GeoIPS ['geoips'] Documentation

Release 1.11.1.post4

U.S. NAVAL RESEARCH LABORATORY

Aug 21, 2023

CONTENTS

1 GeoIPS ® Base Package **3**

2 Introduction **5**
- 2.1 Description of GeoIPS . . . 5
  - 2.1.1 GeoIPS Overview . . . 5
  - 2.1.2 GeoIPS Scope . . . 7
  - 2.1.3 Using GeoIPS for Research . . . 7
  - 2.1.4 Using GeoIPS for Operations . . . 7
- 2.2 Summary of current Functionality . . . 8
- 2.3 Example Output Images . . . 8
- 2.4 Code of Conduct . . . 9

3 Getting Started **11**
- 3.1 Conda-based Installation . . . 11
  - 3.1.1 Complete Local conda-based GeoIPS Installation . . . 11
- 3.2 Expert User Installation (Administrative privileges) . . . 14
  - 3.2.1 Expert User GeoIPS Installation . . . 15
- 3.3 Simple command line examples . . . 16
  - 3.3.1 Full Install . . . 16
  - 3.3.2 Step by Step . . . 16
- 3.4 Extending GeoIPS with your own functionality . . . 17
  - 3.4.1 Discussion . . . 17

4 User Guide **19**
- 4.1 Description of GeoIPS structure . . . 19
  - 4.1.1 Overview of interfaces and plugins . . . 19
  - 4.1.2 Module-based interfaces . . . 23
  - 4.1.3 YAML-based Interfaces . . . 23
- 4.2 Command Line Interface . . . 24
- 4.3 GeoIPS Functionality . . . 24
- 4.4 Extend GeoIPS with Plugins . . . 26
  - 4.4.1 Developing Module-based plugin . . . 26
  - 4.4.2 Developing YAML-based plugin . . . 26

4.4.3 Example Module-based Plugins . . . 26
4.4.4 Example YAML-based Plugins . . . 26

5 Developer Guide **29**
5.1 Software Requirements Specification . . . 29
5.1.1 Table of Contents . . . 30
5.1.2 Revision History . . . 31
5.2 Contributors Guide . . . 41
5.3 Setting up for development . . . 42
5.3.1 Instructions for setting up a new data type . . . 42
5.4 Building documentation . . . 45
5.5 GeoIPS git workflow . . . 45
5.5.1 GeoIPS GitHub Issue Creation Workflow . . . 45
5.5.2 GeoIPS command line workflow . . . 47
5.5.3 GeoIPS GitHub Pull Request workflow . . . 48
5.5.4 GeoIPS Merge PR and Close Issue workflow . . . 49
5.6 Documentation and Style Strategy . . . 50
5.6.1 GeoIPS Syntax and Style Checking . . . 50
5.7 Xarray and NetCDF Metadata Standards . . . 51
5.7.1 Xarray Standard Variables . . . 51
5.7.2 Xarray Standard Attributes . . . 51
5.7.3 NetCDF CF Standards . . . 52

6 API Reference **55**
6.1 geoips package . . . 55
6.1.1 Subpackages . . . 55
6.1.2 Submodules . . . 276
6.1.3 geoips.cli module . . . 276
6.1.4 geoips.compare_outputs module . . . 277
6.1.5 geoips.errors module . . . 282
6.1.6 geoips.geoips_utils module . . . 282
6.1.7 Module contents . . . 286

7 Contact **289**
7.1 About Us . . . 289

Python Module Index **291**

Index **297**

Date: Aug 21, 2023 **Version**: 1.11.1.post4

Download PDF documentation: `GeoIPS_geoips.pdf`

Previous versions: Documentation of previous geoips versions are available at github.com/NRLMMD-GEOIPS.

Useful links: Source Repository | GeoIPS License | NRLMMD |

geoips is a free software program, United States Governmnet NRLMMD licensed.

```
Distribution Statement A. Approved for public release. Distribution␣
↪unlimited.

Author:
Naval Research Laboratory, Marine Meteorology Division

This program is free software: you can redistribute it and/or modify it␣
↪under
the terms of the NRLMMD License included with this program. This program␣
↪is
distributed WITHOUT ANY WARRANTY; without even the implied warranty of
MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the included␣
↪license
for more details. If you did not receive the license, for more␣
↪information see:
https://github.com/U-S-NRL-Marine-Meteorology-Division/
```

The Geolocated Information Processing System (GeoIPS).

CHAPTER

ONE

GEOIPS ® BASE PACKAGE

The GeoIPS Base Package provides a Python 3 based architecture supporting a wide variety of satellite and weather data processing. The modular nature of the GeoIPS base infrastructure also allows plug-and-play capability for user-specified custom functionality.

Homepage: https://github.com/NRLMMD-GEOIPS/geoips

User Guide

The user guide provides in-depth information on the key concepts of geoips with useful background information and explanation.

User Guide

The API reference guide

The reference guide contains a detailed description of geoips API. The reference describes how the methods work and which parameters can be used. It assumes that you have an understanding of the key concepts.

API

To the release notes

Change logs, versioning and contribution history.

Release Notes

Distribution Statement A. Approved for public release. Distribution unlimited.
# # #
Author:
Naval Research Laboratory, Marine Meteorology Division
# # #
This program is free software: you can redistribute it and/or modify it under
the terms of the NRLMMD License included with this program. This program is
distributed WITHOUT ANY WARRANTY; without even the implied warranty of
MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the included license
for more details. If you did not receive the license, for more information see:
https://github.com/U-S-NRL-Marine-Meteorology-Division/

CHAPTER

TWO

INTRODUCTION

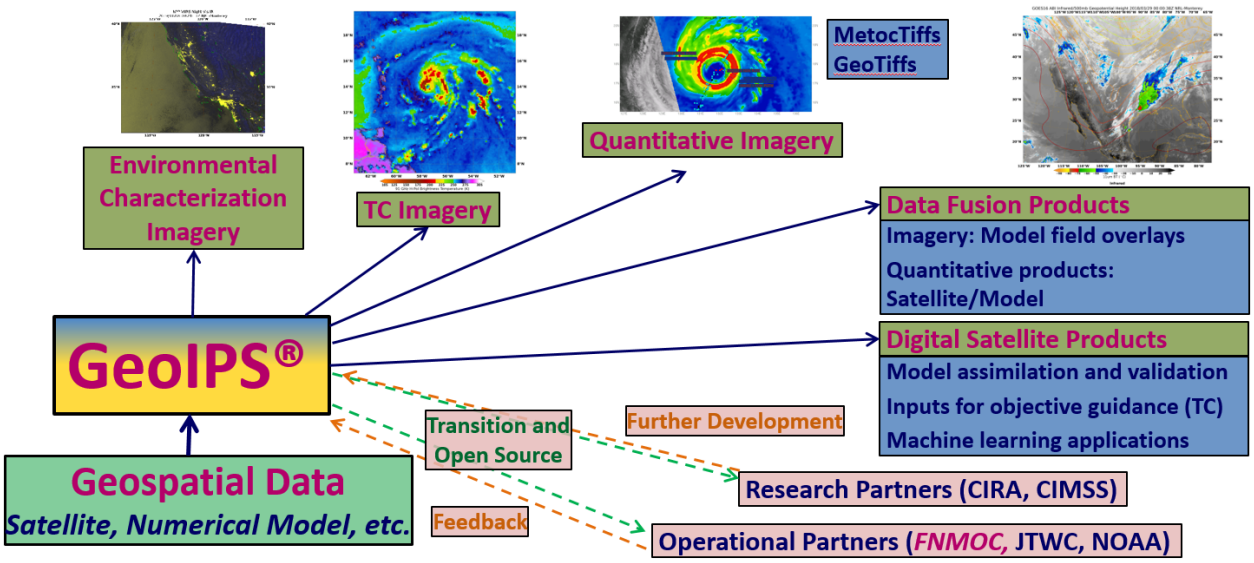
Software Requirements Specification

Distribution Statement A. Approved for public release. Distribution unlimited.
# # #
Author:
Naval Research Laboratory, Marine Meteorology Division
# # #
This program is free software: you can redistribute it and/or modify it under
the terms of the NRLMMD License included with this program. This program is
distributed WITHOUT ANY WARRANTY; without even the implied warranty of
MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the included license
for more details. If you did not receive the license, for more information see:
https://github.com/U-S-NRL-Marine-Meteorology-Division/

