 1 and 2

```
>>> dilation = band_dilation(input_bands=['path_1', 'path_2'], value_list=[1,
↪ 2], size=5, output_path='directory_path', circular_structure=True)
```

remotior_sensus.tools.band_erosion module

Band erosion.

This tool allows for the spatial erosion, through a moving window, of band pixels selected by values.

Typical usage example:

```
>>> # import Remotior Sensus and start the session
>>> import remotior_sensus
>>> rs = remotior_sensus.Session()
>>> # start the process
>>> dilation = rs.band_erosion(input_bands=['file1.tif', 'file2.tif'],
... value_list=[1], size=1, output_path='directory_path',
... circular_structure=True, prefix='erosion_')
```

`remotior_sensus.tools.band_erosion.band_erosion`(*input_bands*: list | int | [BandSet](#) (page 39), *value_list*: list, *size*: int, *output_path*: list | str | None = None, *overwrite*: bool | None = False, *circular_structure*: bool | None = None, *prefix*: str | None = "", *extent_list*: list | None = None, *n_processes*: None | int = None, *available_ram*: None | int = None, *bandset_catalog*: [BandSetCatalog](#) (page 50) | None = None, *virtual_output*: bool | None = None)

Perform erosion of band pixels.

This tool performs the erosion of pixels identified by a list of values. A new raster is created for each input band.

Parameters

- **input_bands** – input of type `BandSet` or list of paths or integer number of `BandSet`.
- **output_path** – string of output path directory or list of paths.
- **overwrite** – if True, output overwrites existing files.
- **value_list** – list of values for dilation.
- **size** – size of dilation in pixels.
- **virtual_output** – if True (and `output_path` is directory), save output as virtual raster of multiprocessing parts.
- **circular_structure** – if True, use circular structure; if False, square structure.
- **prefix** – optional string for output name prefix.
- **extent_list** – list of boundary coordinates left top right bottom.
- **n_processes** – number of parallel processes.
- **available_ram** – number of megabytes of RAM available to processes.
- **bandset_catalog** – optional type `BandSetCatalog` for `BandSet` number.

Returns

Object `OutputManager()` (page 69) with

- `paths` = output list

Examples

Perform the erosion of size 1 for value 1 and 2

```
>>> dilation = band_erosion(input_bands=['path_1', 'path_2'], value_list=[1, 2], size=1, output_path='directory_path', circular_structure=True)
```

remotior_sensus.tools.band_neighbor_pixels module

Band neighbor pixels.

This tool allows for the calculation of a function over neighbor pixels defined by size (i.e. number of pixels) or a structure.

Typical usage example:

```
>>> # import Remotior Sensus and start the session
>>> import remotior_sensus
>>> rs = remotior_sensus.Session()
>>> # start the process
>>> neighbor = rs.band_neighbor_pixels(
... input_bands=['file1.tif', 'file2.tif'],
... size=1, output_path='directory_path', stat_name='Mean',
... circular_structure=True, prefix='neighbor_')
```

`remotior_sensus.tools.band_neighbor_pixels.band_neighbor_pixels`(*input_bands*: list | int | BandSet (page 39), *size*: int, *output_path*: None | str = None, *overwrite*: bool | None = False, *stat_name*: str | None = None, *structure*: any | None = None, *circular_structure*: bool | None = True, *stat_percentile*: None | int | str = None, *output_data_type*: None | str = None, *virtual_output*: bool | None = None, *prefix*: str | None = "", *extent_list*: list | None = None, *n_processes*: None | int = None, *available_ram*: None | int = None, *bandset_catalog*: BandSetCatalog (page 50) | None = None) → *OutputManager* (page 69)

Performs band neighbor pixels.

This tool calculates a function over neighbor pixels defined by size (i.e. number of pixels) or structure. A new raster is created for each input band, where each pixel is the result of the calculation of the function over the neighbor pixels (e.g. the mean of the pixel values of a 3x3 window around the pixel). Available functions are:

- Count
- Max
- Mean
- Median
- Min

- Percentile
- StandardDeviation
- Sum

Parameters

- **input_bands** – input of type BandSet or list of paths or integer number of BandSet.
- **output_path** – string of output path directory or list of paths.
- **overwrite** – if True, output overwrites existing files.
- **size** – size of dilation in pixels.
- **structure** – optional path to csv file of structures, if None then the structure is created from size.
- **circular_structure** – if True use circular structure.
- **stat_percentile** – integer value for percentile parameter.
- **stat_name** – statistic name as in configurations.statistics_list.
- **output_data_type** – optional raster output data type, if None the data type is the same as input raster.
- **virtual_output** – if True (and output_path is directory), save output as virtual raster of multiprocessing parts.
- **prefix** – optional string for output name prefix.
- **extent_list** – list of boundary coordinates left top right bottom.
- **n_processes** – number of parallel processes.
- **available_ram** – number of megabytes of RAM available to processes.
- **bandset_catalog** – optional type BandSetCatalog for BandSet number.

Returns

Object [OutputManager\(\)](#) (page 69) with

- paths = output list

Examples

Perform the band neighbor of size 10 pixels with the function Sum

```
>>> neighbor = band_neighbor_pixels(input_bands=['file1.tif', 'file2.tif'],
↳ size=10, output_path='directory_path', stat_name='Sum', circular_
↳ structure=True)
```

remotior_sensus.tools.band_pca module

Band PCA.

This tool allows for the calculation of Principal Components Analysis on input bands, producing rasters corresponding to the principal components.

Typical usage example:

```
>>> # import Remotior Sensus and start the session
>>> import remotior_sensus
>>> rs = remotior_sensus.Session()
>>> catalog = rs.bandset_catalog()
>>> # create a BandSets
>>> file_list_1 = ['file1_b1.tif', 'file1_b2.tif', 'file1_b3.tif']
>>> catalog.create_bandset(file_list_1, bandset_number=1)
>>> # start the process
>>> output = rs.band_pca(input_bands=catalog.get_bandset(1),
... output_path='directory_path')
```

`remotior_sensus.tools.band_pca.band_pca`(*input_bands*: list | int | [BandSet](#) (page 39), *output_path*: list | str | None = None, *overwrite*: bool | None = False, *nodata_value*: None | int = None, *extent_list*: list | None = None, *n_processes*: None | int = None, *available_ram*: int | None = None, *bandset_catalog*: [BandSetCatalog](#) (page 50) | None = None, *number_components*: None | int = None, *progress_message*: bool | None = True) → [OutputManager](#) (page 69)

Calculation of Principal Components Analysis.

This tool allows for the calculation of Principal Components Analysis of raster bands obtaining the principal components. A new raster file is created for each component.

Parameters

- **input_bands** – input of type [BandSet](#) or list of paths or integer number of [BandSet](#).
- **output_path** – string of output path directory or list of paths.
- **overwrite** – if True, output overwrites existing files.
- **nodata_value** – value to be considered as nodata.
- **extent_list** – list of boundary coordinates left top right bottom.
- **n_processes** – number of parallel processes.
- **available_ram** – number of megabytes of RAM available to processes.
- **bandset_catalog** – optional type [BandSetCatalog](#) for [BandSet](#) number.
- **number_components** – defines the maximum number of components calculated.
- **progress_message** – if True then start progress message, if False does not start the progress message (useful if launched from other tools).

Returns

Object [OutputManager\(\)](#) (page 69) with

- paths = output list

Examples

Perform the PCA on the first BandSet

```
>>> # import Remotior Sensus and start the session
>>> import remotior_sensus
>>> rs = remotior_sensus.Session()
>>> catalog = rs.bandset_catalog()
>>> # create a BandSets
>>> file_list_1 = ['file1_b1.tif', 'file1_b2.tif', 'file1_b3.tif']
>>> catalog.create_bandset(file_list_1, bandset_number=1)
>>> # start the process
>>> pca = rs.band_pca(input_bands=catalog.get_bandset(1), output_path=
↳ 'directory_path')
```

remotior_sensus.tools.band_sieve module

Band sieve.

This tool allows for performing the sieve of raster bands removing patches having size lower than a threshold (i.e. number of pixels).

Typical usage example:

```
>>> # import Remotior Sensus and start the session
>>> import remotior_sensus
>>> rs = remotior_sensus.Session()
>>> # start the process
>>> sieve = rs.band_sieve(input_bands= ['file1.tif', 'file2.tif'], size=2
... output_path='directory_path', connected=False, prefix='sieve_')
```

`remotior_sensus.tools.band_sieve.band_sieve`(*input_bands*: list | int | [BandSet](#) (page 39), *size*: int, *output_path*: list | str | None = None, *connected*: bool | None = None, *overwrite*: bool | None = False, *prefix*: str | None = "", *extent_list*: list | None = None, *n_processes*: None | int = None, *available_ram*: None | int = None, *bandset_catalog*: [BandSetCatalog](#) (page 50) | None = None, *virtual_output*: bool | None = None) → [OutputManager](#) (page 69)

Perform band sieve.

This tool allows for performing the sieve of raster bands removing patches having size lower than a threshold (i.e. number of pixels).

Parameters

- **input_bands** – input of type `BandSet` or list of paths or integer number of `BandSet`.
- **output_path** – string of output path directory or list of paths.
- **overwrite** – if True, output overwrites existing files.
- **size** – size of dilation in pixels.
- **virtual_output** – if True (and `output_path` is directory), save output as virtual raster of multiprocessing parts
- **connected** – if True, consider 8 pixel connection; if False, consider 4 pixel connection.
- **prefix** – optional string for output name prefix.
- **extent_list** – list of boundary coordinates left top right bottom.
- **n_processes** – number of parallel processes.
- **available_ram** – number of megabytes of RAM available to processes.

- **bandset_catalog** – optional type BandSetCatalog for BandSet number

Returns

Object `OutputManager()` (page 69) with

- `paths` = output list

Examples

Perform the sieve of size 3 with connected pixel (8 connection)

```
>>> sieve = band_sieve(input_bands = ['file1.tif', 'file2.tif'], size=3, output_
↳ path='directory_path', connected=True, prefix='sieve_')
```

remotior_sensus.tools.cross_classification module

Cros classification.

This tool performs the cross classification which is similar to band combination, but it is executed between two files only. The reference file can also be of vector type. A unique value is assigned to each combination of values. The output is a raster made of unique values corresponding to combinations of values. An output text file describes the correspondance between unique values and combinations, as well as the statistics of each combination, with the option to calculate an error matrix or linear regression.

Typical usage example:

```
>>> # import Remotior Sensus and start the session
>>> import remotior_sensus
>>> rs = remotior_sensus.Session()
>>> # start the process
>>> cross = rs.cross_classification(classification_path='file1.tif', reference_path=
↳ 'file2.tif', output_path='output.tif')
```

```
remotior_sensus.tools.cross_classification.cross_classification(classification_path: str,
                                                                reference_path: str,
                                                                output_path: None | str =
                                                                None, overwrite: bool | None
                                                                = False, vector_field: None |
                                                                str = None, nodata_value:
                                                                None | int = None,
                                                                cross_matrix: bool | None =
                                                                False, regression_raster:
                                                                bool | None = False,
                                                                error_matrix: bool | None =
                                                                False, extent_list: list | None
                                                                = None, n_processes: None |
                                                                int = None, available_ram:
                                                                None | int = None) →
                                                                OutputManager (page 69)
```

Calculation of cross classification.

This tool allows for the cross classification of two files (a classification raster and a reference vector or raster) in order to get a raster where each value corresponds to a combination of class values. Input raster values must be integer type. The output is a cross raster and, depending on the parameters, a text file reporting the statistics of each combination, error matrix, or linear regression statistics.

Parameters

- **classification_path** – path of raster used as classification input.
- **reference_path** – path of the vector or raster file used as reference input.
- **vector_field** – in case of vector reference, the name of the field used as reference value.
- **output_path** – path of the output raster.
- **overwrite** – if True, output overwrites existing files.
- **nodata_value** – value to be considered as nodata.
- **cross_matrix** – if True then calculate the cross matrix.
- **regression_raster** – if True then calculate linear regression statistics.
- **error_matrix** – if True then calculate error matrix.
- **extent_list** – list of boundary coordinates left top right bottom.
- **n_processes** – number of parallel processes.
- **available_ram** – number of megabytes of RAM available to processes.

Returns

Object *OutputManager()* (page 69) with

- paths = [output raster path, output table path]

Examples

Perform the cross classification between two files

```
>>> cross = cross_classification(classification_path='file1.tif',reference_
↳ path='file2.tif',output_path='output.tif')
```

Perform the cross classification between two files and calculate the error matrix

```
>>> cross = cross_classification(classification_path='file1.tif',reference_
↳ path='file2.tif',output_path='output.tif',error_matrix=True)
```

remotior_sensus.tools.download_products module

Download products.

This tool allows for downloading products such as Landsat and Sentinel-2 datasets.

```
remotior_sensus.tools.download_products.download(product_table, output_path, exporter=False,
band_list=None, virtual_download=False,
extent_coordinate_list=None, proxy_host=None,
proxy_port=None, proxy_user=None,
proxy_password=None,
authentication_uri=None, user=None,
password=None, progress_message=True) →
OutputManager (page 69)
```

Download products.

This tool downloads product. Downloads Sentinel-2 images using the following Google service: <https://storage.googleapis.com/gcp-public-data-sentinel-2> . Downloads HLS images from: https://cmr.earthdata.nasa.gov/search/site/search_api_docs.html.

Parameters

- **product_table** –
- **output_path** –
- **exporter** –
- **band_list** –
- **virtual_download** – if True create a virtual raster of the linked image
- **extent_coordinate_list** – list of coordinates for defining a subset region [left, top, right, bottom]
- **proxy_host** – proxy host
- **proxy_port** – proxy port
- **proxy_user** – proxy user
- **proxy_password** – proxy password
- **authentication_uri** – authentication uri
- **user** – user for authentication
- **password** – password for authentication
- **progress_message** – progress message

Returns

object OutputManger

`remotior_sensus.tools.download_products.query_nasa_cmr`(*product, date_from, date_to, max_cloud_cover=100, result_number=50, name_filter=None, coordinate_list=None, progress_message=True, proxy_host=None, proxy_port=None, proxy_user=None, proxy_password=None*) → *OutputManager* (page 69)

Perform the query of NASA CMR.

This tool performs the query of NASA CMR Search https://cmr.earthdata.nasa.gov/search/site/search_api_docs.html.

Parameters

- **product** –
- **date_from** – date defining the starting period of the query
- **date_to** –
- **max_cloud_cover** –
- **result_number** –
- **name_filter** –
- **coordinate_list** –
- **progress_message** –
- **proxy_host** –
- **proxy_port** –
- **proxy_user** –
- **proxy_password** –

Returns

object `OutputManger`

`remotior_sensus.tools.download_products.query_sentinel_2_database`(*date_from*, *date_to*,
max_cloud_cover=100,
result_number=50,
name_filter=None,
coordinate_list=None,
progress_message=True)
 → *OutputManager*
 (page 69)

Perform the query of Sentinel-2 database.

This tool performs the query of Sentinel-2 database.

Query using Copernicus Data Space Ecosystem API <https://documentation.dataspace.copernicus.eu/#/APIs/OData> (from <https://documentation.dataspace.copernicus.eu>: ‘Copernicus Data Space Ecosystem represents an overall and comprehensive data ecosystem accessible via web portal, applications and APIs. ... The Copernicus Data Space Ecosystem provides the essential data and service offering for everyone to use, for commercial and non-commercial purposes’).

Sentinel-2 metadata are downloaded through the following Google service: <https://storage.googleapis.com/gcp-public-data-sentinel-2> .

Parameters

- **date_from** – date defining the starting period of the query
- **date_to** –
- **max_cloud_cover** –
- **result_number** –
- **name_filter** –
- **coordinate_list** – list [left, top, right, bottom] WGS84 coordinates
- **progress_message** – progress message

Returns

object *OutputManager*() (page 69)

`remotior_sensus.tools.download_products.search`(*product*, *date_from*, *date_to*,
max_cloud_cover=100, *result_number=50*,
name_filter=None, *coordinate_list=None*,
progress_message=True, *proxy_host=None*,
proxy_port=None, *proxy_user=None*,
proxy_password=None) → *OutputManager*
 (page 69)

Perform the query of image databases.

It allows for the search of image products, currently Landsat and Sentinel-2.

remotior_sensus.tools.mosaic module

Band mosaic.

This tool performs the mosaic of single bands, or multiple BandSets. Corresponding bands in two or more BandSet are merged into a single raster for each band, covering the extent of input bands. For instance, this is useful to mosaic several multiband images together, obtaining a multispectral mosaic of bands (e.g., the first band of the mosaic corresponds to all the first bands of input images).