2.1 Description of GeoIPS

2.1.1 GeoIPS Overview

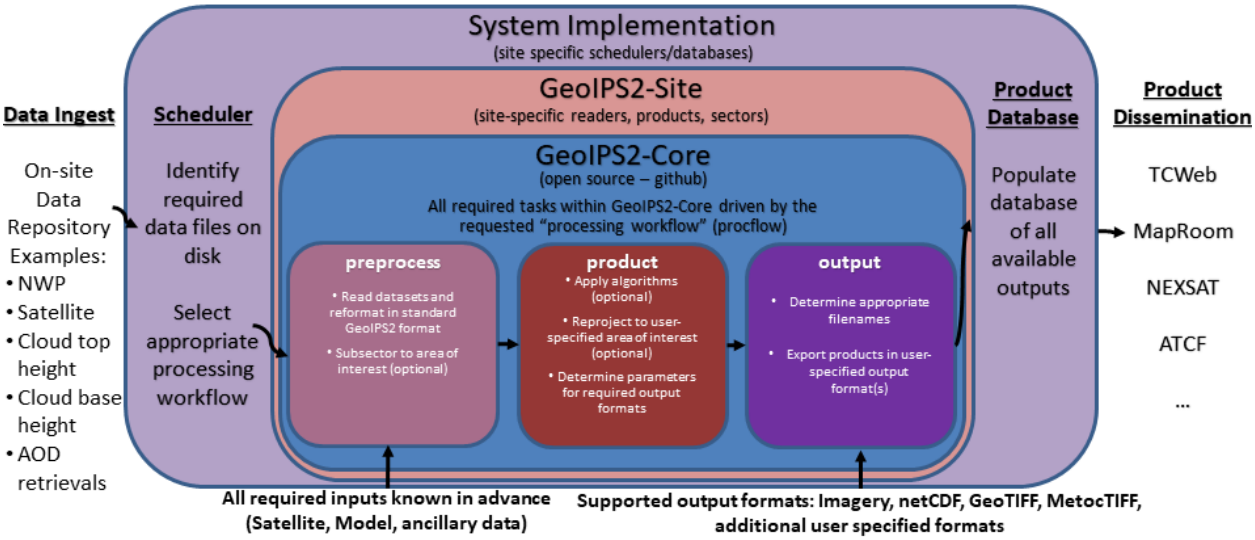
The Geolocated Information Processing System (GeoIPS) is a generalized processing system, providing a collection of algorithm and product implementations facilitating consistent and reliable application of specific products across a variety of sensors and data types.



GeoIPS acts as a toolbox for internal GeoIPS-based product development - all modules are expected to have simple inputs and outputs (Python numpy or dask arrays or xarrays, dictionaries, strings, lists), to enable portability and simplified interfacing between modules.

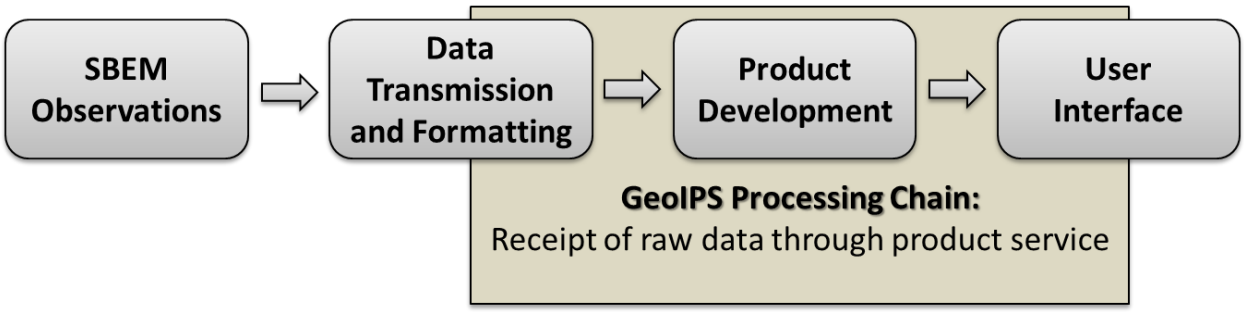
Some of the primary benefits / requirements of GeoIPS include:

- Seamless application to proprietary data types and products (no reference to external functionality within the main code base)
- Consistent product application across multiple sensors (both open source and proprietary)
- Flexible workflow to allow efficient real-time processing as well as interactive processing
- Modular interfaces to facilitate product development
- Consistent code base for research and development through operational transitions
- Ability to generate log outputs
- Ability to interface with workflow management tools (cylc)
- Ability to interface with databases (postgres)



2.1.2 GeoIPS Scope

The GeoIPS® "core" package is responsible for data processing from reading and reformatting the data into the common internal GeoIPS® internal format, through algorithm and product application, to outputting user configurable data formats (imagery, NetCDF, etc).



Data collection, data transfers, and product dissemination are all site specific implementations for driving GeoIPS® processing, and fall outside the scope of the GeoIPS® "core" processing system.

2.1.3 Using GeoIPS for Research

2.1.4 Using GeoIPS for Operations

Static sectors

Dynamic sectors

Distribution Statement A. Approved for public release. Distribution unlimited.

# # #
Author:
Naval Research Laboratory, Marine Meteorology Division
# # #
This program is free software: you can redistribute it and/or modify it under
the terms of the NRLMMD License included with this program. This program is
distributed WITHOUT ANY WARRANTY; without even the implied warranty of
MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the included license
for more details. If you did not receive the license, for more information see:
https://github.com/U-S-NRL-Marine-Meteorology-Division/

2.2 Summary of current Functionality

Distribution Statement A. Approved for public release. Distribution unlimited.
# # #
Author:
Naval Research Laboratory, Marine Meteorology Division
# # #
This program is free software: you can redistribute it and/or modify it under
the terms of the NRLMMD License included with this program. This program is
distributed WITHOUT ANY WARRANTY; without even the implied warranty of
MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the included license
for more details. If you did not receive the license, for more information see:
https://github.com/U-S-NRL-Marine-Meteorology-Division/

2.3 Example Output Images

Distribution Statement A. Approved for public release. Distribution unlimited.
# # #
Author:
Naval Research Laboratory, Marine Meteorology Division
# # #
This program is free software: you can redistribute it and/or modify it under
the terms of the NRLMMD License included with this program. This program is

distributed WITHOUT ANY WARRANTY; without even the implied warranty of
MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the included license
for more details. If you did not receive the license, for more information see:
https://github.com/U-S-NRL-Marine-Meteorology-Division/

2.4 Code of Conduct

Distribution Statement A. Approved for public release. Distribution unlimited.
# # #
Author:
Naval Research Laboratory, Marine Meteorology Division
# # #
This program is free software: you can redistribute it and/or modify it under
the terms of the NRLMMD License included with this program. This program is
distributed WITHOUT ANY WARRANTY; without even the implied warranty of
MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the included license
for more details. If you did not receive the license, for more information see:
https://github.com/U-S-NRL-Marine-Meteorology-Division/

CHAPTER

THREE

GETTING STARTED

Distribution Statement A. Approved for public release. Distribution unlimited.
# # #
Author:
Naval Research Laboratory, Marine Meteorology Division
# # #
This program is free software: you can redistribute it and/or modify it under
the terms of the NRLMMD License included with this program. This program is
distributed WITHOUT ANY WARRANTY; without even the implied warranty of
MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the included license
for more details. If you did not receive the license, for more information see:
https://github.com/U-S-NRL-Marine-Meteorology-Division/

3.1 Conda-based Installation

Using a fresh Mini/Anaconda Python 3.9+ Environment is the easiest way to get geoips up and running.

3.1.1 Complete Local conda-based GeoIPS Installation

The following instructions will guide you through installing GeoIPS using Anaconda Python. This installation method allows users to install GeoIPS without requiring administrative privileges by using Conda to install all of the "Required" system dependencies, then installing geoips into that conda environment.

1. Set GeoIPS Environment Variables

In order to support GeoIPS' testing infrastructure, there are a few required environment variables. You can change your installation location by changing the value of `$GEOIPS_PACKAGES_DIR` below.

```
# GeoIPS Default Locations
export GEOIPS_REPO_URL=https://github.com/NRLMMD-GeoIPS  # Point to base␣
↪URL for git clone commands
export GEOIPS_PACKAGES_DIR=$HOME/geoips
export GEOIPS_TESTDATA_DIR=$GEOIPS_PACKAGES_DIR/test_data
export GEOIPS_OUTDIRS=$GEOIPS_PACKAGES_DIR/outdirs
```

If desired, the GeoIPS environment variables can be added to your `$HOME/.bashrc` by running the following commands:

```
echo "export GEOIPS_REPO_URL=$GEOIPS_REPO_URL" >> ~/.bashrc
echo "export GEOIPS_PACKAGES_DIR=$GEOIPS_PACKAGES_DIR" >> ~/.bashrc
echo "export GEOIPS_TESTDATA_DIR=$GEOIPS_TESTDATA_DIR" >> ~/.bashrc
echo "export GEOIPS_OUTDIRS=$GEOIPS_OUTDIRS" >> ~/.bashrc
```

2. Clone the GeoIPS git repository, for installation and testing commands

```
mkdir -p $GEOIPS_PACKAGES_DIR
git clone ${GEOIPS_REPO_URL}/geoips.git $GEOIPS_PACKAGES_DIR/geoips
```

3. Install Anaconda or Miniconda

- Download the appropriate version of Conda, Miniconda, or Miniforge/Mambaforge.

For example, for Linux with Intel chips, one of the following:

```
# wget https://repo.anaconda.com/archive/Anaconda3-2023.03-1-Linux-x86_64.
↪sh
# wget https://repo.anaconda.com/miniconda/Miniconda3-latest-Linux-x86_64.
↪sh
# wget https://github.com/conda-forge/miniforge/releases/latest/download/
↪Miniforge3-Linux-x86_64.sh
wget https://github.com/conda-forge/miniforge/releases/latest/download/
↪Mambaforge-Linux-x86_64.sh
```

- Make the install script executable and run the installer, following the prompts (particularly the bit about conda init / restarting terminal!):

```
chmod u+x Mambaforge-Linux-x86_64.sh
./Mambaforge-Linux-x86_64.sh
# Follow instructions regarding conda init / restarting your terminal !
```

4. Create and activate a conda environment with some dependencies

Next we'll create a conda environment named `geoips` that contains all system requirements for GeoIPS. Many of these may already be installed on your system, but this command will ensure that for everyone.

```
# Note geos no longer required for cartopy >= 0.22
# openblas / gcc required for recenter_tc / akima build.
# imagemagick required for image comparisons
# git required for -C commands
conda create -y -n geoips -c conda-forge python=3.10 gcc gxx openblas␣
↪imagemagick git
conda activate geoips  # RUN EVERY TIME YOU WANT TO USE GEOIPS!
```

Note: You will need to run `conda activate geoips` every time you want to run or work on GeoIPS.

5. Install the GeoIPS git repository

This command installs all GeoIPS Python dependencies, and GeoIPS itself.

```
# Ensure geoips python environment enabled before installing geoips
pip install -e "$GEOIPS_PACKAGES_DIR/geoips"[doc,lint,test,debug]
```

6. Test your installation

To test your installation you will call two scripts:

- `base_install.sh` will clone repositories containing test data.
- `base_test.sh` will run a few integration tests to ensure that your installation is working correctly.

```
# Ensure geoips python environment enabled
conda activate geoips
# Download the test data
$GEOIPS_PACKAGES_DIR/geoips/tests/integration_tests/base_install.sh
```

(continues on next page)

(continued from previous page)

```
# Run integration tests
$GEOIPS_PACKAGES_DIR/geoips/tests/integration_tests/base_test.sh
```

7. Test output

For reference, the end of the output from the base_test.sh command should look something like below, indicating that none of the tests failed:

```
Package: geoips_base
Total run time: 82 seconds
Number data types run: 3
Number data types failed: 0
```

Distribution Statement A. Approved for public release. Distribution unlimited.
# # #
Author:
Naval Research Laboratory, Marine Meteorology Division
# # #
This program is free software: you can redistribute it and/or modify it under
the terms of the NRLMMD License included with this program. This program is
distributed WITHOUT ANY WARRANTY; without even the implied warranty of
MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the included license
for more details. If you did not receive the license, for more information see:
https://github.com/U-S-NRL-Marine-Meteorology-Division/

3.2 Expert User Installation (Administrative privileges)

The *Complete Local conda-based GeoIPS Installation <./installation.rst>* is the easiest way to get GeoIPS up and running, but if you have administrative privileges on your system, and are confident you can install all the required system dependencies yourself, you can install GeoIPS by setting a few environment variables for testing purposes.

3.2.1 Expert User GeoIPS Installation

System Dependencies

Use the *Complete Local conda-based GeoIPS Installation <./installation.rst>* **for the fully supported installation, which includes all dependencies**

Required (**included in** *Complete Local conda-based GeoIPS Installation <./installation.rst>*)

- wget (Miniconda installation)
- git >= 2.19.1 (git -C commands in complete installation)
- imagemagick (required for test output comparisons)
- openblas (required for scipy pip install)
- Python >= 3.9 (3.9 required for entry points)
- Test data repos contained in $GEOIPS_TESTDATA_DIR (required for tests to pass)

Optional

- gfortran (only required for plugins including fortran builds, build-essential)
- gcc & g++ (required for plugins including fortran or C builds, build-essential)
- screen (convenience package)
- ncurses (only required if building vim, ncurses and libncurses5-dev)

Minimal install

Use the *Complete Local conda-based GeoIPS Installation <./installation.rst>*. **for the fully supported installation, which includes all dependencies**

If you are confident you have all system requirements installed (Python+cartopy), and do not wish to go through the full installation process (which includes conda, rclone, test data, etc), you can clone the geoips repo and pip install from your local copy.