Typical usage example:

```
>>> # import Remotior Sensus and start the session
>>> import remotior_sensus
>>> rs = remotior_sensus.Session()
>>> # start the process
>>> file_list = ['file1_b1.tif', 'file2_b1.tif']
>>> mosaic_bands = rs.mosaic(input_bands=file_list, output_path='output_directory')
```

`remotior_sensus.tools.mosaic.mosaic`(*input_bands*: list | int | BandSet (page 39), *output_path*: None | str = None, *overwrite*: bool | None = False, *prefix*: str | None = "", *nodata_value*: None | int = None, *n_processes*: None | int = None, *available_ram*: None | int = None, *virtual_output*: bool | None = False, *output_name*: None | str = None, *reference_raster_crs*: None | str = None, *bandset_catalog*: BandSetCatalog (page 50) | None = None) → *OutputManager* (page 69)

Mosaic bands.

This tool performs the mosaic of corresponding bands from multiple BandSets. A new raster is created for each band, named as `output_name` or first band name, and followed by band number.

Parameters

- **input_bands** – list of paths of input rasters, or number of BandSet, or BandSet object.
- **output_path** – string of output path directory.
- **overwrite** – if True, output overwrites existing files.
- **prefix** – optional string for output name prefix.
- **nodata_value** – value to be considered as nodata.
- **n_processes** – number of parallel processes.
- **available_ram** – number of megabytes of RAM available to processes.
- **virtual_output** – if True (and `output_path` is directory), save output as virtual raster of multiprocessing parts.
- **output_name** – string used as general name for all output bands.
- **reference_raster_crs** – path to a raster to be used as crs reference; if None, the first band is used as reference.
- **bandset_catalog** – optional type BandSetCatalog for BandSet number.

Returns

Object *OutputManager()* (page 69) with

- `paths` = output list

Examples

Perform the mosaic of three BandSets

```

>>> # import Remotior Sensus and start the session
>>> import remotior_sensus
>>> rs = remotior_sensus.Session()
>>> catalog = rs.bandset_catalog()
>>> # create three BandSets
>>> file_list_1 = ['file1_b1.tif', 'file1_b2.tif', 'file1_b3.tif']
>>> file_list_2 = ['file2_b1.tif', 'file2_b2.tif', 'file2_b3.tif']
>>> file_list_3 = ['file3_b1.tif', 'file3_b2.tif', 'file3_b3.tif']
>>> catalog.create_bandset(file_list_1, bandset_number=1)
>>> catalog.create_bandset(file_list_2, bandset_number=2)
>>> catalog.create_bandset(file_list_3, bandset_number=3)
>>> # start the process
>>> mosaic_bands = rs.mosaic(input_bands=[1, 2, 3],
... output_path='output_directory', bandset_catalog=catalog)

```

remotior_sensus.tools.preprocess_products module

Perform the preprocessing of products.

```

remotior_sensus.tools.preprocess_products.create_product_table(input_path,
                                                                metadata_file_path=None,
                                                                product=None,
                                                                nodata_value=None,
                                                                sensor=None,
                                                                acquisition_date=None)

```

```

remotior_sensus.tools.preprocess_products.perform_preprocess(product_table, output_path,
                                                             dos1_correction=False,
                                                             output_prefix="", n_processes:
                                                             int | None = None,
                                                             available_ram: int | None =
                                                             None, progress_message=True)
→ OutputManager (page 69)

```

Preprocess products.

Perform image conversion to reflectance of several products.

Can calculate DOS1 corrected reflectance (Sobrino, J. et al., 2004. Land surface temperature retrieval from LANDSAT TM 5. Remote Sensing of Environment, Elsevier, 90, 434-440) approximating path radiance to path reflectance for level 1 data: TOA reflectance = $DN * reflectance_scale + reflectance_offset$ path reflectance $p = DN_m - Dark\ Object\ reflectance = DN_m * reflectance_scale + reflectance_offset - 0.01$ land surface reflectance = TOA reflectance - $p = (DN * reflectance_scale) - (DN_m * reflectance_scale - 0.01)$

Landsat's data Collection 1 and 2 Level 1T Landsat 8-9 TOA reflectance proportional to exo-atmospheric solar irradiance in each band and the Earth-Sun distance (USGS, 2021. Landsat 8-9 Calibration and Validation (Cal/Val) Algorithm Description Document (ADD). Version 4.0. Department of the Interior U.S. Geological Survey, South Dakota) TOA reflectance with correction for the sun angle = $DN * Reflectance\ multiplicative\ scaling\ factor + Reflectance\ additive\ scaling\ factor / \sin(\text{Sun elevation})$ Level 2S Surface reflectance = $DN * Reflectance\ multiplicative\ scaling\ factor + Reflectance\ additive\ scaling\ factor$

Sentinel-2 data Level 1C TOA reflectance = $DN / QUANTIFICATION\ VALUE + OFFSET$ Level 2S Surface reflectance = $DN / QUANTIFICATION\ VALUE + OFFSET$

Parameters

- **product_table** –
- **output_path** –

- `dos1_correction` –
- `output_prefix` –
- `n_processes` –
- `available_ram` – number of megabytes of RAM available to processes.
- `progress_message` –

Returnsobject `OutputManager()` (page 69)

```
remotior_sensus.tools.preprocess_products.preprocess(input_path, output_path,  
                                                    metadata_file_path=None, product=None,  
                                                    nodata_value=None, sensor=None,  
                                                    acquisition_date=None,  
                                                    dos1_correction=False, output_prefix="",  
                                                    n_processes: int | None = None,  
                                                    available_ram: int | None = None,  
                                                    progress_message=True)
```

remotior_sensus.tools.raster_reclassification module

Raster reclassification.

This tool allows for the reclassification of a raster based on a reclassification table.

Typical usage example:

```
>>> # import Remotior Sensus and start the session  
>>> import remotior_sensus  
>>> rs = remotior_sensus.Session()  
>>> # start the process  
>>> reclassification = rs.raster_reclassification(raster_path='file1.tif',  
... output_path='output.tif',  
... reclassification_table=[[1, -10], ['nan', 6000]])
```

```
remotior_sensus.tools.raster_reclassification.raster_reclassification(raster_path: str,  
                                                                        output_path: None |  
                                                                        str = None,  
                                                                        overwrite: bool |  
                                                                        None = False, re-  
                                                                        classification_table:  
                                                                        list | array | None =  
                                                                        None, csv_path:  
                                                                        None | str = None,  
                                                                        separator: str | None  
                                                                        = ';',  
                                                                        output_data_type:  
                                                                        None | str = None,  
                                                                        extent_list: list |  
                                                                        None = None,  
                                                                        n_processes: None |  
                                                                        int = None,  
                                                                        available_ram: None  
                                                                        | int = None,  
                                                                        virtual_output: bool  
                                                                        | None = None) →  
                                                                        OutputManager  
                                                                        (page 69)
```

Performs raster reclassification.

This tool reclassifies a raster based on a reclassification table. The reclassification table is defined by two columns: old values and new values. Old values define the values of the raster to be reclassified to new values. Old values can be integer numbers or conditions selecting ranges of values.

Parameters

- **raster_path** – path of raster used as input.
- **output_path** – string of output path.
- **overwrite** – if True, output overwrites existing files.
- **reclassification_table** – table of values for reclassification or list of values [[old_value, new_value], ...]; if None, csv_path is used.
- **csv_path** – path to a csv file containing the reclassification table; used if reclassification_table is None.
- **separator** – separator character for csv file; default is comma separated.
- **output_data_type** – set output data type such as 'Float32' or 'Int32'; if None, input data type is used.
- **extent_list** – list of boundary coordinates left top right bottom.
- **n_processes** – number of parallel processes.
- **available_ram** – number of megabytes of RAM available to processes.
- **virtual_output** – if True (and output_path is directory), save output as virtual raster of multiprocessing parts.

Returns

Object [OutputManager\(\)](#) (page 69) with

- path = output path

Examples

Perform the reclassification using a csv file containing the reclassification values

```
>>> # import Remotior Sensus and start the session
>>> import remotior_sensus
>>> rs = remotior_sensus.Session()
>>> # start the process
>>> reclassification = rs.raster_reclassification(raster_path='file1.tif',
↪ output_path='output.tif', csv_path='file.csv')
```

```
remotior_sensus.tools.raster_reclassification.unique_values_table(raster_path: str,
                                                                    n_processes: None | int =
                                                                    None, available_ram:
                                                                    None | int = None,
                                                                    incremental: bool | None
                                                                    = False,
                                                                    progress_message: bool |
                                                                    None = True)
```

Calculate unique values from raster.

remotior_sensus.tools.raster_report module

Raster report.

This tool allows for the calculation of a report providing information extracted from a raster.

Typical usage example:

```
>>> # import Remotior Sensus and start the session
>>> import remotior_sensus
>>> rs = remotior_sensus.Session()
>>> # start the process
>>> report = rs.raster_report(raster_path='file.tif', output_path='report.csv')
```

`remotior_sensus.tools.raster_report.raster_report`(*raster_path: str, output_path: None | str = None, nodata_value: None | int = None, extent_list: list | None = None, n_processes: None | int = None, available_ram: None | int = None*)

Calculation of a report providing information extracted from a raster.

This tool allows for the calculation of a report providing information such as pixel count, area per class and percentage of the total area. The output is a csv file. This tool is intended for integer rasters.

Parameters

- **raster_path** – path of raster used as input.
- **output_path** – string of output path.
- **nodata_value** – value to be considered as nodata.
- **extent_list** – list of boundary coordinates left top right bottom.
- **n_processes** – number of parallel processes.
- **available_ram** – number of megabytes of RAM available to processes.

Returns

object `OutputManager()` (page 69) with

- path = output path

Examples

Perform the report of a raster

```
>>> raster_report(raster_path='file.tif', output_path='report.csv')
```

remotior_sensus.tools.raster_to_vector module

Raster to vector.

This tool allows for the conversion from raster to vector. A new geopackage is created from the raster conversion.

Typical usage example:

```
>>> # import Remotior Sensus and start the session
>>> import remotior_sensus
>>> rs = remotior_sensus.Session()
```

(continues on next page)

(continued from previous page)

```
>>> # start the process
>>> vector = rs.raster_to_vector(raster_path='file.tif', output_path='vector.gpkg')
)
```

`remotior_sensus.tools.raster_to_vector.raster_to_vector`(*raster_path*, *output_path*: *None* | *str* = *None*, *dissolve*: *bool* | *None* = *None*, *field_name*: *None* | *str* = *None*, *extent_list*: *list* | *None* = *None*, *n_processes*: *None* | *int* = *None*, *available_ram*: *None* | *int* = *None*) → *OutputManager* (page 69)

Performs the conversion from raster to vector.

This tool performs the conversion from raster to vector. Parallel processes are used for the conversion, resulting in a vector output which is split as many in portions as the process numbers. The argument `dissolve` allows for merging these portions, but it requires additional processing time depending on vector size.

Parameters

- **raster_path** – path of raster used as input.
- **output_path** – string of output path.
- **dissolve** – if True, dissolve adjacent polygons having the same values; if False, polygons are not dissolved and the process is rapider.
- **field_name** – name of the output vector field to store raster values (default = DN).
- **extent_list** – list of boundary coordinates left top right bottom.
- **n_processes** – number of parallel processes.
- **available_ram** – number of megabytes of RAM available to processes.

Returns

object *OutputManager*() (page 69) with

- `path` = output path

Examples

Perform the conversion to vector of a raster

```
>>> raster_to_vector(raster_path='file.tif', output_path='vector.gpkg')
```

remotior_sensus.tools.vector_to_raster module

Vector to raster.

This tool allows for the conversion from vector polygons to raster.

Typical usage example:

```
>>> # import Remotior Sensus and start the session
>>> import remotior_sensus
>>> rs = remotior_sensus.Session()
>>> # start the process
>>> vector = rs.vector_to_raster(vector_path='file.gpkg',
...                               output_path='vector.tif')
```

`remotior_sensus.tools.vector_to_raster.vector_to_raster`(*vector_path*, *align_raster*: *str* | [BandSet](#) (page 39) | *int*, *vector_field*: *None* | *str* = *None*, *constant*: *None* | *int* = *None*, *pixel_size*: *None* | *int* = *None*, *output_path*: *None* | *str* = *None*, *method*: *None* | *str* = *None*, *area_precision*: *int* | *None* = 20, *resample*='mode', *nodata_value*: *int* | *None* = 255, *minimum_extent*: *bool* | *None* = *True*, *extent_list*: *list* | *None* = *None*, *output_format*='GTiff', *compress*=*None*, *compress_format*=*None*, *n_processes*: *None* | *int* = *None*, *available_ram*: *None* | *int* = *None*, *bandset_catalog*: [BandSetCatalog](#) (page 50) | *None* = *None*) → [OutputManager](#) (page 69)

Performs the conversion from vector to raster.

This tool performs the conversion from vector polygons to raster.

Parameters

- **vector_path** – path of vector used as input.
- **align_raster** – optional string path of raster used for aligning output pixels and projections; it can also be a [BandSet](#) or an integer number of a [BandSet](#) in a [Catalog](#).
- **output_path** – string of output path.
- **vector_field** – the name of the field used as reference value.
- **constant** – integer value used as reference for all the polygons.
- **pixel_size** – size of pixel of output raster.
- **minimum_extent** – if *True*, raster has the minimum vector extent; if *False*, the extent is the same as the align raster.
- **extent_list** – list of boundary coordinates left top right bottom.
- **output_format** – output format, default *GTiff*
- **method** – method of conversion, default *pixel_center*, other methods are *all_touched* for burning all pixels touched or *area_based* for burning values based on area proportion.
- **area_precision** – for *area_based* method, the higher the value, the more is the precision in area proportion calculation.
- **resample** – type for resample when method is *area_based*.
- **compress** – if *True*, compress the output raster.
- **compress_format** – compress format.
- **nodata_value** – value to be considered as nodata.
- **n_processes** – number of parallel processes.
- **available_ram** – number of megabytes of RAM available to processes.
- **bandset_catalog** – [BandSetCatalog](#) object.

Returns

object [OutputManager\(\)](#) (page 69) with

- *path* = output path

Examples

Perform the conversion to raster of a vector

```
>>> vector_to_raster(vector_path='file.gpkg', output_path='vector.tif')
```

Module contents

1.5 Core Modules

The following are the core modules.

1.5.1 remotior_sensus.core package

Submodules

remotior_sensus.core.bandset_catalog module

```
class remotior_sensus.core.bandset_catalog.BandSet(bandset_uid, bands_list=None, name=None,
                                                    date=None, root_directory=None, crs=None,
                                                    box_coordinate_list=None, catalog=None)
```

Bases: object

Manages band sets.

This module allows for managing bands in a BandSet.

bands

BandSet of band tables.

get

alias for get_band function.

Examples

Create a BandSet

```
>>> import remotior_sensus
>>> rs = remotior_sensus.Session()
>>> file_list = ['file_1.tif', 'file_2.tif', 'file_3.tif']
>>> bandset = rs.bandset.create(file_list)
```

Get the first band

```
>>> band_1 = bandset.get_band(1)
```

```
__init__(bandset_uid, bands_list=None, name=None, date=None, root_directory=None, crs=None,
          box_coordinate_list=None, catalog=None)
```

Initializes a BandSet.

Initializes a BandSet defining a unique ID. One should use the create method for creating new BandSets.

Parameters

- **bandset_uid** – unique BandSet ID.
-

- **bands_list** – list of band tables.
- **name** – optional BandSet name.
- **date** – optional BandSet date.
- **root_directory** – optional BandSet root directory for relative path.
- **crs** – BandSet coordinate reference system.
- **box_coordinate_list** – list of coordinates [left, top, right, bottom] to create a virtual subset.
- **catalog** – BandSet Catalog

Examples

Initialize a BandSet

```
>>> BandSet.create(name=name, box_coordinate_list=box_coordinate_list)
```

add_new_band(*path: str, band_number: int | None = None, raster_band=None, band_name=None, date=None, root_directory=None, multiplicative_factor=None, additive_factor=None, wavelength=None, unit=None*)

Adds new band to a BandSet.

Adds a new band to a BandSet with the option to set the band number for the position in the BandSet order.

Parameters

- **path** – file path.
- **band_number** – sets the position of the band in BandSet order.
- **raster_band** – number of band for multiband rasters.
- **band_name** – name of raster used for band.
- **date** – single date (as YYYY-MM-DD).
- **multiplicative_factor** – multiplicative factor.
- **additive_factor** – additive factor.
- **wavelength** – center wavelength.
- **unit** – wavelength unit.
- **root_directory** – root directory for relative path.