For the fully supported installation, please use the *Complete Local conda-based GeoIPS Installation <./installation.rst>*.

```
# NOTE: ALL dependencies above MUST be installed/available
#       to use this installation method.
# Please follow complete conda-based installation in the
#       next section for fully supported complete install.
export GEOIPS_REPO_URL=https://github.com/NRLMMD-GEOIPS
export GEOIPS_PACKAGES_DIR=<installation_location>
export GEOIPS_TESTDATA_DIR=<desired_test_data_location>
```

(continues on next page)

(continued from previous page)

```
export GEOIPS_OUTDIRS=<desired_output_file_location>

git clone $GEOIPS_REPO_URL/geoips.git $GEOIPS_PACKAGES_DIR/geoips
pip install -e $GEOIPS_PACKAGES_DIR/geoips
$GEOIPS_PACKAGES_DIR/geoips/tests/integration_tests/base_install.sh
$GEOIPS_PACKAGES_DIR/geoips/tests/integration_tests/base_test.sh
```

Distribution Statement A. Approved for public release. Distribution unlimited.
# # #
Author:
Naval Research Laboratory, Marine Meteorology Division
# # #
This program is free software: you can redistribute it and/or modify it under
the terms of the NRLMMD License included with this program. This program is
distributed WITHOUT ANY WARRANTY; without even the implied warranty of
MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the included license
for more details. If you did not receive the license, for more information see:
https://github.com/U-S-NRL-Marine-Meteorology-Division/

3.3 Simple command line examples

3.3.1 Full Install

In order to obtain all test repos and plugin repositories currently available publicly, you can run the test script $GEOIPS_PACKAGES_DIR/geoips/tests/test_full_install.sh. This will download, install, and test all possible data types and products.

3.3.2 Step by Step

In the future, we will provide simple step by step examples using Jupyter Notebooks in order to quickly get up to speed on GeoIPS capabilities and functionality. Maybe we could link to some Jupyter notebooks here?

Distribution Statement A. Approved for public release. Distribution unlimited.
# # #
Author:
Naval Research Laboratory, Marine Meteorology Division

```
# # #
# # # This program is free software: you can redistribute it and/or modify it under
# # # the terms of the NRLMMD License included with this program. This program is
# # # distributed WITHOUT ANY WARRANTY; without even the implied warranty
of
# # # MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See
the included license
# # # for more details. If you did not receive the license, for more information see:
# # # https://github.com/U-S-NRL-Marine-Meteorology-Division/
```

3.4 Extending GeoIPS with your own functionality

3.4.1 Discussion

Please see "Extending GeoIPS with Plugins" in the User's Guide for more information on using GeoIPS for your own use cases.

```
# # # Distribution Statement A. Approved for public release. Distribution unlimited.
# # #
# # # Author:
# # # Naval Research Laboratory, Marine Meteorology Division
# # #
# # # This program is free software: you can redistribute it and/or modify it under
# # # the terms of the NRLMMD License included with this program. This program is
# # # distributed WITHOUT ANY WARRANTY; without even the implied warranty
of
# # # MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See
the included license
# # # for more details. If you did not receive the license, for more information see:
# # # https://github.com/U-S-NRL-Marine-Meteorology-Division/
```

CHAPTER

FOUR

USER GUIDE

Distribution Statement A. Approved for public release. Distribution unlimited.
# # #
Author:
Naval Research Laboratory, Marine Meteorology Division
# # #
This program is free software: you can redistribute it and/or modify it under
the terms of the NRLMMD License included with this program. This program is
distributed WITHOUT ANY WARRANTY; without even the implied warranty of
MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the included license
for more details. If you did not receive the license, for more information see:
https://github.com/U-S-NRL-Marine-Meteorology-Division/

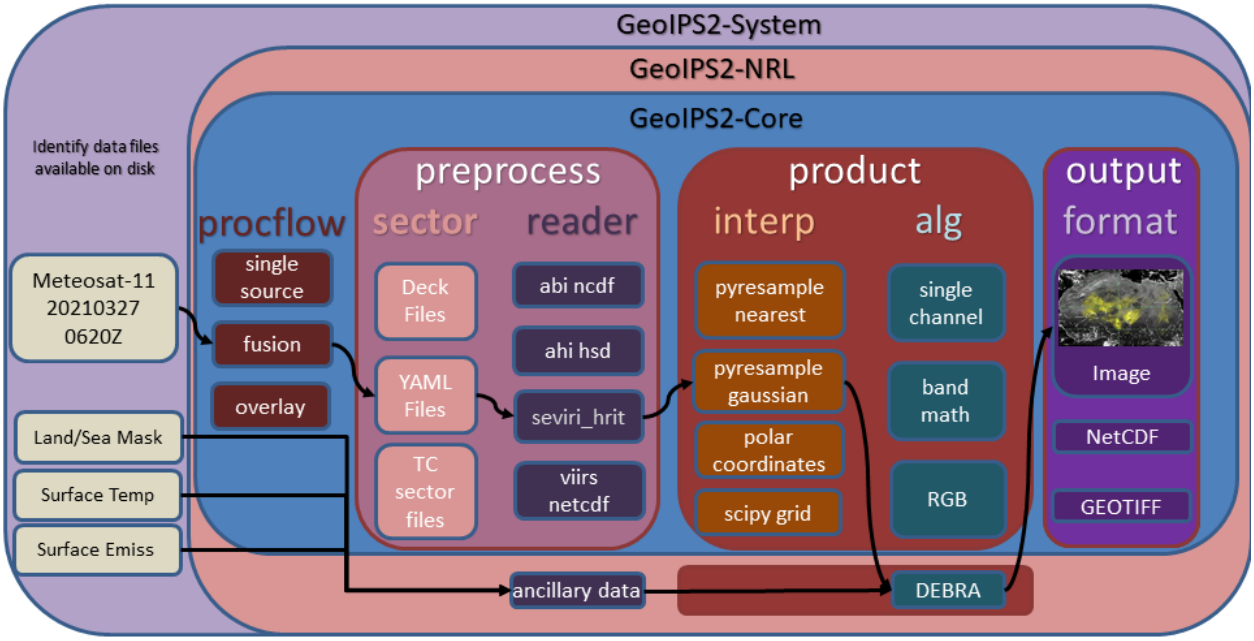
4.1 Description of GeoIPS structure

4.1.1 Overview of interfaces and plugins

A primary goal of GeoIPS is to provide seamless integration of external functionality, with no reference to proprietary algorithms and readers from the base open source geoips code base.

GeoIPS Interfaces are used to abstract the process of accessing different pieces of GeoIPS functionality (plugins) in order to support installing GeoIPS Plugins from external repositories with no reference to those pieces of functionality from within the main code base.

Example external plugin functionality:



GeoIPS makes use of Python entry points to install external packages within the geoips namespace, then an internal Application Programming Interface to access specific modules.

GeoIPS is made up of a collection of plugins of different types, accessed via specific plugin interfaces.