Examples

Add a new band

```
>>> import remotior_sensus
>>> rs = remotior_sensus.Session()
>>> bandset = BandSet()
>>> bandset.add_new_band(
... path=path, band_number=band_number, raster_band=raster_band,
... band_name=band_name, date=date, root_directory=root_directory,
... multiplicative_factor=multiplicative_factor,
... additive_factor=additive_factor, wavelength=wavelength, unit=unit
... )
```

property box_coordinate_list

Optional BandSet box coordinate list for virtual subset.

Type

list

calc(*args, **kwargs)

Executes a calculation.

Executes a calculation directly from the BandSet, passing the argument `input_bands`. The arguments are related to the function. Bands in the BandSet can be referred in the expressions such as “b1” or “b2”; also band alias such as “#RED#” and expression alias such as #NDVI# can be used.

Parameters

kwargs – See `band_calc()` (page 8).

Returns

The output of the function.

Examples

Given that a session was previously started

```
>>> import remotior_sensus
>>> rs = remotior_sensus.Session()
>>> bandset = rs.bandset.create(
...     ['file1.tif', 'file2.tif'], wavelengths='Sentinel-2',
... )
```

Calculation of sum of the first two bands and saving the output in temporary directory

```
>>> output_object = bandset.calc("b1" + "b2")
```

Calculation of NDVI

```
>>> output_object = bandset.calc(output_path='output.tif',
...     expression_string='("#NIR#" - "#RED#") / ("#NIR#" + "#RED#")'
... )
```

classification(*args, **kwargs)

Executes a classification.

Executes a classification directly from the BandSet, passing the argument `input_bands`. The arguments are related to the function.

Parameters

kwargs – See `band_classification()` (page 10).

Returns

The output of the function.

combination(*args, **kwargs)

Band combination.

Combines classifications directly from the BandSet, in order to get a raster where each value corresponds to a combination of class values. The arguments are related to the function.

Parameters

kwargs – See `band_combination()` (page 20).

Returns

The output of the function.

Examples

Given that a session was previously started

```
>>> import remotior_sensus
>>> rs = remotior_sensus.Session()
>>> bandset = rs.bandset.create(['file1.tif', 'file2.tif'])
```

Combination using two rasters having paths `path_1` and `path_2`

```
>>> combination = bandset.combination(output_path='output_path')
```

classmethod `create`(*paths*: list | str | None = None, *band_names*: list | None = None, *wavelengths*: list | None = None, *unit*: None | str = None, *multiplicative_factors*: list | None = None, *additive_factors*: list | None = None, *dates*: list | None = None, *root_directory*: None | str = None, *box_coordinate_list*: list | None = None, *name*: None | str = None, *catalog*=None)

Creates a BandSet.

This method creates a BandSet defined by input files. Raster properties are derived from files to populate the table of bands.

Parameters

- **catalog** – BandSet Catalog object.
- **name** – BandSet name.
- **paths** – list of file paths or a string of directory path; also, a list of directory path and name filter is accepted.
- **band_names** – list of raster names used for identifying the bands, if None then the names are automatically extracted from file names.
- **wavelengths** – list of center wavelengths of bands or string of sensor names (also partial).
- **unit** – wavelength unit.
- **multiplicative_factors** – multiplicative factors for bands during calculations.
- **additive_factors** – additive factors for bands during calculations.
- **dates** – list of date strings, or single date string (format YYYY-MM-DD) or string defined in configurations `date_auto` to detect date from directory name.
- **root_directory** – root directory for relative path.
- **box_coordinate_list** – list of coordinates [left, top, right, bottom] to create a virtual subset.

Returns

returns BandSet.

Examples

Given that a session was previously started

```
>>> import remotior_sensus
>>> rs = remotior_sensus.Session()
>>> # set lists of files, names and center wavelength
>>> file = ['file_1.tif', 'file_2.tif', 'file_3.tif']
>>> names = ['name_1', 'name_2', 'name_3']
```

(continues on next page)

(continued from previous page)

```
>>> wavelength = [0.6, 0.7, 0.8]
>>> # create a BandSet
>>> bandset = rs.bandset.create(file, band_names=names,
↳ wavelength=wavelength)
```

Passing a directory with file name filter and the wavelenght from satellite name.

```
>>> bandset = rs.bandset.create(['directory_path', 'tif'], wavelength=
↳ 'Sentinel-2')
```

property crs

crs.

Type

str

property date

Optional date.

dilation(*args, **kwargs)

Band dilation.

Band dilation directly from the BandSet, passing the argument input_bands. The arguments are related to the function.

Parameters

kwargs – See *band_dilation()* (page 21).

Returns

The output of the function.

Examples

Given that a session was previously started

```
>>> import remotior_sensus
>>> rs = remotior_sensus.Session()
>>> bandset = rs.bandset.create(['file1.tif', 'file2.tif'])
```

Dilation

```
>>> dilation = bandset.dilation(output_path='directory_path',
... value_list=[1, 2], size=5)
```

erosion(*args, **kwargs)

Band erosion.

Band erosion directly from the BandSet, passing the argument input_bands. The arguments are related to the function.

Parameters

kwargs – See *band_erosion()* (page 23).

Returns

The output of the function.

Examples

Given that a session was previously started

```
>>> import remotior_sensus
>>> rs = remotior_sensus.Session()
>>> bandset = rs.bandset.create(['file1.tif', 'file2.tif'])
```

Erosion

```
>>> erosion = bandset.erosion(output_path='directory_path',
...                             value_list=[1, 2], size=1)
```

execute(function, *args, **kwargs)

Executes a function.

Executes a functions directly from the BandSet, passing the argument input_bands. The arguments are related to the function.

Parameters

function – the function to be executed.

Returns

The output of the function.

Examples

Given that a session was previously started

```
>>> import remotior_sensus
>>> rs = remotior_sensus.Session()
>>> bandset = rs.bandset.create(
...     ['file1.tif', 'file2.tif'], wavelengths='Sentinel-2',
... )
```

Calculation of sum of the first two bands

```
>>> output_object = bandset.execute(
...     rs.band_calc, output_path='output.tif', expression_string='b1"
↪ + "b2"'
... )
```

Calculation of NDVI

```
>>> output_object = bandset.execute(
...     rs.band_calc, output_path='output.tif',
...     expression_string='("#NIR#" - "#RED#") / ("#NIR#" + "#RED#")'
... )
```

Calculation of band combination

```
>>> output_object = bandset.execute(rs.band_combination, output_path=
↪ 'output.tif')
```

export_as_xml(output_path=None)

Exports a BandSet as xml.

Exports a BandSet bands and attributes.

Examples

Export a BandSet

```
>>> bandset = BandSet()
>>> # reset
>>> bandset.export_as_xml()
```

find_values_in_list(*attribute, value_list, output_attribute=None*) → list

Adds new band to a BandSet.

Finds BandSet values in a list and return a band attribute or band numbers.

Parameters

- **attribute** – attribute name for identification.
- **value_list** – attribute value list for indentification.
- **output_attribute** – attribute name of desired output; if None returns the band number.

Returns

returns list of band attributes.

Examples

Find values from wavelength attribute

```
>>> import remotior_sensus
>>> rs = remotior_sensus.Session()
>>> bandset = BandSet()
>>> bandset_values = bandset.find_values_in_list(
...     attribute='wavelength', value_list=[0.6, 0.7],
...     output_attribute='path'
... )
```

static generate_uid() → str

Generates unique ID for BandSet

Returns

returns table of bands.

get_absolute_paths() → list

Gets the list of absolute paths.

get_band(*number=None*) → array

Gets a band.

Gets a band by number.

Parameters

number – band number.

Returns

the band table; if number is None returns all the bands; returns None if band number is not found.

Examples

Get bands by number

```
>>> bandset = BandSet()
>>> # get first band
>>> band = bandset.get_band(1)
>>> # band x size
>>> band_x_size = band.x_size
>>> # get band nodata directly
>>> band_nodata = bandset.get_band(1).nodata
```

get_band_alias() → list

Gets bands alias (with band number) used in band_calc

Returns

the list of band aliases.

get_band_attributes(attribute: str | None = None) → list

Gets band attributes.

Gets an attribute of bands.

Parameters

attribute – attribute name.

Returns

returns list of attributes; if attribute is None returns the bands; if attribute is not found returns None.

Examples

Get band names

```
>>> bandset = BandSet()
>>> names = bandset.get_band_attributes('name')
```

get_band_by_wavelength(wavelength: float, threshold: float | None = None, output_as_number=False) → array

Gets band by wavelength.

Gets band by nearest wavelength.

Parameters

- **wavelength** – value of wavelength.
- **threshold** – threshold for wavelength identification, identifying bands within wavelength +/- threshold; if None, the nearest value is returned.
- **output_as_number** – if True returns the number of the band, if False returns the band table.

Returns

returns band table.

Examples

Get band by wavelength

```
>>> import remotior_sensus
>>> rs = remotior_sensus.Session()
>>> bandset = BandSet()
```

(continues on next page)

(continued from previous page)

```
>>> # get band by nearest wavelength
>>> band_x = bandset.get_band_by_wavelength(
>>> wavelength=0.7, threshold=0.1
>>> )
```

get_band_count() → int

Gets the count of bands.

get_bands_by_attributes(*attribute: str, attribute_value, output_as_number=False*) → array

Gets bands by attribute.

Gets bands identified by an attribute value (identified bands have the attribute value equal to the attribute_value).

Parameters

- **attribute** – attribute name.
- **attribute_value** – attribute value.
- **output_as_number** – if True returns the number of the band, if False returns the band table.

Returns

if output_number is True, returns list of band number; if output_number is False, returns bands identified by attribute; returns None if no band has the attribute value.

Examples

Get bands by attributes

```
>>> bandset = BandSet()
>>> band_x = bandset.get_bands_by_attributes('wavelength', 0.8)
```

get_paths() → list

Gets the list of bands.

get_raster_band_list() → list

Gets the list of raster bands.

get_wavelength_units() → list

Gets the list of wavelength units.

get_wavelengths() → list

Gets the list of center wavelength.

import_as_xml(*xml_path*)

Imports a BandSet as xml.

Imports a BandSet bands and attributes.

Examples

Import a BandSet

```
>>> bandset = BandSet()
>>> # reset
>>> bandset.import_as_xml('xml_path')
```

property name

Optional BandSet name.

Type

str

neighbor_pixels(*args, **kwargs)

Band neighbor pixels.

Band neighbor pixels directly from the BandSet, passing the argument `input_bands`. The arguments are related to the function.

Parameters

kwargs – See `band_neighbor_pixels()` (page 24).

Returns

The output of the function.

Examples

Given that a session was previously started

```
>>> import remotior_sensus
>>> rs = remotior_sensus.Session()
>>> bandset = rs.bandset.create(['file1.tif', 'file2.tif'])
```

Neighbor pixels

```
>>> neighbor = bandset.neighbor_pixels(output_path='directory_path',
... size=10, circular_structure=True, stat_name='Sum')
```

pca(*args, **kwargs)

Band PCA.

Band PCA directly from the BandSet, passing the argument `input_bands`. The arguments are related to the function.

Parameters

kwargs – See `band_pca()` (page 26).

Returns

The output of the function.

Examples

Given that a session was previously started

```
>>> import remotior_sensus
>>> rs = remotior_sensus.Session()
>>> bandset = rs.bandset.create(['file1.tif', 'file2.tif'])
```

PCA

```
>>> pca = bandset.pca(output_path='directory_path')
```

reset()

Resets a BandSet.

Resets a BandSet destroying bands and attributes.

Examples

Reset a BandSet

```
>>> bandset = BandSet()
>>> # reset
>>> bandset.reset()
```

property root_directory

Optional BandSet root directory for relative path.

Type

str

sieve(*args, **kwargs)

Band sieve.

Band sieve directly from the BandSet, passing the argument input_bands. The arguments are related to the function.

Parameters

kwargs – See [band_sieve\(\)](#) (page 27).

Returns

The output of the function.

Examples

Given that a session was previously started

```
>>> import remotior_sensus
>>> rs = remotior_sensus.Session()
>>> bandset = rs.bandset.create(['file1.tif', 'file2.tif'])
```

Sieve

```
>>> sieve = bandset.sieve(output_path='directory_path', size=3)
```

sort_bands_by_wavelength()

Sorts band order by wavelength

spectral_range_bands(output_as_number=True) → list

Gets bands from spectral range.

Gets bands from spectral range blue, green, red, nir, swir_1, swir_2.

Parameters

output_as_number – if True returns the number of the band, if False returns the band table.

Returns

returns list of bands.

Examples

Gets bands from spectral range

```
>>> import remotior_sensus
>>> rs = remotior_sensus.Session()
>>> bandset = BandSet()
>>> # spectral range bands
>>> (blue_band, green_band, red_band, nir_band, swir_1_band,
...  swir_2_band) = bandset.spectral_range_bands(
...  output_as_number=False)
```

property uid

Unique BandSet ID.

Type

str

class remotior_sensus.core.bandset_catalog.BandSetCatalog

Bases: object

Manages BandSets.

This class manages BandSets through a catalog, defining attributes and properties of BandSets.

bandsets

dictionary of the actual BandSets

bandsets_table

table of BandSets containing the properties thereof

get

alias for get_bandset

Examples**Create a BandSet Catalog**

```
>>> import remotior_sensus
>>> rs = remotior_sensus.Session()
>>> catalog = rs.bandset_catalog()
```

__init__()

Initializes the catalog with an empty BandSet

add_band_to_bandset(*path*: str, *bandset_number*: None | int = None, *band_number*: None | int = None, *raster_band*: None | int = None, *band_name*: None | str = None, *date*: None | str = None, *unit*: None | str = None, *root_directory*: None | str = None, *multiplicative_factor*: None | int = None, *additive_factor*: None | int = None, *wavelength*: None | float = None)

Adds a new band to BandSet.

This function creates a new band and adds it to a BandSet.

Parameters

- **path** – file path.
- **band_name** – raster name used for identifying the bands.
- **wavelength** – center wavelengths of band.
- **unit** – wavelength unit as string
- **multiplicative_factor** – multiplicative factor for bands during calculations.

- **additive_factor** – additive factors for band during calculations.
- **date** – date string (format YYYY-MM-DD).
- **bandset_number** – number of the BandSet; if None, the band is added to the current BandSet.
- **root_directory** – root directory for relative path.
- **raster_band** – raster band number.
- **band_number** – number of band in BandSet.

Examples

Add a band to BandSet 1.

```
>>> catalog = BandSetCatalog()
>>> catalog.add_band_to_bandset(
    path='file1.tif', bandset_number=1, band_number=1, raster_band=1
... )
```

add_bandset(*bandset*: [BandSet](#) (page 39), *bandset_number*: None | int = None, *insert*=False, *keep_uid*=False)

Adds a BandSet to Catalog.

This function adds a previously created BandSet to BandSet Catalog.

Parameters

- **bandset** – the BandSet to be added.
- **bandset_number** – number of BandSet; if None, current BandSet is used.
- **insert** – if True insert the BandSet at bandset_number (other BandSets are moved), if False replace the BandSet number.
- **keep_uid** – if True, keeps uid from the original bandset to the added one.

Examples

Insert a BandSet as BandSet 1.

```
>>> catalog = BandSetCatalog()
>>> catalog.add_bandset(bandset_number=1, insert=True
... )
```

clear_bandset(*bandset_number*)

Function to clear a BandSet.

create_band_string_list(*bandset_number*: int | None = None) → list

Creates band string list.

Creates band string list such as “bandset1b1” for all the bands in a BandSet. Used in tool [band_calc\(\)](#) (page 8).

Parameters

bandset_number – number of BandSet; if None, current BandSet is used.

Returns

List of band string such as [“bandset1b1”, “bandset1b2”].

Examples

Get BandSet 1 band string list.

```
>>> catalog = BandSetCatalog()
>>> catalog.create_band_string_list(1)
```

create_bandset(*paths: list | str | None = None, band_names: list | None = None, wavelengths: list | None = None, unit: None | str = None, multiplicative_factors: list | None = None, additive_factors: list | None = None, date: list | None = None, bandset_number: None | int = None, insert: bool | None = False, root_directory: None | str = None, bandset_name: None | str = None, box_coordinate_list: list | None = None*) → *BandSet* (page 39)

Creates a BandSet adding it to the catalog.

This function creates a BandSet adding it to the catalog, by inserting or replacing the BandSet at specific BandSet number. Wavelength is defined by providing a list of values for each band, or a string of sensors names as defined in configurations satWList such as:

- Sentinel-2;
- Landsat8;
- Landsat5;
- ASTER.

Wavelength unit is defined by a string such as:

- band number unitless band order as defined in the BandSet;
- μm (1 E-6m) micrometers;
- nm (1 E-9m) nanometers.

Parameters

- **paths** – list of file paths or a string of directory path; also, a list of directory path and name filter is accepted.
- **band_names** – list of raster names used for identifying the bands, if None then the names are automatically extracted from file names.
- **wavelengths** – list of center wavelengths of bands or string of sensor names (also partial).
- **unit** – wavelength unit as string
- **multiplicative_factors** – multiplicative factors for bands during calculations.
- **additive_factors** – additive factors for bands during calculations.
- **date** – list of date strings, or single date string (format YYYY-MM-DD) or string defined in configurations date_auto to detect date from directory name.
- **bandset_number** – number of the BandSet, replacing an existing BandSet with the same number.
- **insert** – if True insert the BandSet at bandset_number, if False replace the BandSet number.
- **root_directory** – root directory for relative path.
- **bandset_name** – name of the BandSet.
- **box_coordinate_list** – list of coordinates [left, top, right, bottom] to create a virtual subset.

Returns

The created BandSet.

Examples

Create a first BandSet from a file list with files inside a data directory, setting root_directory, defining the BandSet date, and the wavelength from satellite name.

```
>>> catalog = BandSetCatalog()
>>> file_list = ['file1.tif', 'file2.tif', 'file3.tif']
>>> bandset_date = '2021-01-01'
>>> data_directory = 'data'
>>> bandset = catalog.create_bandset(
... paths=file_list, wavelengths='Sentinel-2', date=bandset_date,
... root_directory=data_directory
... )
```

Create a new BandSet from a file list with files inside a data directory (setting root_directory), defining the BandSet date, and explicitly defining the BandSet number.

```
>>> catalog = BandSetCatalog()
>>> file_list = ['file1.tif', 'file2.tif', 'file3.tif']
>>> bandset_date = '2021-01-01'
>>> data_directory = 'data'
>>> bandset = catalog.create_bandset(
... paths=file_list, wavelengths='Sentinel-2', date=bandset_date,
... bandset_number=2, root_directory=data_directory
... )
```

Passing a directory with file name filter and the wavelength from satellite name.

```
>>> bandset = catalog.create_bandset(['directory_path', 'tif'],
↪wavelengths='Sentinel-2')
```

property current_bandset: int

Property that defines the current BandSet in the catalog.

This property identifies a BandSet number which is considered current (i.e. active) in several other tools when no BandSet is specified.

Returns

The integer number of current BandSet.

Examples

Get current BandSet.

```
>>> catalog = BandSetCatalog()
>>> bandset_number = catalog.current_bandset
>>> print(bandset_number)
1
```

Set current BandSet.

```
>>> catalog = BandSetCatalog()
>>> print(catalog.current_bandset)
1
```

(continues on next page)

(continued from previous page)

```
>>> catalog.current_bandset = 2
>>> print(catalog.current_bandset)
2
```

export_bandset_as_xml(*bandset_number*, *output_path=None*)

Function to export a BandSet as xml.

find_bandset_names_in_list(*names: list*, *lower=True*, *output_number=True*, *exact_match=False*)
→ list

Finds BandSet names.

This function finds BandSet names in a list and return BandSets
or BandSet numbers.

Parameters

- **names** – list of string names.
- **lower** – if True, finds by lowering all the names.
- **output_number** – if True returns the number of the band, if False returns the BandSet.
- **exact_match** – if True, names are compared by equality.

Returns

if *output_number* is True, returns list of band number; if *output_number* is False, returns list of BandSets.

Examples

Find BandSet from name list.

```
>>> catalog = BandSetCatalog()
>>> name_list = ['name1', 'name2']
>>> bandsets = catalog.find_bandset_names_in_list(names=name_list)
```

get_band_count(*bandset_number: None | int = None*) → None | int

Gets band count.

Gets band count of a BandSet.

Parameters

bandset_number – number of BandSet; if None, current BandSet is used.

Returns

Count of bands.

Examples

Get BandSet 1 band count.

```
>>> catalog = BandSetCatalog()
>>> catalog.get_band_count(1)
```

static get_band_list(*bandset: int | list | BandSet (page 39) = None*, *bandset_catalog: Optional = None*) → list

Gets band list.

This function gets band list from several types of input.

Parameters

- **bandset** – input of type BandSet or list of paths or integer number of BandSet.
- **bandset_catalog** – optional type BandSetCatalog if bandset argument is a number.

Returns

If argument bandset is a BandSet or a BandSet number,
returns list of bands in the BandSet; if argument bandset is already a list, returns the same list.

Examples

Get band list of BandSet 1.

```
>>> # import Remotior Sensus and start the session
>>> import remotior_sensus
>>> rs = remotior_sensus.Session()
>>> catalog = BandSetCatalog()
>>> list_1 = rs.bandset_catalog.get_band_list(1, catalog)
>>> # which is equivalent to
>>> bandset_1 = bandset_catalog.get_bandset(1)
>>> list_2 = bandset_1.get_absolute_paths()
```

get_bandset(bandset_number: int, attribute: None | str = None) → None | str | list | *BandSet* (page 39)

Get BandSet or BandSet attributes by number of the BandSet.

This function gets the BandSet or BandSet attributes by the number
of the BandSet.

Parameters

- **bandset_number** – number of the BandSet
- **attribute** – string of the attribute name, if None then the BandSet is returned

Returns

The BandSet band attributes or the BandSet.

Examples

Get the 'name' attribute of bands of the BandSet 1.

```
>>> catalog = BandSetCatalog()
>>> names = catalog.get_bandset(bandset_number=1,
attribute='name')
>>> print(names)
['file1', 'file2', 'file3']
```

Get the BandSet 1.

```
>>> catalog = BandSetCatalog()
>>> bandset_1 = catalog.get_bandset(bandset_number=1)
>>> print(bandset_1)
BandSet
```

`get_bandset_bands_by_attribute`(*bandset_number*: int, *attribute*: str, *attribute_value*: float | int | str, *output_number*: bool | None = False) → None | list | recarray

Get BandSet bands from the attributes thereof.

This function gets BandSet bands or the band number of bands whose attributes match an attribute value.

Parameters

- **bandset_number** – number of the BandSet
- **attribute** – string of the attribute name
- **attribute_value** – value of the band attribute to find
- **output_number** – if True then the output is the band number list, if False then the output is the band array

Returns

The band number list or the array of bands.

Examples

Get the ‘name’ attribute of bands of the BandSet 1.

```
>>> catalog = BandSetCatalog()
>>> band_number = catalog.get_bandset_bands_by_attribute(bandset_
↪ number=1, attribute='wavelength', attribute_value=0.443, output_
↪ number=True)
>>> print(band_number)
[2]
```

Get the BandSet 1.

```
>>> catalog = BandSetCatalog()
>>> band_x = catalog.get_bandset_bands_by_attribute(bandset_number=1,
↪ attribute='wavelength', attribute_value=0.443, output_number=True)
>>> print(band_x)
[(1, 1, '/data/file1.tif', '/data/file1.tif', 'file1', 0.443, 'µm (1 E-
↪ 6m)', ...)]
```

`get_bandset_by_name`(*bandset_name*: str, *output_number*: bool | None = False) → None | int | *BandSet* (page 39)

Get BandSet by name thereof.

This function gets the BandSet by the name thereof.

Parameters

- **bandset_name** – name of the BandSet
- **output_number** – if True then the output is the BandSet number, if False then the output is the BandSet

Returns

The BandSet identified by the name.

Examples

Get the number of the BandSet having the name ‘example’.

```
>>> catalog = BandSetCatalog()
>>> bandset_number = catalog.get_bandset_by_name(
bandset_name='example', output_number=True)
>>> print(bandset_number)
3
```

Get the BandSet having the name ‘example’.

```
>>> catalog = BandSetCatalog()
>>> bandset = catalog.get_bandset_by_name(
bandset_name='example', output_number=False)
>>> print(bandset)
BandSet object
```

get_bandset_by_number(*number: int*) → None | *BandSet* (page 39)

Get BandSet by number thereof.

This function gets the BandSet by the number thereof.

Parameters

number – number of the BandSet

Returns

The BandSet identified by the number.

Examples

Get the first BandSet.

```
>>> catalog = BandSetCatalog()
>>> bandset_1 = catalog.get_bandset_by_number(1)
>>> print(bandset_1)
BandSet object
```

get_bandset_catalog_attributes(*bandset_number: int, attribute: None | str = None*) → None | str | list

Get BandSet Catalog attributes by number of the BandSet.

This function gets the BandSet Catalog attributes by number of the BandSet.

Parameters

- **bandset_number** – number of the BandSet
- **attribute** – string of the attribute name, if None then the list of attributes is returned

Returns

The BandSet attribute or the list of attributes

(‘bandset_number’, ‘bandset_name’, ‘date’, ‘root_directory’, ‘uid’).

Examples

Get the ‘date’ attribute of the BandSet 1.

```
>>> catalog = BandSetCatalog()
>>> date = catalog.get_bandset_catalog_attributes(bandset_number=1,
↳ attribute='date')
>>> print(date)
2000-12-31
```

Get the list of attributes of the BandSet 1.

```
>>> catalog = BandSetCatalog()
>>> attributes = catalog.get_bandset_catalog_attributes(bandset_number=1,
↳ attributes)
>>> print(attributes)
[(1, 'example', '2000-12-31', 'None', '20000101_1605495327_293')]
```

get_bandset_count() → int

Gets count of BandSets in the catalog.

This function gets the count of BandSets present in the catalog.

Returns

The integer number of BandSets.

Examples

Count of BandSets present.

```
>>> catalog = BandSetCatalog()
>>> count = catalog.get_bandset_count()
>>> print(count)
1
```

get_bandsets_by_date(date_list: list, output_number: bool | None = False) → list

Get BandSets by date.

This function gets the BandSets by date.

Parameters

- **date_list** – list of date strings, or single date string (format YYYY-MM-DD); it can also include ranges using > or <, multiple conditions including &
- **output_number** – if True then the output is the BandSet number, if False then the output is the BandSet

Returns

The BandSet identified by the name.

Examples

Get the number of the BandSet by date.

```
>>> catalog = BandSetCatalog()
>>> bandset_number = catalog.get_bandsets_by_date(date_list=['2020-01-01
↳ '], output_number=True)
>>> print(bandset_number)
[1]
```

Get the number of the BandSet by range of dates.

```
>>> catalog = BandSetCatalog()
>>> bandset_number = catalog.get_bandsets_by_date(date_list=['2020-01-01
↳', '>=2021-01-01 & <=2022-01-02'], output_number=True)
>>> print(bandset_number)
[1, 3]
```

get_bandsets_by_list(*bandset_list*: list | None = None, *output_number*: bool | None = False) → list

Gets BandSets by list.

This function gets all the BandSets in the Catalog or filtered using a list of BandSet numbers.

Parameters

- **bandset_list** – list of integers BandSet numbers.
- **output_number** – if True, returns the list of BandSet number; if False, returns the list of BandSet object.

Returns

List of BandSets, or list of BandSets numbers if *output_number* is True.

Examples

Iterate BandSets.

```
>>> catalog = BandSetCatalog()
>>> bandset_t = catalog.get_bandsets_by_list()
>>> for bandset in bandset_t:
>>>     print(bandset)
```

get_box_coordinate_list(*bandset_number*: int | None = None) → None | list

Gets BandSet box coordinate list.

Gets BandSet box coordinate list.

Parameters

bandset_number – number of BandSet; if None, current BandSet is used.

Returns

List of box coordinates [left, top, right, bottom].

Examples

Get BandSet 1 band count.

```
>>> catalog = BandSetCatalog()
>>> catalog.get_box_coordinate_list(1)
```

get_date(*bandset_number*: None | int = None) → str

Gets BandSet date.

Gets BandSet date.

Parameters

bandset_number – number of BandSet; if None, current BandSet is used.

Returns

Date string.

Examples

Get BandSet 1 date.

```
>>> catalog = BandSetCatalog()
>>> catalog.get_date(1)
```

get_name(*bandset_number: None | int = None*) → str

Gets BandSet name.

Gets BandSet name.

Parameters

bandset_number – number of BandSet; if None, current BandSet is used.

Returns

BandSet name.

Examples

Get BandSet 1 date.

```
>>> catalog = BandSetCatalog()
>>> catalog.get_name(1)
```

get_root_directory(*bandset_number: None | int = None*) → str

Gets BandSet root directory.

Gets BandSet root directory.

Parameters

bandset_number – number of BandSet; if None, current BandSet is used.

Returns

Root directory string.

Examples

Get BandSet 1 root directory.

```
>>> catalog = BandSetCatalog()
>>> catalog.get_root_directory(1)
```

import_bandset_from_xml(*bandset_number, xml_path*)

Function to import a BandSet from xml.

iterate_bandset_bands(*attribute: str*) → list

Iterates BandSet attributes.

This function gets BandSet attributes iterating all the BandSets
in the Catalog.

Parameters

attribute – attribute name.

Returns

List of attributes

Examples

Get BandSet names.

```
>>> catalog = BandSetCatalog()
>>> names = catalog.iterate_bandset_bands('name')
```

move_band_in_bandset(*band_number_input: int, band_number_output: int, bandset_number: None | int = None, wavelength=True*)

Moves band in Bandset.

This function reorders a band in a Bandset.

Parameters

- **band_number_input** – number of band to be moved.
- **band_number_output** – position of the moved band.
- **bandset_number** – number of BandSet; if None, current BandSet is used.
- **wavelength** – if True, keep the wavelength attributes of the band order.

Examples

Move the second band of the BandSet 1 to position 5.

```
>>> catalog = BandSetCatalog()
>>> catalog.move_band_in_bandset(
...     bandset_number=1, band_number_input=2, band_number_output=5
... )
```

move_bandset(*bandset_number_input: int, bandset_number_output: int*)

Moves a BandSet.

This function reorders a BandSet in the BandSet Catalog.

Parameters

- **bandset_number_input** – the BandSet number to be moved.
- **bandset_number_output** – the new position of BandSet.

Examples

Move BandSet 1 to position 3.

```
>>> catalog = BandSetCatalog()
>>> catalog.move_bandset(bandset_number_input=1, bandset_number_output=3)
```

remove_band_in_bandset(*band_number: int, bandset_number: None | int = None*)

Removes band in Bandset.

This function removes a band in Bandset identified by a number. Automatically reorders other band numbers.

Parameters

- **band_number** – number of band to be removed.
- **bandset_number** – number of BandSet; if None, current BandSet is used.

Examples

Remove the second band of the BandSet.

```
>>> catalog = BandSetCatalog()
>>> catalog.remove_band_in_bandset(
... bandset_number=1, band_number=2
... )
```

`remove_bandset(bandset_number: int)`

Removes a BandSet.

This function removes a BandSet by the number thereof.

Parameters

bandset_number – the BandSet number to be removed.

Examples

Remove BandSet 2.

```
>>> catalog = BandSetCatalog()
>>> catalog.remove_bandset(2)
```

`reset()`

Resets the BandSets Catalog.

This function resets the BandSets Catalog removing all BandSets and creating an empty one.

Examples

Reset the BandSets catalog.

```
>>> catalog = BandSetCatalog()
>>> catalog.reset()
```

`set_box_coordinate_list(box_coordinate_list: list, bandset_number: None | int = None)`

Sets BandSet box coordinate list.

Sets BandSet box coordinate list for virtual subset.

Parameters

- **box_coordinate_list** – list of coordinates [left, top, right, bottom] to create a virtual subset.
 - **bandset_number** – number of BandSet; if None, current BandSet is used.
-

Examples

Set BandSet 1 coordinate list.

```
>>> catalog = BandSetCatalog()
>>> catalog.set_box_coordinate_list(
... bandset_number=1, box_coordinate_list=[230000, 4680000, 232000, ]
```

(continues on next page)

(continued from previous page)

```
↪4670000]
... )
```

set_date(*date*: str, *bandset_number*: None | int = None)

Sets BandSet date.

Sets BandSet date.

Parameters

- **date** – date string (format YYYY-MM-DD).
- **bandset_number** – number of BandSet; if None, current BandSet is used.

Examples

Set BandSet 1 date.

```
>>> catalog = BandSetCatalog()
>>> catalog.set_date(bandset_number=1, date='2021-01-01')
```

set_name(*name*: str, *bandset_number*: int | None = None)

Sets BandSet name.

Sets BandSet name.

Parameters

- **name** – name.
- **bandset_number** – number of BandSet; if None, current BandSet is used.

Examples

Set BandSet 1 name.

```
>>> catalog = BandSetCatalog()
>>> catalog.set_name(bandset_number=1, name='example')
```

set_root_directory(*root_directory*: str, *bandset_number*: None | int = None)

Sets BandSet root directory.