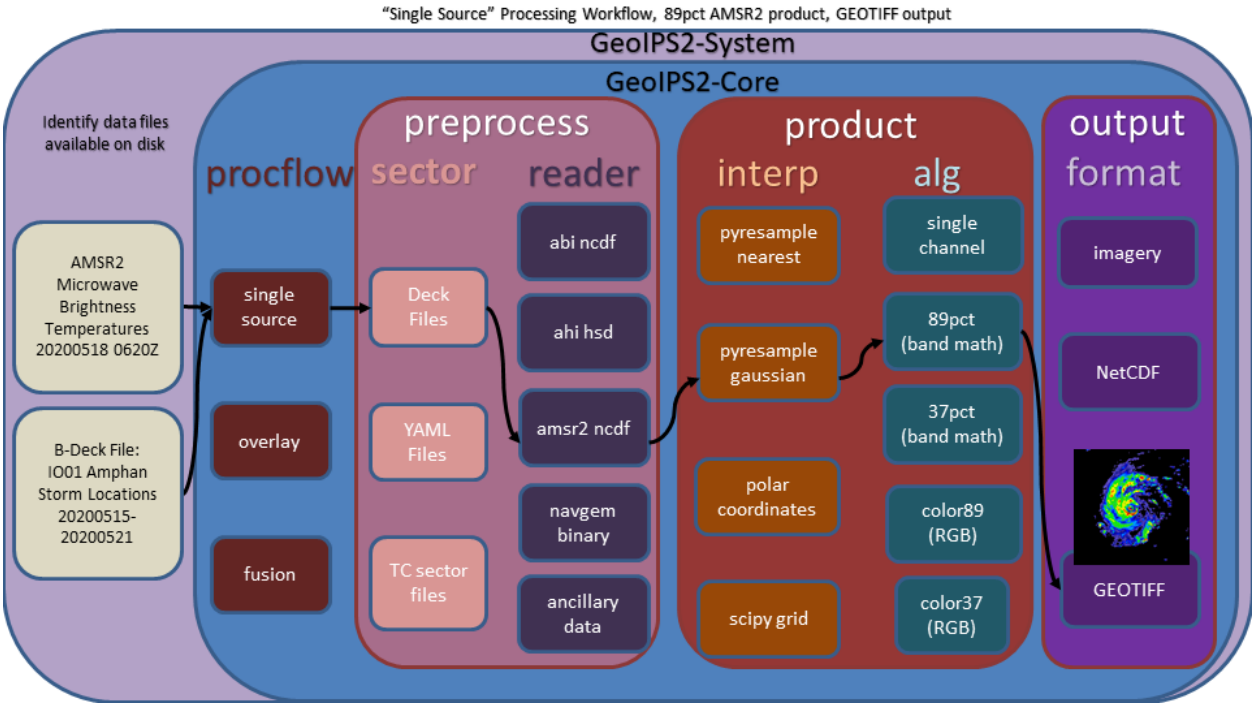
The primary plugin interfaces include:

- **processing workflows (procflows)** - drive a specific collection of steps for a particular type of processing
- **static sectors** - specifications of domains of interest
- **dynamic sectors** - specifications of dynamic domains of interest
- **readers** - specifications for ingesting a specific data type, and storing in the GeoIPS xarray-based internal format
- **products** - overall product specification, including interpolation routine, algorithm, colormaps, etc (see YAML-based interfaces)
  - **interpolators** - interpolation routine to apply when reprojecting data
  - **algorithms** - data manipulations to apply to dataset
  - **colormaps** - colormap to apply to resulting product
- **output_formatters** - data format for the resulting output product (ie, netCDF, png, etc)
- **filename_formatters** - full path and file name formatting specification, using attributes within the xarray objects

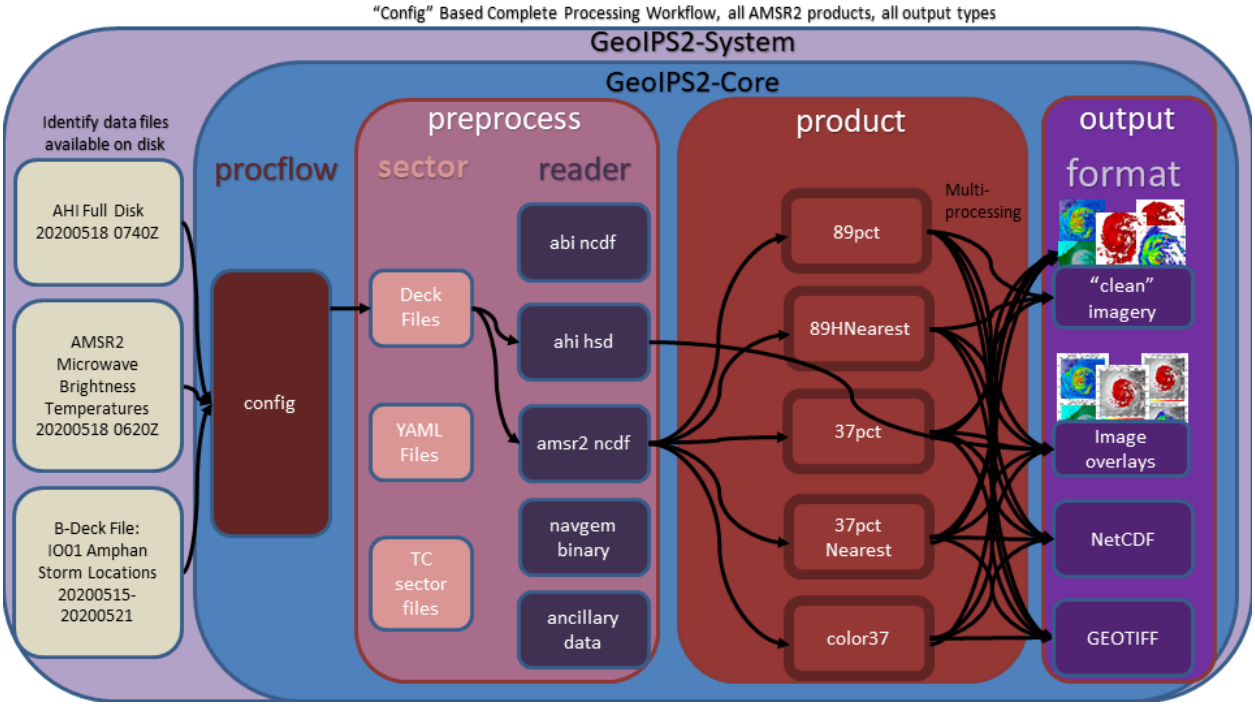
The primary processing workflows available at this time, which access the appropriate plugins at the appropriate point in the processing stream using the appropriate plugin interface, include:

- **single_source** - single input type and single output type
- **config_based** - efficient method for producing all possible outputs for a given set of data files.

Example single_source processing workflow:



Example of config based processing workflow


"Config" Based Complete Processing Workflow, all AMSR2 products, all output types
GeoIPS2-System
GeoIPS2-Core
Identify data files available on disk
AHI Full Disk 20200518 0740Z
AMSR2 Microwave Brightness Temperatures 20200518 0620Z
B-Deck File: IO01 Amphan Storm Locations 20200515-20200521
procflow
config
preprocess
sector
reader
Deck Files
YAML Files
TC sector files
abi ncdf
ahi hsd
amsr2 ncdf
navgem binary
ancillary data
product
Multi-processing
89pct
89HNearest
37pct
37pct Nearest
color37
output
format
"clean" imagery
Image overlays
NetCDF
GEOTIFF

4.1.2 Module-based interfaces

Algorithms

Colormaps

Filename formatters

Interpolators

Output formatters

ProcFlows

Readers

Title formatters

4.1.3 YAML-based Interfaces

Feature Annotator

Gridline Annotator

Product defaults

Products

Dynamic sectors

Static sectors

ProcFlow configurations

```
# # # Distribution Statement A. Approved for public release. Distribution unlimited.
# # #
# # # Author:
# # # Naval Research Laboratory, Marine Meteorology Division
# # #
# # # This program is free software: you can redistribute it and/or modify it under
# # # the terms of the NRLMMD License included with this program. This program is
```

distributed WITHOUT ANY WARRANTY; without even the implied warranty of
MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the included license
for more details. If you did not receive the license, for more information see:
https://github.com/U-S-NRL-Marine-Meteorology-Division/

4.2 Command Line Interface

The GeoIPS Command Line Interface is currently under development - once the functionality is finalized, instructions will be included here for interrogating GeoIPS capabilities via the CLI.