Sets BandSet root directory for relative path.

Parameters

- **root_directory** – root directory path.
- **bandset_number** – number of BandSet; if None, current BandSet is used.

Examples

Set BandSet 1 root directory.

```
>>> catalog = BandSetCatalog()
>>> catalog.set_root_directory(bandset_number=1, root_directory='data_
↪directory')
```

sort_bands_by_wavelength(*bandset_number: None | int = None*)

Sorts bands by wavelength.

This function numerically sorts bands in a BandSet by wavelength center.

Parameters

bandset_number – number of BandSet; if None, current BandSet is used.

Examples

Sort bands in BandSet 1.

```
>>> catalog = BandSetCatalog()
>>> catalog.sort_bands_by_wavelength(bandset_number=1)
```

remotior_sensus.core.configurations module

Configuration module. Module containing shared variables and parameters across tools.

remotior_sensus.core.configurations.multiprocess: [Multiprocess](#) (page 66)

remotior_sensus.core.log module

Logging manger.

Core class that manages logs during processes.

Typical usage example:

```
>>> # create a log file in a directory
>>> Log(directory='directory_path', level=10)
```

class remotior_sensus.core.log.Log(*file_path: None | str = None, directory: None | str = None, level: int | str | None = None, multiprocess=False, time=True*)

Bases: object

__init__(*file_path: None | str = None, directory: None | str = None, level: int | str | None = None, multiprocess=False, time=True*)

Manages logs.

This module allows for managing logs of processes.

file_path

path of a log file.

directory

directory path where a log file is created if file_path is None.

level

level of logging (10 for DEBUG, 20 for INFO).

multiprocess

if True, sets logging for parallel processes.

time

if True, time is saved in log file.

Examples

Create a log file and starts logging.

```
>>> Log(file_path='file.txt', level=20)
```

log = None

remotior_sensus.core.messages module

Manager of messages.

Console messages during processes.

Typical usage example:

```
>>> # display a warning message
>>> warning('warning message')
```

`remotior_sensus.core.messages.error(message: str)`

Error message.

Prints an error message.

Parameters

message – message.

Examples

Display a message

```
>>> error('error message')
```

`remotior_sensus.core.messages.info(message: str)`

Info message.

Prints an info message.

Parameters

message – message.

Examples

Display a message

```
>>> info('error message')
```

`remotior_sensus.core.messages.warning(message: str)`

Warning message.

Prints a warning message.

Parameters

message – message.

Examples

Display a message

```
>>> warning('warning message')
```

remotior_sensus.core.multiprocess_manager module

```
class remotior_sensus.core.multiprocess_manager.Multiprocess(n_processes: int | None = None, multiprocess_module=None)
```

Bases: object

```
__init__(n_processes: int | None = None, multiprocess_module=None)
```

```
create_warped_vrt(raster_path, output_path, output_wkt=None, align_raster_path=None, same_extent=False, n_processes: int | None = None, src_nodata=None, dst_nodata=None, extra_params=None)
```

```
find_minimum_dn()
```

```
gdal_copy_raster(input_raster, output, output_format='GTiff', compress=None, compress_format='DEFLATE', additional_params=', n_processes=1, available_ram: int | None = None, min_progress=None, max_progress=None)
```

```
gdal_warping(input_raster, output, output_format='GTiff', s_srs=None, t_srs=None, resample_method=None, raster_data_type=None, compression=None, compress_format='DEFLATE', additional_params=', n_processes: int | None = None, available_ram: int | None = None, src_nodata=None, dst_nodata=None, min_progress=None, max_progress=None)
```

```
get_dictionary_sum()
```

```
join_tables_multiprocess(table1, table2, field1_name, field2_name, nodata_value=None, join_type=None, postfix=None, n_processes: int | None = None, progress_message=None, min_progress=None, max_progress=None)
```

Parameters

- **table1** – input numpy table 1
- **table2** – input numpy table 2
- **field1_name** – input field table 1
- **field2_name** – input field table 2
- **nodata_value** – input nodata value
- **join_type** – join type
- **postfix** – postfix string
- **n_processes** – number of parallel processes.
- **progress_message** – progress message
- **min_progress** – minimum progress value
- **max_progress** – maximum progress value

```
multiprocess_raster_sieve(raster_path, n_processes: int | None = None, sieve_size=None, connected=None, output_nodata_value=None, output=None, output_data_type=None, compress=None, compress_format=None, available_ram: int | None = None, min_progress=0, max_progress=100)
```

multiprocess_raster_to_vector(*raster_path*, *output_vector_path*, *field_name=None*, *n_processes: int | None = None*, *dissolve_output=True*, *min_progress=0*, *max_progress=100*, *available_ram: int | None = None*)

multiprocess_roi_arrays()

multiprocess_spectral_signature()

multiprocess_sum_array(*nodata=None*)

multiprocess_unique_values()

multiprocess_vector_to_raster(*vector_path*, *field_name=None*, *output_path=None*, *reference_raster_path=None*, *output_format=None*, *nodata_value=None*, *background_value=None*, *burn_values=None*, *minimum_extent=None*, *x_y_size=None*, *all_touched=None*, *area_based=None*, *area_precision=None*, *compress=None*, *compress_format=None*, *available_ram: int | None = None*, *min_progress=0*, *max_progress=100*)

run(*raster_path*, *function=None*, *function_argument=None*, *function_variable=None*, *calculation_datatype=None*, *use_value_as_nodata=None*, *any_nodata_mask=True*, *output_raster_path=None*, *output_data_type=None*, *output_nodata_value=None*, *compress=None*, *compress_format='LZW'*, *n_processes: int | None = None*, *available_ram: int | None = None*, *dummy_bands=1*, *output_band_number=1*, *boundary_size=None*, *unique_section=False*, *keep_output_array=False*, *keep_output_argument=False*, *delete_array=True*, *scale=None*, *offset=None*, *input_nodata_as_value=None*, *classification=False*, *classification_confidence=False*, *signature_raster=False*, *virtual_raster=False*, *multi_add_factors=None*, *separate_bands=False*, *progress_message=None*, *device=None*, *min_progress=None*, *max_progress=None*)

Parameters

- **device** – processing device ‘cpu’ or ‘cuda’ if available.
- **classification_confidence** – if True, write also additional classification confidence rasters as output.
- **signature_raster** – if True, write additional rasters for each spectral signature as output.
- **raster_path** – input path.
- **multi_add_factors** – list of multiplicative and additive factors.
- **virtual_raster** – if True, create virtual raster output.
- **offset** – integer number of output offset.
- **scale** – integer number of output scale.
- **delete_array** – if True delete output array.
- **keep_output_argument** – if True keep output argument for post processing.
- **keep_output_array** – if True keep output array for post processing.
- **unique_section** – if True consider the whole raster as unique section.
- **dummy_bands** – integer number of dummy bands to be counted for calculating block size
- **available_ram** – integer value of RAM in MB.
- **any_nodata_mask** – True to apply the nodata where any input is nodata, False to apply nodata where all inputs are nodata, None not apply nodata
- **input_nodata_as_value** – consider nodata as value in calculation

- **output_data_type** – string of data type for output raster such as Float32 or Int16
- **output_band_number** – number of bands of the output
- **output_nodata_value** – output nodata value
- **compress_format** – string of format of raster compression
- **compress** – True to compress the output raster or False not to compress
- **boundary_size** – integer number of pixels used to extend the boundary of calculations
- **output_raster_path** – list of output path strings
- **calculation_datatype** – datatype use during calculation
- **function_variable** – list of variables for function
- **function_argument** – arguments of function
- **n_processes** – number of parallel processes.
- **function** – function name
- **classification** – if True, settings are defined for a classification output
- **use_value_as_nodata** – integer value as nodata in calculation
- **separate_bands** – if True, calculate a section for each raster range
- **progress_message** – progress message
- **min_progress** – minimum progress value
- **max_progress** – maximum progress value

run_iterative_process(*function_list, argument_list, min_progress=None, max_progress=None, n_processes=None*)

run_scikit(*function, classifier_list=None, list_train_argument_dictionaries=None, n_processes=None, available_ram: int | None = None, min_progress=None, max_progress=None*)

run_separated(*raster_path_list, function=None, function_argument=None, function_variable=None, calculation_datatype=None, use_value_as_nodata=None, any_nodata_mask=True, output_raster_list=None, output_data_type=None, output_nodata_value=None, compress=None, compress_format='LZW', n_processes: int | None = None, available_ram: int | None = None, output_band_number_list=None, boundary_size=None, dummy_bands=0, keep_output_array=False, keep_output_argument=False, scale=None, offset=None, input_nodata_as_value=None, multi_add_factors=None, progress_message=None, min_progress=None, max_progress=None*)

Parameters

- **multi_add_factors** – list of multiplicative and additive factors
- **offset** – list integer number of output offset
- **scale** – list of integer number of output scale
- **keep_output_argument** – if True keep output argument for post-processing
- **dummy_bands** – integer number of dummy bands to be counted for calculating block size
- **keep_output_array** – if True keep output array for post-processing
- **available_ram** – integer value of RAM in MB
- **any_nodata_mask** – True to apply the nodata where any input is nodata, False to apply nodata where all inputs are nodata, None not apply nodata

- **input_nodata_as_value** – consider nodata as value in calculation
- **output_data_type** – list of data type string for output raster such as Float32 or Int16
- **output_band_number_list** – list of number of bands of the output
- **output_nodata_value** – nodata value of the output
- **compress_format** – string of format of raster compression
- **compress** – True to compress the output raster or False not to compress
- **boundary_size** – integer number of pixels used to extend the boundary of calculations
- **output_raster_list** – list of output path strings
- **calculation_datatype** – datatype use during calculation
- **function_variable** – list of variables for function
- **function_argument** – arguments of function
- **n_processes** – number of parallel processes.
- **function** – function name
- **raster_path_list** – input path
- **use_value_as_nodata** – list of integer values as nodata in calculation
- **progress_message** – progress message
- **min_progress** – minimum progress value
- **max_progress** – maximum progress value

start(*n_processes, multiprocessing_module=None*)

stop()

remotior_sensus.core.output_manager module

Output manager.

Core class that manages several types of output, mainly intended for tools that have several outputs.

Typical usage example:

```
>>> # process output is checked
>>> OutputManager()
```

```
class remotior_sensus.core.output_manager.OutputManager(check: bool = True, path: str | None = None, paths: list | None = None, extra=None)
```

Bases: object

Manages output.

This class manages several types of output, mainly intended for tools that have several outputs. Check argument is False if output failed. Single output raster or multiple file paths can be defined as arguments. Additional output files or tables are managed with an extra argument. The type of the extra argument can be flexible depending on the process output.

check

True if output is as expected, False if process failed.

path

path of the first output.

paths

list of output paths in case of multiple outputs.

extra

additional output elements depending on the process.

Examples**Output failed**

```
>>> OutputManager(check=False)
```

Output is checked and file path is provided

```
>>> OutputManager(path='file.tif')
```

__init__(*check: bool = True, path: str | None = None, paths: list | None = None, extra=None*)

Initializes an Output.

Initializes an Output.

Parameters

- **check** – True if output is as expected, False if process failed.
- **path** – path of the first output.
- **paths** – list of output paths in case of multiple outputs.
- **extra** – additional output elements depending on the process.

Examples**Create an object with a single file path**

```
>>> OutputManager(path='file.tif')
```

Create an object with several output file paths in a list and an extra argument for a dictionary

```
>>> OutputManager(  
... paths=['file1.tif', 'file2.tif'],  
... extra=['additional_output', 'file.csv']  
... )
```

)

add_to_bandset(*bandset_catalog: BandSetCatalog (page 50), bandset_number=None, band_number=None, raster_band=None, band_name=None, date=None, unit=None, root_directory=None, multiplicative_factor=None, additive_factor=None, wavelength=None*)

Adds output to BandSet.

Adds the OutputManager.path as a band to a BandSet in a BandSetCatalog.

Parameters

- **bandset_catalog** – BandSetCatalog object.
- **band_name** – raster name used for identifying the bands.

- **wavelength** – center wavelengths of band.
- **unit** – wavelength unit as string
- **multiplicative_factor** – multiplicative factor for bands during calculations.
- **additive_factor** – additive factors for band during calculations.
- **date** – date string (format YYYY-MM-DD).
- **bandset_number** – number of the BandSet; if None, the band is added to the current BandSet.
- **root_directory** – root directory for relative path.
- **raster_band** – raster band number.
- **band_number** – number of band in BandSet.

Examples

Add the output to BandSet 1 as band 1.

```
>>> catalog = BandSetCatalog()
>>> OutputManager.add_to_bandset(
... bandset_catalog=catalog, bandset_number=1, band_number=1
... )
```

remotior_sensus.core.processor module

```
class remotior_sensus.core.processor.RasterPiece(section_list, x_min, y_min, x_max, y_max,
                                                x_size, y_size, y_min_no_boundary=None,
                                                y_max_no_boundary=None,
                                                y_size_boundary_top=None,
                                                y_size_boundary_bottom=None,
                                                y_size_no_boundary=None)
```

Bases: object

```
__init__(section_list, x_min, y_min, x_max, y_max, x_size, y_size, y_min_no_boundary=None,
         y_max_no_boundary=None, y_size_boundary_top=None, y_size_boundary_bottom=None,
         y_size_no_boundary=None)
```

Parameters

- **section_list** – list of sections
- **x_min** – left position
- **y_min** – top position
- **x_max** – right position
- **y_max** – bottom position
- **x_size** – piece x size
- **y_size** – piece y size
- **y_min_no_boundary** – top position without boundary
- **y_max_no_boundary** – bottom position without boundary
- **y_size_boundary_top** – top pixel buffer
- **y_size_boundary_bottom** – bottom pixel buffer

- **y_size_no_boundary** – piece size without buffer

```
class remotior_sensus.core.processor.RasterSection(x_min, y_min, x_max, y_max, x_size, y_size,
                                                    y_min_no_boundary=None,
                                                    y_max_no_boundary=None,
                                                    y_size_boundary_top=None,
                                                    y_size_boundary_bottom=None,
                                                    y_size_no_boundary=None)
```

Bases: object

```
__init__(x_min, y_min, x_max, y_max, x_size, y_size, y_min_no_boundary=None,
          y_max_no_boundary=None, y_size_boundary_top=None, y_size_boundary_bottom=None,
          y_size_no_boundary=None)
```

Parameters

- **x_min** – left position
- **y_min** – top position
- **x_max** – right position
- **y_max** – bottom position
- **x_size** – section x size
- **y_size** – section y size
- **y_min_no_boundary** – top position without boundary
- **y_max_no_boundary** – bottom position without boundary
- **y_size_boundary_top** – top pixel buffer
- **y_size_boundary_bottom** – bottom pixel buffer
- **y_size_no_boundary** – section size without buffer

y_details()

```
remotior_sensus.core.processor.array_multiplicative_additive_factors(array,
                                                                       multiplicative_factor,
                                                                       additive_factor)
```

```
remotior_sensus.core.processor.function_initiator(process_parameters=None,
                                                    input_parameters=None,
                                                    output_parameters=None, function=None,
                                                    function_argument=None,
                                                    function_variable=None,
                                                    run_separate_process=False,
                                                    classification=False,
                                                    classification_confidence=False,
                                                    signature_raster=False)
```

```
remotior_sensus.core.processor.gdal_translate(input_file=None, output=None,
                                               option_string=None, process_parameters=None)
```

```
remotior_sensus.core.processor.gdal_warp(input_file=None, output=None, option_string=None,
                                          process_parameters=None)
```

```
remotior_sensus.core.processor.raster_sieve_process(process_parameters=None,
                                                      input_parameters=None,
                                                      output_parameters=None)
```

```
remotior_sensus.core.processor.raster_to_vector_process(raster_path, output_vector_path,  
                                                    field_name=False,  
                                                    process_parameters=None,  
                                                    connected=4)
```

```
remotior_sensus.core.processor.table_join(table_1_parameters, table_2_parameters, nodata_value,  
                                         join_type, features, output_names, process_parameters)
```

```
remotior_sensus.core.processor.vector_to_raster(process_parameters=None,  
                                              input_parameters=None,  
                                              output_parameters=None)
```

remotior_sensus.core.processor_functions module

```
class remotior_sensus.core.processor_functions.ArrayLike(input_array)
```

Bases: ndarray

```
remotior_sensus.core.processor_functions.band_calculation(*argv)
```

```
remotior_sensus.core.processor_functions.bands_covariance(*argv)
```

```
remotior_sensus.core.processor_functions.calculate_pca(*argv)
```

```
remotior_sensus.core.processor_functions.classification_maximum_likelihood(*argv)
```

```
remotior_sensus.core.processor_functions.classification_minimum_distance(*argv)
```

```
remotior_sensus.core.processor_functions.classification_pytorch(*argv)
```

```
remotior_sensus.core.processor_functions.classification_scikit(*argv)
```

```
remotior_sensus.core.processor_functions.classification_spectral_angle_mapping(*argv)
```

```
remotior_sensus.core.processor_functions.cross_rasters(*argv)
```

```
remotior_sensus.core.processor_functions.fit_classifier(*argv)
```

```
remotior_sensus.core.processor_functions.get_band_arrays(*argv)
```

```
remotior_sensus.core.processor_functions.raster_dilation(*argv)
```

```
remotior_sensus.core.processor_functions.raster_erosion(*argv)
```

```
remotior_sensus.core.processor_functions.raster_neighbor(*argv)
```

```
remotior_sensus.core.processor_functions.raster_pixel_count(*argv)
```

```
remotior_sensus.core.processor_functions.raster_unique_values(*argv)
```

```
remotior_sensus.core.processor_functions.raster_unique_values_with_sum(*argv)
```

```
remotior_sensus.core.processor_functions.reclassify_raster(*argv)
```

```
remotior_sensus.core.processor_functions.score_classifier(*argv)
```

```
remotior_sensus.core.processor_functions.score_classifier_stratified(process,  
                                                                    progress_queue,  
                                                                    argument_list,  
                                                                    logger)
```

```
remotior_sensus.core.processor_functions.spectral_signature(*argv)
```

remotior_sensus.core.progress module

```
class remotior_sensus.core.progress.Progress(process=None, step=None, message=None,
                                             percentage=None, callback=None)
```

Bases: object

```
__init__(process=None, step=None, message=None, percentage=None, callback=None)
```

```
callback = None
```

```
elapsed_time = None
```

```
finish()
```

Ends progress and resets.

```
get()
```

```
message = 'starting'
```

```
percentage = False
```

```
ping = 0
```

```
previous_step = 0
```

```
previous_step_time = None
```

```
static print_progress(process=None, step=None, message=None, percentage=None,
                      elapsed_time=None, previous_step=None, start=None, end=None, ping=0)
```

```
static print_progress_replace(process=None, step=None, message=None, percentage=None,
                               elapsed_time=None, previous_step=None, start=None, end=None,
                               ping=0)
```

```
process = 'remotior_sensus'
```

```
remaining = ''
```

```
start_time = None
```

```
step = 0
```

```
update(process=None, step=None, message=None, percentage=None, start=None, end=None,
        ping=None, steps=None, minimum=None, maximum=None)
```

remotior_sensus.core.session module

Session manager.

Core class that manages the Remotior Sensus' session. It defines fundamental parameters such as available RAM and number of parallel processes. Creates a temporary directory to store temporary files. Exposes core functions and tools. It includes a method to close the session removing temporary files and stopping parallel processes.

Typical usage example:

```
>>> # start the session
>>> rs = Session()
>>> # optionally set session parameters
>>> rs.set(n_processors=2)
>>> # close the session when processing is done
>>> rs.close()
```

```
class remotior_sensus.core.session.Session(n_processes: int | None = 2, available_ram: int = 2048,
temporary_directory: str | None = None,
directory_prefix: str | None = None, log_level: int = 20,
log_time: bool = True, progress_callback=None,
multiprocess_module=None, messages_callback=None)
```

Bases: object

Manages system parameters.

This module allows for managing Remotior Sensus' session, setting fundamental processing parameters and exposing core functions and tools.

configurations

module containing shared variables and functions

bandset

access [BandSet\(\)](#) (page 39) class

bandset_catalog

access [BandSetCatalog\(\)](#) (page 50) class

spectral_signatures_catalog

access [SpectralSignaturesCatalog\(\)](#) (page 78) class

output_manager

access [OutputManager\(\)](#) (page 69) class

table_manager

access functions of [table_manager\(\)](#) (page 79) module

dates_times

access dates and times utilities

band_calc

tool [band_calc\(\)](#) (page 8)

band_classification

tool [band_classification\(\)](#) (page 10)

classifier

tool [Classifier\(\)](#) (page 10)

band_combination

tool [band_combination\(\)](#) (page 20)

band_dilation

tool [band_dilation\(\)](#) (page 21)

band_erosion

tool [band_erosion\(\)](#) (page 23)

band_neighbor_pixels

tool [band_neighbor_pixels\(\)](#) (page 24)

band_pca

tool [band_pca\(\)](#) (page 26)

band_sieve

tool [band_sieve\(\)](#) (page 27)

cross_classification

tool [cross_classification\(\)](#) (page 28)

download_productstool `download_products()` (page 29)**mosaic**tool `mosaic()` (page 32)**preprocess_products**tool `preprocess_products()` (page 33)**raster_reclassification**tool `preprocess_products()` (page 33)**raster_report**tool `raster_report()` (page 36)**raster_to_vector**tool `raster_to_vector()` (page 36)**vector_to_raster**tool `vector_to_raster()` (page 37)

Examples**Start a session**

```
>>> import remotior_sensus
>>> rs = remotior_sensus.Session()
```

Start a session defining number of parallel processes. and available RAM

```
>>> import remotior_sensus
>>> rs = remotior_sensus.Session(n_processes=4, available_ram=4096)
```

Create a `BandSetCatalog()` (page 50)

```
>>> catalog = rs.bandset_catalog()
```

Run the tool for raster report

```
>>> output = rs.raster_report(raster_path='file.tif', output_path='output.txt'
↵')
```

Stop a session at the end to clear temporary directory

```
>>> rs.close()
```

```
__init__(n_processes: int | None = 2, available_ram: int = 2048, temporary_directory: str | None =
None, directory_prefix: str | None = None, log_level: int = 20, log_time: bool = True,
progress_callback=None, multiprocessing_module=None, messages_callback=None)
```

Starts a session.

Starts a new session setting fundamental parameters for processing. It sets the number of parallel processes (default 2) and available RAM (default 2048MB) to be used in calculations. It starts the class Temporary to manage temporary files by creating a temporary directory with an optional name prefix. It starts the class Log for logging (with a default level INFO) and creates a logging formatter with the option to hide time. It starts the class Progress for displaying progress with a default callback function. A custom progress callback function can be passed optionally.

The sessions also allows for accessing to the core functions and tools.

In the end, the `close()` function should be called to clear the temporary directory and stop the parallel processes.

Parameters

- **n_processes** – number of parallel processes.
- **available_ram** – number of megabytes of RAM available to processes.
- **temporary_directory** – path to a temporary directory.
- **directory_prefix** – prefix of the name of the temporary directory.
- **log_level** – level of logging (10 for DEBUG, 20 for INFO).
- **log_time** – if True, logging includes the time.
- **progress_callback** – function for progress callback.
- **multiprocess_module** – multiprocessing module, useful if Remotior Sensus' session is started from another Python module.
- **messages_callback** – message module, useful if Remotior Sensus' session is started from another Python module.

Examples

Start a session

```
>>> import remotior_sensus
>>> rs = remotior_sensus.Session()
```

close(*log_path: str | None = None*)

Closes a Session.

This function closes current session by deleting the temporary files and stopping parallel processes.

Parameters

log_path – path where the log file is saved

Examples

Given that a session was previously started

```
>>> import remotior_sensus
>>> rs = remotior_sensus.Session()
```

Set the number of parallel processes and available RAM

```
>>> rs.set(n_processes=8, available_ram=20480)
```

Set the logging level to DEBUG

```
>>> rs.set(log_level=10)
```

set(*n_processes: int | None = None, available_ram: int | None = None, temporary_directory: str | None = None, directory_prefix: str | None = None, log_level: int | None = None, log_time: bool | None = None, progress_callback: function | None = None*)

Sets or changes the parameters of an existing Session.

Sets the parameters of an existing Session such as number of processes or temporary directory.

Parameters

- **n_processes** – number of parallel processes.
- **available_ram** – number of megabytes of RAM available to processes.
- **temporary_directory** – path to a temporary directory.
- **directory_prefix** – prefix of the name of the temporary directory.
- **log_level** – level of logging (10 for DEBUG, 20 for INFO).
- **log_time** – if True, logging includes the time.
- **progress_callback** – function for progress callback.

Examples

Given that a session was previously started

```
>>> import remotior_sensus
>>> rs = remotior_sensus.Session()
```

Stop a session

```
>>> rs.close()
```

Stop a session saving also the log to a file

```
>>> rs.close(log_path='file.txt')
```

remotior_sensus.core.spectral_signatures module

```
class remotior_sensus.core.spectral_signatures.SpectralSignaturesCatalog(bandset:
    BandSet
    (page 39) | None = None, catalog_table=None, geometry_file_path=None,
    macroclass_field=None, class_field=None)
```

Bases: object

A class to manage Spectral Signatures and ROIs.

```
__init__(bandset: BandSet (page 39) | None = None, catalog_table=None, geometry_file_path=None,
    macroclass_field=None, class_field=None)
```

```
add_spectral_signature(value_list, macroclass_id=None, class_id=None, macroclass_name=None,
    class_name=None, wavelength_list=None, standard_deviation_list=None,
    signature_id=None, selected=1, min_dist_thr=0, max_like_thr=0,
    spec_angle_thr=0)
```

Adds a spectral signature.

This method adds spectral signature to Spectral Signatures Catalog.

Parameters

- **value_list** –
- **macroclass_id** –

- `class_id` –
- `macroclass_name` –
- `class_name` –
- `wavelength_list` –
- `standard_deviation_list` –
- `signature_id` –
- `selected` –
- `min_dist_thr` –
- `max_like_thr` –
- `spec_angle_thr` –

Returns

object OutputManger

`calculate_signature(roi_path, n_processes: int | None = None)`

`import_spectral_signature_csv(csv_path, macroclass_id=None, class_id=None, macroclass_name=None, class_name=None, separator=',')`

`import_vector(file_path, macroclass_field, class_field, macroclass_name_field, class_name_field, calculate_signature=True)`

`signature_to_catalog(signature_id, macroclass_id, class_id, macroclass_name=None, class_name=None, selected=1, min_dist_thr=0, max_like_thr=0, spec_angle_thr=0)`

`remotior_sensus.core.spectral_signatures.generate_signature_id()`

remotior_sensus.core.table_manager module

Table manager.

This tool allows for managing table data as NumPy structured arrays. It includes functions for field calculation, join and pivot tables. Also, functions to manage tables used in other tools are included. Tables can be exported to csv files.

Typical usage example:

```
>>> # import Remotior Sensus and start the session
>>> import remotior_sensus
>>> rs = remotior_sensus.Session()
>>> # open a file
>>> file1 = 'file1.csv'
>>> table1 = rs.table_manager.open_file('file1.csv', field_names=['field1', 'field2'])
>>> # perform a calculation
>>> calculation = rs.table_manager.calculate(
... matrix=table1, expression_string='"field1" * 1.5',
... output_field_name='calc'
... )
>>> # export the table to csv
>>> rs.table_manager.export_table(
... matrix=calculation, output_path='output.csv',
... fields=['field1', 'calc'], separator=';', decimal_separator='.'
... )
```

```
remotior_sensus.core.table_manager.add_product_to_preprocess(product_list, spacecraft_list,  
processing_level,  
band_name_list,  
product_path_list, scale_list,  
offset_list, nodata_list, date_list,  
k1_list, k2_list,  
band_number_list, e_sun_list,  
sun_elevation_list,  
earth_sun_distance_list)
```

```
remotior_sensus.core.table_manager.add_spectral_signature_to_catalog_table(signature_id=None,  
macro-  
class_id=0,  
class_id=0,  
class_name=None,  
previ-  
ous_catalog=None,  
selected=1,  
min_dist_thr=0,  
max_like_thr=0,  
spec_angle_thr=0)
```

```
remotior_sensus.core.table_manager.append_field(matrix, field_name, data, data_type)
```

```
remotior_sensus.core.table_manager.append_tables(matrix1, matrix2)
```

```
remotior_sensus.core.table_manager.append_values_to_table(matrix, value_list)
```

```
remotior_sensus.core.table_manager.calculate(matrix, expression_string, output_field_name,  
progress_message=True)
```

Parameters

- **matrix** – input matrix
- **expression_string** – string used for calculation where fields must be named using double quotes e.g. "field_name" or using 'field.' such as field.field_name or 'matrix.' such as matrix.field_name
- **output_field_name** – string of output field name
- **progress_message** – optional, if True display process message

```
remotior_sensus.core.table_manager.calculate_multi(matrix, expression_string_list,  
output_field_name_list,  
progress_message=True)
```

Parameters

- **matrix** – input matrix
- **expression_string_list** – list of strings used for calculation where fields must be named using double quotes e.g. "field_name" or using 'field.' such as field.field_name or 'matrix.' such as matrix.field_name
- **output_field_name_list** – list of strings of output field names
- **progress_message** – optional, if True display process message

```
remotior_sensus.core.table_manager.columns(matrix)
```

```
remotior_sensus.core.table_manager.create_band_table(band_number=0, raster_band=None,
                                                    path=None, absolute_path=None,
                                                    name=None, wavelength=0,
                                                    wavelength_unit=None, additive_factor=0,
                                                    multiplicative_factor=1,
                                                    date='1900-01-01', x_size=0, y_size=0,
                                                    top=0, left=0, bottom=0, right=0,
                                                    x_count=0, y_count=0, nodata=None,
                                                    data_type=None, crs=None,
                                                    root_directory=None,
                                                    number_of_bands=0, x_block_size=0,
                                                    y_block_size=0, scale=None, offset=None)
```

```
remotior_sensus.core.table_manager.create_bandset_catalog_table(bandset_number=0,
                                                                root_directory=None,
                                                                date='NaT', bandset_uid=0,
                                                                bandset_name=None,
                                                                previous_catalog=None,
                                                                crs=None,
                                                                box_coordinate_left=None,
                                                                box_coordinate_top=None,
                                                                box_coordinate_right=None,
                                                                box_coordinate_bottom=None)
```

```
remotior_sensus.core.table_manager.create_bandset_table(band_list)
```

```
remotior_sensus.core.table_manager.create_product_table(product=None, product_id=None,
                                                        acquisition_date=None,
                                                        cloud_cover=None, zone_path=None,
                                                        row=None, min_lat=None,
                                                        min_lon=None, max_lat=None,
                                                        max_lon=None, collection=None,
                                                        size=None, preview=None, uid=None,
                                                        image=None)
```

```
remotior_sensus.core.table_manager.create_spectral_signature_table(value_list,
                                                                    wavelength_list, stan-
                                                                    dard_deviation_list=None)
```

```
remotior_sensus.core.table_manager.define_fields(matrix, field_list)
```

```
remotior_sensus.core.table_manager.export_table(matrix, output_path, fields=None, field_decimals:
                                                int | list = 2, separator=None,
                                                decimal_separator=None, nodata_value=None,
                                                nodata_value_output='nan',
                                                progress_message=True)
```

Parameters

- **matrix** – input matrix
- **output_path** – output path
- **fields** – optional list of fields to export
- **field_decimals** – integer number of decimals for decimal fields or list of integer values for decimal fields
- **separator** – separator of fields
- **decimal_separator** – optional decimal separator for float values replacing . character
- **nodata_value** – optional nodata value to be replaced

- **nodata_value_output** – optional string to replace nodata value in output
- **progress_message** –

`remotior_sensus.core.table_manager.find_nearest_value(array, field_name, value, threshold=None)`

`remotior_sensus.core.table_manager.get_values(matrix, value_field, conditional_string=None, progress_message=True)`

Parameters

- **matrix** – input matrix
- **value_field** – string of value field name
- **conditional_string** – optional string used for condition where field must be referred to using ‘field.’ such as field.field_name or ‘matrix.’ such as matrix.field_name
- **progress_message** – optional, if True display process message

`remotior_sensus.core.table_manager.join_matrices(matrix1, matrix2, field1_name, field2_name, join_type='leftouter', matrix1_postfix='_m1', matrix2_postfix='_m2', use_mask=False, progress_message=True)`

`remotior_sensus.core.table_manager.join_tables(table1, table2, field1_name, field2_name, postfix='2', nodata_value=None, join_type='left', n_processes: int | None = None, progress_message=True, min_progress=None, max_progress=None)`

`remotior_sensus.core.table_manager.matrix_to_csv(matrix, output_path, fields=None, field_decimals: int | list = 2, separator=None, decimal_separator=None, nodata_value=None, nodata_value_output='nan', progress_message=True)`

Parameters

- **matrix** – input matrix
- **output_path** – output path
- **fields** – optional list of fields to export
- **field_decimals** – integer number of decimals for decimal fields or list of integer values for decimal fields
- **separator** – separator of fields
- **decimal_separator** – optional decimal separator for float values replacing . character
- **nodata_value** – optional nodata value to be replaced
- **nodata_value_output** – optional string to replace nodata value in output
- **progress_message** –

`remotior_sensus.core.table_manager.open_file(file_path: str, separators: str | list | None = None, field_names: list | None = None, skip_first_line: bool | None = True, progress_message: bool | None = True) → recarray`

Opens a file.