Distribution Statement A. Approved for public release. Distribution unlimited.
# # #
Author:
Naval Research Laboratory, Marine Meteorology Division
# # #
This program is free software: you can redistribute it and/or modify it under
the terms of the NRLMMD License included with this program. This program is
distributed WITHOUT ANY WARRANTY; without even the implied warranty of
MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the included license
for more details. If you did not receive the license, for more information see:
https://github.com/U-S-NRL-Marine-Meteorology-Division/

4.3 GeoIPS Functionality

The GeoIPS Command Line Interface is currently under development - once the CLI is complete, it will provide a simple method of listing all possible GeoIPS Plugins with a single command. The resulting functionality list will be outlined in this section.

Distribution Statement A. Approved for public release. Distribution unlimited.
# # #
Author:
Naval Research Laboratory, Marine Meteorology Division
# # #
This program is free software: you can redistribute it and/or modify it under
the terms of the NRLMMD License included with this program. This program is

distributed WITHOUT ANY WARRANTY; without even the implied warranty of

MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the included license

for more details. If you did not receive the license, for more information see:

https://github.com/U-S-NRL-Marine-Meteorology-Division/

4.4 Extend GeoIPS with Plugins

4.4.1 Developing Module-based plugin

4.4.2 Developing YAML-based plugin

4.4.3 Example Module-based Plugins

Algorithnms

Colormaps

Filename formatters

Interpolators

Output Formatters

ProcFlows

Readers

Title Formatters

4.4.4 Example YAML-based Plugins

Boundary Annotators

Gridline Annotators

Product Defaults

Products

Dynamic Sectors

Static Sectors

ProcFlow Configurations

Distribution Statement A. Approved for public release. Distribution unlimited.
# # #

```
# # # Author:
# # # Naval Research Laboratory, Marine Meteorology Division
# # #
# # # This program is free software: you can redistribute it and/or modify it under
# # # the terms of the NRLMMD License included with this program. This program is
# # # distributed WITHOUT ANY WARRANTY; without even the implied warranty
of
# # # MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See
the included license
# # # for more details. If you did not receive the license, for more information see:
# # # https://github.com/U-S-NRL-Marine-Meteorology-Division/
```

CHAPTER

FIVE

DEVELOPER GUIDE

Distribution Statement A. Approved for public release. Distribution unlimited.
# # #
Author:
Naval Research Laboratory, Marine Meteorology Division
# # #
This program is free software: you can redistribute it and/or modify it under
the terms of the NRLMMD License included with this program. This program is
distributed WITHOUT ANY WARRANTY; without even the implied warranty of
MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the included license
for more details. If you did not receive the license, for more information see:
https://github.com/U-S-NRL-Marine-Meteorology-Division/

5.1 Software Requirements Specification

For **Geolocated Information Processing System**

Version 1.1 approved
Prepared by Mindy Surratt and Chris Camacho
Naval Research Laboratory Marine Meteorology Division
7 February 2023

5.1.1 Table of Contents

- *Revision History*
- ***1. Introduction***
  - *1.1 Purpose*
  - *1.2 Intended Audience and Reading Suggestions*
  - *1.3 Product Scope*
  - *1.4 References*
- ***Overall Description***
  - *2.1 Product Perspective*
  - *2.2 Product Functions*
  - *2.3 User Classes and Characteristics*
  - *2.4 Operating Environment*
  - *2.5 Design and Implementation Constraints*
  - *2.6 User Documentation*
  - *2.7 Assumptions and Dependencies*
- ***External Interface Requirements***
  - *3.1 User Interfaces*
  - *3.2 Hardware Interfaces*
  - *3.3 Software Interfaces*
- ***System Features***
  - *4.1 Data Fusion Capability*
  - *4.2 Products Over Various Spatial Domains*
  - *4.3 Products of Varied Output Formats*

5.1.2 Revision History

Name	Date	Reason For Changes	Version
Surratt/Camacho	2021-10-20	Initial version	1.0
Surratt/Camacho	2023-02-06	Google -> NumPy docstrings black/flake8/bandit code checks	1.1

1. Introduction

1.1 Purpose

The Geolocated Information Processing System (GeoIPS) is a generalized processing system, providing a collection of **algorithm and product implementations** facilitating **consistent and reliable application** of specific products across a **variety of sensors and data types**.

GeoIPS acts as a toolbox for internal GeoIPS-based product development - all modules are expected to have simple inputs and outputs (Python numpy or dask arrays or xarrays, dictionaries, strings, lists), to enable portability and simplified interfacing between modules.

Some of the primary benefits / requirements of GeoIPS include:

- Seamless application to proprietary data types and products (no reference to external functionality within the main code base)
- Consistent product application across multiple sensors (both open source and proprietary)
- Flexible workflow to allow efficient real-time processing as well as interactive processing
- Modular interfaces to facilitate product development
- Consistent code base for research and development through operational transitions
- Ability to generate log outputs
- Ability to interface with workflow management tools (cylc)
- Ability to interface with databases (postgres)

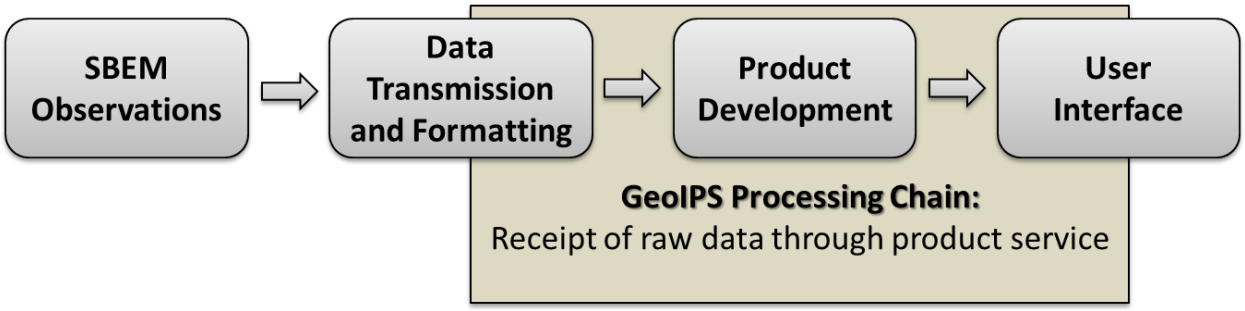
1.2 Intended Audience and Reading Suggestions

This document is primarily intended for system administrators, users, testers and project managers.

Software developers should consider this required reading prior to working through the documentation.

1.3 Product Scope

The GeoIPS® "core" package is responsible for data processing from reading and reformatting the data into the common internal GeoIPS® internal format, through algorithm and product application, to outputting user configurable data formats (imagery, NetCDF, etc).



Data collection, data transfers, and product dissemination are all site specific implementations for driving GeoIPS® processing, and fall outside the scope of the GeoIPS® "core" processing system.

1.4 References

Software Requirements Specification Template

This Software Requirements Specification Document was developed using the following template:

https://github.com/rick4470/IEEE-SRS-Tempate

Documentation and Style Strategy

GeoIPS uses Sphinx with the Napoleon extension for automated documentation generation.

https://www.sphinx-doc.org/en/master/usage/extensions/napoleon.html

GeoIPS Syntax and Style Checking

GeoIPS uses the NumPy docstring format within the code base for simplicity:

https://numpydoc.readthedocs.io/en/latest/format.html

bandit, flake8, and black are used to enforce appropriate style, security, and syntax usage. flake8-rst and flake8-rst-docstring plugins are used to enforce numpy docstring formatting.