Opens a file by reading the content and creating a table (which is a NumPy structured array). File formats csv and dbf are supported.

Parameters

- **file_path** – path of output file.
- **separators** – list of characters used as separator in csv files; default is tab and comma.
- **field_names** – list of strings to be used as field names.
- **skip_first_line** – skip the first line in csv files, in case the first line contains field names.
- **progress_message** – if True then start progress message; if False does not start the progress message (useful if launched from other tools).

Returns

Table as NumPy structured array.

Examples**Open a file**

```
>>> table_1 = open_file('file.csv')
```

```
remotior_sensus.core.table_manager.pivot_matrix(matrix, row_field, column_function_list,
                                                secondary_row_field_list=None,
                                                filter_string=None, nodata_value=-999,
                                                cross_matrix=None, field_names=False,
                                                progress_message=True)
```

Parameters

- **matrix** – matrix array
- **row_field** – field out_column_name of values used as rows
- **column_function_list** – list of lists of column out_column_name and function on values and optional out dtype
- **secondary_row_field_list** – optional list of fields to be used as secondary rows
- **filter_string** – optional string for filtering input matrix values
- **nodata_value** – optional value for nodata
- **cross_matrix** – optional, if True the output table field names are only combination values
- **field_names** – optional, if True returns field names without performing the pivot matrix
- **progress_message** – optional, if True display process message

```
remotior_sensus.core.table_manager.redefine_matrix_columns(matrix, input_field_names,
                                                          output_field_names=None,
                                                          progress_message=True)
```

Parameters

- **matrix** – matrix array
- **input_field_names** – list of field names to be included in output
- **output_field_names** – optional list of output field names with the same length as input_field_names
- **progress_message** – optional, if True display process message

```
remotior_sensus.core.table_manager.rename_field(matrix, old_field, new_field)
remotior_sensus.core.table_manager.replace_numpy_operators(expression)
remotior_sensus.core.table_manager.replace_variables(matrix, expression_string)
remotior_sensus.core.table_manager.sort_table_by_field(matrix, field_name)
remotior_sensus.core.table_manager.stack_product_table(product_list)
```

remotior_sensus.core.temporary module

```
class remotior_sensus.core.temporary.Temporary(temp_dir=None)
    Bases: object
    __init__(temp_dir=None)
    clear()
    classmethod create_root_temporary_directory(prefix=None, directory=None)
    create_temporary_directory()
    temporary_file_path(name_suffix=None, name_prefix=None, name=None, directory=None)
    temporary_raster_path(name=None, name_suffix=None, name_prefix=None, extension='.tif')
remotior_sensus.core.temporary.remove_root_temporary_directory(directory)
```

Module contents

1.6 API Reference

1.6.1 remotior_sensus package

Subpackages

Module contents

1.7 Basic Tutorials

The following tutorials are available.

1.7.1 Download Sentinel-2 Data and Calculate NDVI

Link to the guide: https://colab.research.google.com/gist/semiautomaticgit/e20479e1cbfbf76f078bfb4f211f23fa/download_sentinel-2_data_and_calculate_ndvi.ipynb

Except where otherwise noted, content of this work is licensed under a Creative Commons Attribution-ShareAlike 4.0 International License.

PYTHON MODULE INDEX

r

- remotior_sensus, 84
- remotior_sensus.core, 84
 - remotior_sensus.core.bandset_catalog, 39
 - remotior_sensus.core.configurations, 64
 - remotior_sensus.core.log, 64
 - remotior_sensus.core.messages, 65
 - remotior_sensus.core.multiprocess_manager, 66
 - remotior_sensus.core.output_manager, 69
 - remotior_sensus.core.processor, 71
 - remotior_sensus.core.processor_functions, 73
 - remotior_sensus.core.progress, 74
 - remotior_sensus.core.session, 74
 - remotior_sensus.core.spectral_signatures, 78
 - remotior_sensus.core.table_manager, 79
 - remotior_sensus.core.temporary, 84
- remotior_sensus.tools, 39
 - remotior_sensus.tools.band_calc, 8
 - remotior_sensus.tools.band_classification, 10
 - remotior_sensus.tools.band_combination, 20
 - remotior_sensus.tools.band_dilation, 21
 - remotior_sensus.tools.band_erosion, 23
 - remotior_sensus.tools.band_neighbor_pixels, 24
 - remotior_sensus.tools.band_pca, 26
 - remotior_sensus.tools.band_sieve, 27
 - remotior_sensus.tools.cross_classification, 28
 - remotior_sensus.tools.download_products, 29
 - remotior_sensus.tools.mosaic, 32
 - remotior_sensus.tools.preprocess_products, 33
 - remotior_sensus.tools.raster_reclassification, 34
 - remotior_sensus.tools.raster_report, 36
 - remotior_sensus.tools.raster_to_vector, 36
 - remotior_sensus.tools.vector_to_raster, 37

Symbols

- `__init__()` (*remotior_sensus.core.bandset_catalog.BandSet* method), 39
 - `__init__()` (*remotior_sensus.core.bandset_catalog.BandSetCatalog* method), 50
 - `__init__()` (*remotior_sensus.core.log.Log* method), 64
 - `__init__()` (*remotior_sensus.core.multiprocess_manager.Multiprocess* method), 66
 - `__init__()` (*remotior_sensus.core.output_manager.OutputManager* method), 70
 - `__init__()` (*remotior_sensus.core.processor.RasterPiece* method), 71
 - `__init__()` (*remotior_sensus.core.processor.RasterSection* method), 72
 - `__init__()` (*remotior_sensus.core.progress.Progress* method), 74
 - `__init__()` (*remotior_sensus.core.session.Session* method), 76
 - `__init__()` (*remotior_sensus.core.spectral_signatures.SpectralSignaturesCatalog* method), 78
 - `__init__()` (*remotior_sensus.core.temporary.Temporary* method), 84
 - `__init__()` (*remotior_sensus.tools.band_classification.Classifier* method), 11
- A**
- `add_band_to_bandset()` (*remotior_sensus.core.bandset_catalog.BandSetCatalog* method), 50
 - `add_bandset()` (*remotior_sensus.core.bandset_catalog.BandSetCatalog* method), 51
 - `add_new_band()` (*remotior_sensus.core.bandset_catalog.BandSet* method), 40
 - `add_product_to_preprocess()` (in module *remotior_sensus.core.table_manager*), 79
 - `add_spectral_signature()` (*remotior_sensus.core.spectral_signatures.SpectralSignaturesCatalog* method), 78
 - `add_spectral_signature_to_catalog_table()` (in module *remotior_sensus.core.table_manager*), 80
 - `add_to_bandset()` (*remotior_sensus.core.output_manager.OutputManager* method), 70
- B**
- `algorithm_name` (*remotior_sensus.tools.band_classification.Classifier* attribute), 10
 - `append_field()` (in module *remotior_sensus.core.table_manager*), 80
 - `append_tables()` (in module *remotior_sensus.core.table_manager*), 80
 - `append_values_to_table()` (in module *remotior_sensus.core.table_manager*), 80
 - `array_multiplicative_additive_factors()` (in module *remotior_sensus.core.processor*), 72
 - `ArrayLike` (class in *remotior_sensus.core.processor_functions*), 73
- B**
- `band_calc` (*remotior_sensus.core.session.Session* attribute), 75
 - `band_calc()` (in module *remotior_sensus.tools.band_calc*), 8
 - `band_calculation()` (in module *remotior_sensus.core.processor_functions*), 73
 - `band_classification` (*remotior_sensus.core.session.Session* attribute), 75
 - `band_classification()` (in module *remotior_sensus.tools.band_classification*), 16
 - `band_combination` (*remotior_sensus.core.session.Session* attribute), 75
 - `band_combination()` (in module *remotior_sensus.tools.band_combination*), 20
 - `band_dilation` (*remotior_sensus.core.session.Session* attribute), 75
 - `band_dilation()` (in module *remotior_sensus.tools.band_dilation*), 22
 - `band_erosion` (*remotior_sensus.core.session.Session* attribute), 75
 - `band_erosion()` (in module *remotior_sensus.tools.band_erosion*), 23
 - `band_neighbor_pixels` (*remotior_sensus.core.session.Session* attribute), 75

- `band_neighbor_pixels()` (in module `remotior_sensus.tools.band_neighbor_pixels`), 24
- `band_pca` (`remotior_sensus.core.session.Session` attribute), 75
- `band_pca()` (in module `remotior_sensus.tools.band_pca`), 26
- `band_sieve` (`remotior_sensus.core.session.Session` attribute), 75
- `band_sieve()` (in module `remotior_sensus.tools.band_sieve`), 27
- `bands` (`remotior_sensus.core.bandset_catalog.BandSet` attribute), 39
- `bands_covariance()` (in module `remotior_sensus.core.processor_functions`), 73
- `BandSet` (class in `remotior_sensus.core.bandset_catalog`), 39
- `bandset` (`remotior_sensus.core.session.Session` attribute), 75
- `bandset_catalog` (`remotior_sensus.core.session.Session` attribute), 75
- `BandSetCatalog` (class in `remotior_sensus.core.bandset_catalog`), 50
- `bandsets` (`remotior_sensus.core.bandset_catalog.BandSetCatalog` method), 77
- `bandsets_table` (`remotior_sensus.core.bandset_catalog.BandSetCatalog` attribute), 50
- `box_coordinate_list` (`remotior_sensus.core.bandset_catalog.BandSet` property), 40
- ## C
- `calc()` (`remotior_sensus.core.bandset_catalog.BandSet` method), 41
- `calculate()` (in module `remotior_sensus.core.table_manager`), 80
- `calculate_multi()` (in module `remotior_sensus.core.table_manager`), 80
- `calculate_pca()` (in module `remotior_sensus.core.processor_functions`), 73
- `calculate_signature()` (`remotior_sensus.core.spectral_signatures.SpectralSignaturesCatalog` method), 79
- `callback` (`remotior_sensus.core.progress.Progress` attribute), 74
- `check` (`remotior_sensus.core.output_manager.OutputManager` attribute), 69
- `classification()` (`remotior_sensus.core.bandset_catalog.BandSet` method), 41
- `classification_function` (`remotior_sensus.tools.band_classification.Classifier` attribute), 11
- `classification_maximum_likelihood()` (in module `remotior_sensus.core.processor_functions`), 73
- `classification_minimum_distance()` (in module `remotior_sensus.core.processor_functions`), 73
- `classification_pytorch()` (in module `remotior_sensus.core.processor_functions`), 73
- `classification_scikit()` (in module `remotior_sensus.core.processor_functions`), 73
- `classification_spectral_angle_mapping()` (in module `remotior_sensus.core.processor_functions`), 73
- `Classifier` (class in `remotior_sensus.tools.band_classification`), 10
- `classifier` (`remotior_sensus.core.session.Session` attribute), 75
- `clear()` (`remotior_sensus.core.temporary.Temporary` method), 84
- `clear_bandset()` (`remotior_sensus.core.bandset_catalog.BandSetCatalog` method), 51
- `close()` (`remotior_sensus.core.session.Session` method), 77
- `columns()` (in module `remotior_sensus.core.table_manager`), 80
- `combination()` (`remotior_sensus.core.bandset_catalog.BandSet` method), 41
- `configurations` (`remotior_sensus.core.session.Session` attribute), 75
- `covariance_matrices` (`remotior_sensus.tools.band_classification.Classifier` attribute), 10
- `create()` (`remotior_sensus.core.bandset_catalog.BandSet` class method), 42
- `create_band_string_list()` (`remotior_sensus.core.bandset_catalog.BandSetCatalog` method), 51
- `create_band_table()` (in module `remotior_sensus.core.table_manager`), 80
- `create_bandset()` (`remotior_sensus.core.bandset_catalog.BandSetCatalog` method), 52
- `create_bandset_catalog_table()` (in module `remotior_sensus.core.table_manager`), 81
- `create_bandset_table()` (in module `remotior_sensus.core.table_manager`), 81
- `create_product_table()` (in module `remotior_sensus.core.table_manager`), 81
- `create_product_table()` (in module `remotior_sensus.tools.preprocess_products`), 33
- `create_root_temporary_directory()` (`remotior_sensus.core.temporary.Temporary` class

- method), 84
- create_spectral_signature_table() (in module *remotior_sensus.core.table_manager*), 81
- create_temporary_directory() (remotior_sensus.core.temporary.Temporary method), 84
- create_warped_vrt() (remotior_sensus.core.multiprocess_manager.Multiprocess method), 66
- cross_classification (remotior_sensus.core.session.Session attribute), 75
- cross_classification() (in module *remotior_sensus.tools.cross_classification*), 28
- cross_rasters() (in module *remotior_sensus.core.processor_functions*), 73
- crs (remotior_sensus.core.bandset_catalog.BandSet property), 43
- current_bandset (remotior_sensus.core.bandset_catalog.BandSetCatalog property), 53
- ## D
- date (remotior_sensus.core.bandset_catalog.BandSet property), 43
- dates_times (remotior_sensus.core.session.Session attribute), 75
- define_fields() (in module *remotior_sensus.core.table_manager*), 81
- dilation() (remotior_sensus.core.bandset_catalog.BandSet method), 43
- directory (remotior_sensus.core.log.Log attribute), 64
- download() (in module *remotior_sensus.tools.download_products*), 29
- download_products (remotior_sensus.core.session.Session attribute), 75
- ## E
- elapsed_time (remotior_sensus.core.progress.Progress attribute), 74
- erosion() (remotior_sensus.core.bandset_catalog.BandSet method), 43
- error() (in module *remotior_sensus.core.messages*), 65
- execute() (remotior_sensus.core.bandset_catalog.BandSet method), 44
- export_as_xml() (remotior_sensus.core.bandset_catalog.BandSet method), 44
- export_bandset_as_xml() (remotior_sensus.core.bandset_catalog.BandSetCatalog method), 54
- export_table() (in module *remotior_sensus.core.table_manager*), 81
- extra (remotior_sensus.core.output_manager.OutputManager attribute), 70
- ## F
- file_path (remotior_sensus.core.log.Log attribute), 64
- find_bandset_names_in_list() (remotior_sensus.core.bandset_catalog.BandSetCatalog method), 54
- find_minimum_dn() (remotior_sensus.core.multiprocess_manager.Multiprocess method), 66
- find_nearest_value() (in module *remotior_sensus.core.table_manager*), 82
- find_values_in_list() (remotior_sensus.core.bandset_catalog.BandSet method), 45
- finish() (remotior_sensus.core.progress.Progress method), 74
- fit_classifier() (in module *remotior_sensus.core.processor_functions*), 73
- framework_name (remotior_sensus.tools.band_classification.Classifier attribute), 11
- function_initiator() (in module *remotior_sensus.core.processor*), 72
- ## G
- gdal_copy_raster() (remotior_sensus.core.multiprocess_manager.Multiprocess method), 66
- gdal_translate() (in module *remotior_sensus.core.processor*), 72
- gdal_warp() (in module *remotior_sensus.core.processor*), 72
- gdal_warping() (remotior_sensus.core.multiprocess_manager.Multiprocess method), 66
- generate_signature_id() (in module *remotior_sensus.core.spectral_signatures*), 79
- generate_uid() (remotior_sensus.core.bandset_catalog.BandSet static method), 45
- get (remotior_sensus.core.bandset_catalog.BandSet attribute), 39
- get (remotior_sensus.core.bandset_catalog.BandSetCatalog attribute), 50
- get() (remotior_sensus.core.progress.Progress method), 74
- get_absolute_paths() (remotior_sensus.core.bandset_catalog.BandSet method), 45
- get_band() (remotior_sensus.core.bandset_catalog.BandSet method), 45

- `get_band_alias()` (*remotior_sensus.core.bandset_catalog.BandSet* method), 46
- `get_band_arrays()` (in module *remotior_sensus.core.processor_functions*), 73
- `get_band_attributes()` (*remotior_sensus.core.bandset_catalog.BandSet* method), 46
- `get_band_by_wavelength()` (*remotior_sensus.core.bandset_catalog.BandSet* method), 46
- `get_band_count()` (*remotior_sensus.core.bandset_catalog.BandSet* method), 47
- `get_band_count()` (*remotior_sensus.core.bandset_catalog.BandSetCatalog* method), 54
- `get_band_list()` (*remotior_sensus.core.bandset_catalog.BandSetCatalog* static method), 54
- `get_bands_by_attributes()` (*remotior_sensus.core.bandset_catalog.BandSet* method), 47
- `get_bandset()` (*remotior_sensus.core.bandset_catalog.BandSetCatalog* method), 55
- `get_bandset_bands_by_attribute()` (*remotior_sensus.core.bandset_catalog.BandSetCatalog* method), 56
- `get_bandset_by_name()` (*remotior_sensus.core.bandset_catalog.BandSetCatalog* method), 56
- `get_bandset_by_number()` (*remotior_sensus.core.bandset_catalog.BandSetCatalog* method), 57
- `get_bandset_catalog_attributes()` (*remotior_sensus.core.bandset_catalog.BandSetCatalog* method), 57
- `get_bandset_count()` (*remotior_sensus.core.bandset_catalog.BandSetCatalog* method), 58
- `get_bandsets_by_date()` (*remotior_sensus.core.bandset_catalog.BandSetCatalog* method), 58
- `get_bandsets_by_list()` (*remotior_sensus.core.bandset_catalog.BandSetCatalog* method), 59
- `get_box_coordinate_list()` (*remotior_sensus.core.bandset_catalog.BandSetCatalog* method), 59
- `get_date()` (*remotior_sensus.core.bandset_catalog.BandSetCatalog* method), 59
- `get_dictionary_sum()` (*remotior_sensus.core.multiprocess_manager.Multiprocess* method), 66
- `get_name()` (*remotior_sensus.core.bandset_catalog.BandSetCatalog* method), 60
- `get_paths()` (*remotior_sensus.core.bandset_catalog.BandSet* method), 47
- `get_raster_band_list()` (*remotior_sensus.core.bandset_catalog.BandSet* method), 47
- `get_root_directory()` (*remotior_sensus.core.bandset_catalog.BandSetCatalog* method), 60
- `get_values()` (in module *remotior_sensus.core.table_manager*), 82
- `get_wavelength_units()` (*remotior_sensus.core.bandset_catalog.BandSet* method), 47
- `get_wavelengths()` (*remotior_sensus.core.bandset_catalog.BandSet* method), 47
- ## I
- `import_as_xml()` (*remotior_sensus.core.bandset_catalog.BandSet* method), 47
- `import_bandset_from_xml()` (*remotior_sensus.core.bandset_catalog.BandSetCatalog* method), 60
- `import_spectral_signature_csv()` (*remotior_sensus.core.spectral_signatures.SpectralSignaturesCatalog* method), 79
- `import_vector()` (*remotior_sensus.core.spectral_signatures.SpectralSignaturesCatalog* method), 79
- `info()` (in module *remotior_sensus.core.messages*), 65
- `input_normalization` (*remotior_sensus.tools.band_classification.Classifier* attribute), 11
- `iterate_bandset_bands()` (*remotior_sensus.core.bandset_catalog.BandSetCatalog* method), 60
- ## J
- `join_matrices()` (in module *remotior_sensus.core.table_manager*), 82
- `join_tables()` (in module *remotior_sensus.core.table_manager*), 82
- `join_tables_multiprocess()` (*remotior_sensus.core.multiprocess_manager.Multiprocess* method), 66
- ## L
- `level` (*remotior_sensus.core.log.Log* attribute), 64
- `load_classifier()` (*remotior_sensus.tools.band_classification.Classifier* class method), 12
- `Log` (class in *remotior_sensus.core.log*), 64
- `log` (*remotior_sensus.core.log.Log* attribute), 65
- ## M
- `matrix_to_csv()` (in module *remotior_sensus*), 66

- remotior_sensus.core.table_manager*), 82
- message (*remotior_sensus.core.progress.Progress* attribute), 74
- model_classifier (*remotior_sensus.tools.band_classification.Classifier* attribute), 10
- module
- remotior_sensus*, 84
 - remotior_sensus.core*, 84
 - remotior_sensus.core.bandset_catalog*, 39
 - remotior_sensus.core.configurations*, 64
 - remotior_sensus.core.log*, 64
 - remotior_sensus.core.messages*, 65
 - remotior_sensus.core.multiprocess_manager*, 66
 - remotior_sensus.core.output_manager*, 69
 - remotior_sensus.core.processor*, 71
 - remotior_sensus.core.processor_functions*, 73
 - remotior_sensus.core.progress*, 74
 - remotior_sensus.core.session*, 74
 - remotior_sensus.core.spectral_signatures*, 78
 - remotior_sensus.core.table_manager*, 79
 - remotior_sensus.core.temporary*, 84
 - remotior_sensus.tools*, 39
 - remotior_sensus.tools.band_calc*, 8
 - remotior_sensus.tools.band_classification*, 10
 - remotior_sensus.tools.band_combination*, 20
 - remotior_sensus.tools.band_dilation*, 21
 - remotior_sensus.tools.band_erosion*, 23
 - remotior_sensus.tools.band_neighbor_pixels*, 24
 - remotior_sensus.tools.band_pca*, 26
 - remotior_sensus.tools.band_sieve*, 27
 - remotior_sensus.tools.cross_classification*, 28
 - remotior_sensus.tools.download_products*, 29
 - remotior_sensus.tools.mosaic*, 32
 - remotior_sensus.tools.preprocess_products*, 33
 - remotior_sensus.tools.raster_reclassification*, 34
 - remotior_sensus.tools.raster_report*, 36
 - remotior_sensus.tools.raster_to_vector*, 36
 - remotior_sensus.tools.vector_to_raster*, 37
- mosaic (*remotior_sensus.core.session.Session* attribute), 76
- mosaic() (in module *remotior_sensus.tools.mosaic*), 32
- move_band_in_bandset() (*remotior_sensus.core.bandset_catalog.BandSetCatalog* method), 61
- move_bandset() (*remotior_sensus.core.bandset_catalog.BandSetCatalog* method), 61
- Multiprocess (class in *remotior_sensus.core.multiprocess_manager*), 66
- multiprocess (in module *remotior_sensus.core.configurations*), 64
- multiprocess (*remotior_sensus.core.log.Log* attribute), 64
- multiprocess_raster_sieve() (*remotior_sensus.core.multiprocess_manager.Multiprocess* method), 66
- multiprocess_raster_to_vector() (*remotior_sensus.core.multiprocess_manager.Multiprocess* method), 66
- multiprocess_roi_arrays() (*remotior_sensus.core.multiprocess_manager.Multiprocess* method), 67
- multiprocess_spectral_signature() (*remotior_sensus.core.multiprocess_manager.Multiprocess* method), 67
- multiprocess_sum_array() (*remotior_sensus.core.multiprocess_manager.Multiprocess* method), 67
- multiprocess_unique_values() (*remotior_sensus.core.multiprocess_manager.Multiprocess* method), 67
- multiprocess_vector_to_raster() (*remotior_sensus.core.multiprocess_manager.Multiprocess* method), 67
- ## N
- name (*remotior_sensus.core.bandset_catalog.BandSet* property), 47
- neighbor_pixels() (*remotior_sensus.core.bandset_catalog.BandSet* method), 48
- normalization_values (*remotior_sensus.tools.band_classification.Classifier* attribute), 11
- ## O
- open_file() (in module *remotior_sensus.core.table_manager*), 82
- output_manager (*remotior_sensus.core.session.Session* attribute), 75
- OutputManager (class in *remotior_sensus.core.output_manager*), 69
- ## P
- path (*remotior_sensus.core.output_manager.OutputManager* attribute), 69
- paths (*remotior_sensus.core.output_manager.OutputManager* attribute), 70

- `pca()` (*remotior_sensus.core.bandset_catalog.BandSet* method), 48
- `percentage` (*remotior_sensus.core.progress.Progress* attribute), 74
- `perform_preprocess()` (in module *remotior_sensus.tools.preprocess_products*), 33
- `ping` (*remotior_sensus.core.progress.Progress* attribute), 74
- `pivot_matrix()` (in module *remotior_sensus.core.table_manager*), 83
- `preprocess()` (in module *remotior_sensus.tools.preprocess_products*), 34
- `preprocess_products` (*remotior_sensus.core.session.Session* attribute), 76
- `previous_step` (*remotior_sensus.core.progress.Progress* attribute), 74
- `previous_step_time` (*remotior_sensus.core.progress.Progress* attribute), 74
- `print_progress()` (*remotior_sensus.core.progress.Progress* static method), 74
- `print_progress_replace()` (*remotior_sensus.core.progress.Progress* static method), 74
- `process` (*remotior_sensus.core.progress.Progress* attribute), 74
- `Progress` (class in *remotior_sensus.core.progress*), 74
- ## Q
- `query_nasa_cmr()` (in module *remotior_sensus.tools.download_products*), 30
- `query_sentinel_2_database()` (in module *remotior_sensus.tools.download_products*), 31
- ## R
- `raster_dilation()` (in module *remotior_sensus.core.processor_functions*), 73
- `raster_erosion()` (in module *remotior_sensus.core.processor_functions*), 73
- `raster_neighbor()` (in module *remotior_sensus.core.processor_functions*), 73
- `raster_pixel_count()` (in module *remotior_sensus.core.processor_functions*), 73
- `raster_reclassification` (*remotior_sensus.core.session.Session* attribute), 76
- `raster_reclassification()` (in module *remotior_sensus.tools.raster_reclassification*), 34
- `raster_report` (*remotior_sensus.core.session.Session* attribute), 76
- `raster_report()` (in module *remotior_sensus.tools.raster_report*), 36
- `raster_sieve_process()` (in module *remotior_sensus.core.processor*), 72
- `raster_to_vector` (*remotior_sensus.core.session.Session* attribute), 76
- `raster_to_vector()` (in module *remotior_sensus.tools.raster_to_vector*), 37
- `raster_to_vector_process()` (in module *remotior_sensus.core.processor*), 72
- `raster_unique_values()` (in module *remotior_sensus.core.processor_functions*), 73
- `raster_unique_values_with_sum()` (in module *remotior_sensus.core.processor_functions*), 73
- `RasterPiece` (class in *remotior_sensus.core.processor*), 71
- `RasterSection` (class in *remotior_sensus.core.processor*), 72
- `reclassify_raster()` (in module *remotior_sensus.core.processor_functions*), 73
- `redefine_matrix_columns()` (in module *remotior_sensus.core.table_manager*), 83
- `remaining` (*remotior_sensus.core.progress.Progress* attribute), 74
- `remotior_sensus` module, 84
- `remotior_sensus.core` module, 84
- `remotior_sensus.core.bandset_catalog` module, 39
- `remotior_sensus.core.configurations` module, 64
- `remotior_sensus.core.log` module, 64
- `remotior_sensus.core.messages` module, 65
- `remotior_sensus.core.multiprocess_manager` module, 66
- `remotior_sensus.core.output_manager` module, 69
- `remotior_sensus.core.processor` module, 71
- `remotior_sensus.core.processor_functions` module, 73
- `remotior_sensus.core.progress` module, 74
- `remotior_sensus.core.session` module, 74
- `remotior_sensus.core.spectral_signatures` module, 78
- `remotior_sensus.core.table_manager`

- module, 79
 - remotior_sensus.core.temporary
 - module, 84
 - remotior_sensus.tools
 - module, 39
 - remotior_sensus.tools.band_calc
 - module, 8
 - remotior_sensus.tools.band_classification
 - module, 10
 - remotior_sensus.tools.band_combination
 - module, 20
 - remotior_sensus.tools.band_dilation
 - module, 21
 - remotior_sensus.tools.band_erosion
 - module, 23
 - remotior_sensus.tools.band_neighbor_pixels
 - module, 24
 - remotior_sensus.tools.band_pca
 - module, 26
 - remotior_sensus.tools.band_sieve
 - module, 27
 - remotior_sensus.tools.cross_classification
 - module, 28
 - remotior_sensus.tools.download_products
 - module, 29
 - remotior_sensus.tools.mosaic
 - module, 32
 - remotior_sensus.tools.preprocess_products
 - module, 33
 - remotior_sensus.tools.raster_reclassification
 - module, 34
 - remotior_sensus.tools.raster_report
 - module, 36
 - remotior_sensus.tools.raster_to_vector
 - module, 36
 - remotior_sensus.tools.vector_to_raster
 - module, 37
 - remove_band_in_bandset() (remotior_sensus.core.bandset_catalog.BandSetCatalog method), 61
 - remove_bandset() (remotior_sensus.core.bandset_catalog.BandSetCatalog method), 62
 - remove_root_temporary_directory() (in module remotior_sensus.core.temporary), 84
 - rename_field() (in module remotior_sensus.core.table_manager), 83
 - replace_numpy_operators() (in module remotior_sensus.core.table_manager), 84
 - replace_variables() (in module remotior_sensus.core.table_manager), 84
 - reset() (remotior_sensus.core.bandset_catalog.BandSetCatalog method), 48
 - reset() (remotior_sensus.core.bandset_catalog.BandSetCatalog method), 62
 - root_directory (remotior_sensus.core.bandset_catalog.BandSetCatalog property), 49
 - run() (remotior_sensus.core.multiprocess_manager.Multiprocess method), 67
 - run_iterative_process() (remotior_sensus.core.multiprocess_manager.Multiprocess method), 68
 - run_prediction() (remotior_sensus.tools.band_classification.Classifier method), 12
 - run_scikit() (remotior_sensus.core.multiprocess_manager.Multiprocess method), 68
 - run_separated() (remotior_sensus.core.multiprocess_manager.Multiprocess method), 68
- ## S
- save_model() (remotior_sensus.tools.band_classification.Classifier method), 13
 - score_classifier() (in module remotior_sensus.core.processor_functions), 73
 - score_classifier_stratified() (in module remotior_sensus.core.processor_functions), 73
 - search() (in module remotior_sensus.tools.download_products), 31
 - Session (class in remotior_sensus.core.session), 74
 - set() (remotior_sensus.core.session.Session method), 77
 - set_box_coordinate_list() (remotior_sensus.core.bandset_catalog.BandSetCatalog method), 62
 - set_date() (remotior_sensus.core.bandset_catalog.BandSetCatalog method), 63
 - set_name() (remotior_sensus.core.bandset_catalog.BandSetCatalog method), 63
 - set_root_directory() (remotior_sensus.core.bandset_catalog.BandSetCatalog method), 63
 - sieve() (remotior_sensus.core.bandset_catalog.BandSetCatalog method), 49
 - signature_to_catalog() (remotior_sensus.core.spectral_signatures.SpectralSignaturesCatalog method), 79
 - sort_bands_by_wavelength() (remotior_sensus.core.bandset_catalog.BandSetCatalog method), 49
 - sort_bands_by_wavelength() (remotior_sensus.core.bandset_catalog.BandSetCatalog method), 63
 - sort_table_by_field() (in module remotior_sensus.core.table_manager), 84
 - spectral_range_bands() (remotior_sensus.core.bandset_catalog.BandSetCatalog method), 49
 - spectral_signature() (in module remotior_sensus.core.processor_functions),

- 73
- spectral_signatures (remotior_sensus.tools.band_classification.Classifier attribute), 10
- spectral_signatures_catalog (remotior_sensus.core.session.Session attribute), 75
- SpectralSignaturesCatalog (class in remotior_sensus.core.spectral_signatures), 78
- stack_product_table() (in module remotior_sensus.core.table_manager), 84
- start() (remotior_sensus.core.multiprocess_manager.Multiprocess method), 69
- start_time (remotior_sensus.core.progress.Progress attribute), 74
- step (remotior_sensus.core.progress.Progress attribute), 74
- stop() (remotior_sensus.core.multiprocess_manager.Multiprocess method), 69
- T**
- table_join() (in module remotior_sensus.core.processor), 73
- table_manager (remotior_sensus.core.session.Session attribute), 75
- Temporary (class in remotior_sensus.core.temporary), 84
- temporary_file_path() (remotior_sensus.core.temporary.Temporary method), 84
- temporary_raster_path() (remotior_sensus.core.temporary.Temporary method), 84
- time (remotior_sensus.core.log.Log attribute), 64
- train() (remotior_sensus.tools.band_classification.Classifier class method), 13
- U**
- uid (remotior_sensus.core.bandset_catalog.BandSet property), 50
- unique_values_table() (in module remotior_sensus.tools.raster_reclassification), 35
- update() (remotior_sensus.core.progress.Progress method), 74
- V**
- vector_to_raster (remotior_sensus.core.session.Session attribute), 76
- vector_to_raster() (in module remotior_sensus.core.processor), 73
- vector_to_raster() (in module remotior_sensus.tools.vector_to_raster), 37
- W**
- warning() (in module remotior_sensus.core.messages), 65
- y_details() (remotior_sensus.core.processor.RasterSection method), 72