Overall Description

2.1 Product Perspective

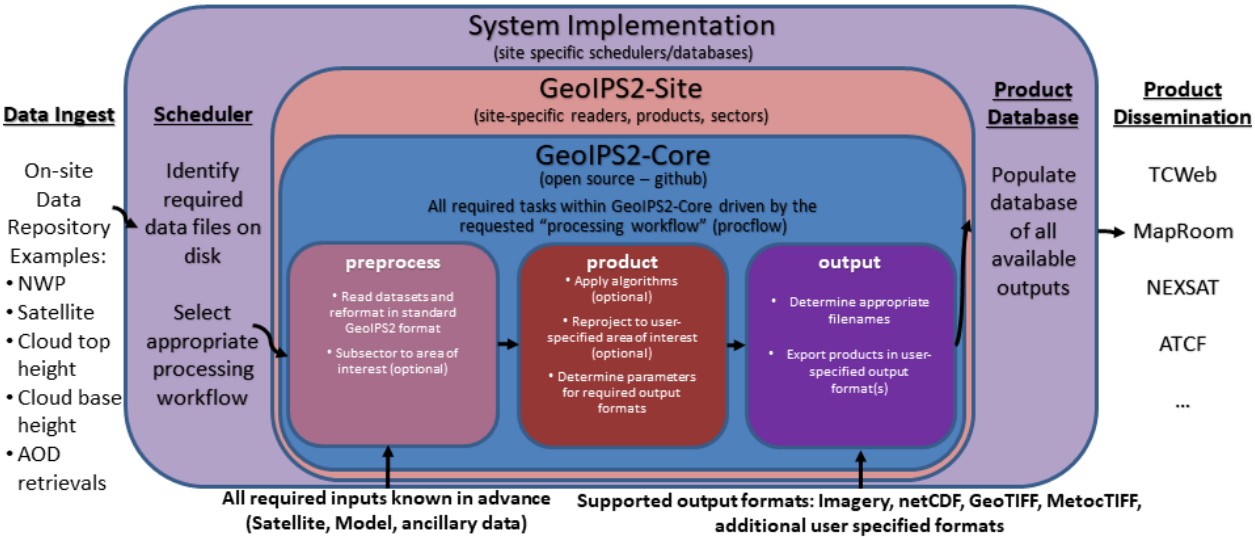
The Meteorology and Oceanography (METOC) community has an increasingly large number of disparate data sources available for advanced environmental exploitation – numerical model outputs, rapid refresh next generation geostationary weather satellites, polar orbiting microwave imagers and sounders, radar data, direct observations from ships and weather stations, climatology, elevation and emissivity databases, and many more data types, both static and dynamic. Additionally, with the upcoming launch of countless microsats, an efficient and easy-to-use processing system is imperative for rapid implementation of these new datasets.

The amount of information that can be gained by combining these datasets in unique ways is far greater than from any single data type. GeoIPS® will result in a collaborative, easy-to-use processing system that can support development efforts integrating these disparate data sources into unique products, and facilitate streamlined operational transitions. This common platform will be used across the METOC community – including basic research, real-time implementation, and operational processing.

With the plethora of weather satellites coming on line, it is imperative to develop a sustainable, open source, community supported, efficient, modular processing platform to enable future functionality and facilitate near real-time operational capability for all new sensors and products.

The GeoIPS® project will deliver a much needed capability for efficient environmental data processing, benefiting METOC users across the community. The collaborative nature of GeoIPS® development will lead to increased efficiency and functionality of the final product.

2.2 Product Functions



2.3 User Classes and Characteristics

There are 4 primary user classes for the GeoIPS® system: researchers, light developers, expert developers, and operators.

Researchers will use the system to generate output products for further analysis, but will not be implementing major changes to the products or algorithms themselves – only using the output of GeoIPS® to aid in their research.

Light developers will make minor changes to the code base (slight modifications to basic products, algorithms, sectors, etc), but will not make major changes to the internals of the GeoIPS® infrastructure. This allows product customization, without a deep knowledge of the backend processing architecture.

Expert developers will work with the internals of the main GeoIPS-Core code base – providing new functionality and features that are commonly used by multiple site-specific implementations. Expert developers should be intimately familiar with all aspects of the Software Requirements Specification.

Operators will drive GeoIPS® processing via YAML config files, specifying all required products and sectors. Operators will implement minimal changes to the GeoIPS® code base, and will require additional software outside of the GeoIPS® system to drive the real-time processing (processing workflows, database management systems, data dissemination protocols).

2.4 Operating Environment

GeoIPS® is developed and tested under the GNU/Linux operating system, on 64-bit x86 hardware architecture.

GeoIPS® must be fully functional under RHEL 8.4 with SELinux enabled (including Fortran and C compiled code)

2.5 Design and Implementation Constraints

GeoIPS® is a Python 3 based processing system, with support for Fortran and C routines.

Minimum Base Python package requirements include xarray, scipy, pyresample, and pyyaml, with additional requirements for specific readers and algorithms.

GeoIPS® requires gfortran and gcc compilers – must support gcc version 8, 9, or 10.

2.6 User Documentation

The GEOIPS-Core code base, documentation, and tutorial support are available on github.

Test datasets can be obtained from geoips@nrlmry.navy.mil.

2.7 Assumptions and Dependencies

Hardware and software requirements stated in this document are pertinent to the GeoIPS-Core version of the codebase. Additional site-specific packages and system implementations could incur additional resource requirements or software dependencies.

External Interface Requirements

3.1 User Interfaces

- Must support Linux-based processing, no GUI requirement.
- Must run via command line interactively, one product at a time.
- Must be able to run via config-based processing to efficiently drive multiple outputs.
- Command line and config-based interfaces must map user requested options to modular software interfaces.
- Must be able to support web-based requests. (Config-based processing supports web-based requests – potentially separate application to generate config file to drive processing)

- Must be able to operate via queueing and scheduling systems (covered via config-based processing)
- Must provide feedback throughout processing to monitor progress.
  - Log output (errors, warnings, status)
  - Messages during runtime identifying which part of the processing is currently active

3.2 Hardware Interfaces

GeoIPS® is developed and tested under the GNU/Linux operating system, on 64-bit x86 hardware architecture. GeoIPS® must successfully operate under Red Hat Enterprise Linux 8.4 with SELinux enabled.

Processing medium resolution next generation geostationary satellite data (ABI, AHI) and polar orbiter satellite data with GeoIPS® requires a minimum of 2 processors with 16GB memory. High resolution next generation geostationary satellite datasets requires at least 24GB memory.

3.3 Software Interfaces

Requirements:

- **Must allow internal multi-processing**
  - Individual modules are allowed to include multi-threading and multi-processing
    - Dask based processing
  - Managing queues to ensure multi-processing is handled properly is outside the scope of GeoIPS itself.
  - Config-based processing allows driving processing in different configurations to ensure optimal efficient processing.
- **Must be able to map each point to**
  - Latitude
  - Longitude
  - vertical position
  - observation time (ie, when the model was run)
  - valid time (ie, observation time + tau)
- **Must store satellite specific attributes**
  - Satellite zenith and azimuth angles
  - Orbital parameters

- **Must be able to ingest geo-located data with temporal and vertical information**
  - LIDAR data, sounder data, model data, and other datasets including vertical coordinates
- **Must store metadata on projections**
- **Must have common backend data format**
  - dictionary of xarray datasets, one for each shape/resolution/attribute set of data.
  - Each individual xarray dataset contains the following variables:
    - 'latitude' - REQUIRED 2d array the same shape as data variables
    - 'longitude' - REQUIRED 2d array the same shape as data variables
    - 'vertical_position' – OPTIONAL 2d array the same shape as data variables
      - Required for feature height, volumetric, models, curtain – ie, if heights change
    - 'time' - OPTIONAL 2d array the same shape as data variables
- **Each individual xarray dataset must contain the following metadata attributes**
  - 'source_name' – REQUIRED
  - 'platform_name' – REQUIRED
  - 'data_provider' – REQUIRED
  - 'start_datetime' – REQUIRED
  - 'end_datetime' – REQUIRED
  - 'interpolation_radius_of_influence – REQUIRED
  - 'vertical_data_type'
    - Surface
    - Column integrated
    - Feature height (ie, Cloud top, ocean, etc)
    - Volumetric (3d)
    - Curtain (2d)
  - Projection information (how it was, or how it will be mapped onto a grid)
- **Each reader return must contain a 'METADATA' dictionary key with only metadata attributes**
  - Must include required metadata fields
  - May include any additional optional desired metadata fields
- **Variables and attributes on xarray datasets will follow CF Standards, with units matching the CF canonical units**

  - http://cfconventions.org/Data/cf-standard-names/current/build/cf-standard-name-table.html

- **Model xarray objects are organized with separate datasets for each level type**
  - Mean sea level
  - Pressure
  - Surface
  - Top
  - Zheight
- **Time-series data (model, fire) is stored in 3 dimensional DataArrays, where the 3rd dimension relates to time (tau for model data), with a separate "time" data array**
- **Must have modular capability for different output formats**
  - Request output format modules during run-time (via config files or command line) – no reference to specific output formats within code base.
    * Xarray based inputs containing
      · data to plot
      · requested region of interest
      · plotting parameters (optional)
    * Performs actual plotting / output commands
      · Generates output directly within the module
    * Returns list of filenames that were generated
- **Must have modular capability for user-specified output filenames**
  - Request filename module during run-time
  - Xarray based inputs with required metadata to compile filename
  - Returns string of resulting filename
- **Must have modular capability for sector specifications**
  - Request sector specification modules during run-time
  - Xarray based inputs
  - Pyresample area definition based output
- **Must support config-file specified "product" parameters referencing one or more of**
  - Interpolation scheme
  - Colormap

  - Algorithm to apply, and required algorithm arguments
  - Separate mapping of sensor to required variables for each supported product (allowing implementation of existing products to proprietary data types)

System Features

4.1 Data Fusion Capability

4.1.1 Description and Priority

High Priority - Must be able to produce output products from different combinations of input datasets.

4.1.2 Stimulus/Response Sequences

Required input data sets and desired output products must be specified via a single command line call.

Required data files and product parameters can be specified either via:

- explicit command line options or
- YAML config specifications

4.1.3 Functional Requirements

- **Must be able to produce output products from combinations of**
  - Geostationary satellites
  - Vis/IR Polar orbiters
  - Passive microwave polar orbiters
  - Numerical Weather Prediction model outputs
  - Radar data
  - Sounder data
  - Lidar data
  - Other ancillary datasets (elevation, surface emissivity, etc)

4.2 Products Over Various Spatial Domains

4.2.1 Description and Priority

High Priority - Must be able to produce output products over a variety of user-specifiable spatial domains.

4.2.2 Stimulus/Response Sequences

Requested spatial domains (referred to as area definitions or sectors) must be requested via a single command line call. Sector information can be specified either via:

- explicit command line options or
- YAML config specifications

4.2.3 Functional Requirements

- **Must be able to produce products over various spatial domains for the above datasets**
  - Tropical cyclone centered imagery
  - Static regions of varying spatial resolutions and coverage, at any location on the globe
  - Algorithms applied to raw datasets (original resolution / coverage)
  - 3 and 4 dimensional outputs (model data, sounder data, lidar data, etc)

4.3 Products of Varied Output Formats

4.3.1 Description and Priority

High Priority - Must be able to produce output products of various user-specifiable output formats.

4.3.2 Stimulus/Response Sequences

Requested output formats must be requested via a single command line call.

Requested output format information can be specified either via:

- explicit command line options or
- YAML config specifications

4.3.3 Functional Requirements

- **Must be able to produce the following output types for any combination of the above datasets and domains**
  - Annotated imagery outputs (titles, coastlines, gridlines)
  - Non-annotated imagery outputs with associated metadata for displaying within external image viewers
  - METOCTIFF quantitative imagery output (for ATCF/JMV/MapRoom viewers)
  - GeoTIFF output
  - GeoJSON output
  - netCDF output with pre-processed data
  - text outputs

Distribution Statement A. Approved for public release. Distribution unlimited.
# # #
Author:
Naval Research Laboratory, Marine Meteorology Division
# # #
This program is free software: you can redistribute it and/or modify it under
the terms of the NRLMMD License included with this program. This program is
distributed WITHOUT ANY WARRANTY; without even the implied warranty of
MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the included license
for more details. If you did not receive the license, for more information see:
https://github.com/U-S-NRL-Marine-Meteorology-Division/

5.2 Contributors Guide

The GeoIPS Team encourages active participation by the user and developer community, so we welcome any and all feedback on and updates to the code base.

Each step in this process can be completed by one or more individuals (we encourage collaboration!), so feel free to submit Issues even if you have no intention of resolving it yourself. And feel free to submit a Pull Request even if all tests are not complete - someone else may have time to finalize for approval even if you are unable.

Contributions generally follow the process of:

1. Submit a GitHub Issue for a bug fix or feature request. *GeoIPS GitHub Issue Creation Workflow*
2. Branch off a given Issue in order to make the required changes *FROM WEB: MEMBERS: Create Branch from Existing Issue*
3. Create a github Pull Request in order for your changes to be reviewed prior to being merged to the integration branch *GeoIPS GitHub Pull Request workflow*
4. Ensure all required tests pass prior to PR approval (more info coming soon)
    - Unit tests
    - Integration tests
    - Code formatting/style tests
    - Documentation formatting/style tests
5. Merge your changes into the main code base! *GeoIPS Merge PR and Close Issue workflow*

5.3 Setting up for development

5.3.1 Instructions for setting up a new data type

Instructions for setting up a new data type, including:

- creating a new geoips test data repository from template,
- creating a new geoips plugin repository from template,

This example will use a data type called "mydatatype".

Anything denoted with an '@' symbol within these instructions and within the associated templates will need to be modified accordingly, based on your desired functionality

Follow Issue, Branch Pull Request Process if within GEOIPS org

This only required if you are creating a new repository within the NRLMMD GEOIPS organization. If you are creating functionality within your own organization, you can follow your own version control process.

1. **Create branch for updates**
    - Follow Issue, Branch, Pull Request process in $GEOIPS/docs/source/devguide/git_workflow.rst
    - *Git Workflow <./git_workflow.rst>*

Create test data repo

If a new data type is required for your plugin, create a separate test data repo to hold the test datasets. Source code and test outputs are stored separately from test datasets (idea being test datasets are effectively static, and will very infrequently require updates, unless new datasets are added).

Use the "template_test_data" template on github

- https://github.com/NRLMMD-GEOIPS/template_test_data
- Click green "Use this template" button top right
- Owner: GEOIPS
    - NOTE if you do not have permissions to create a repo, contact geoips@nrlmry.navy.mil, or create it under your own organization / user.
- Repository name: geoips_@mydatatype@
- Description: @Include useful description@
- Private: Select "Private"
- Include all branches: SELECT
- Click green "Create repository from template" button at bottom
- Repository name: geoips_@mydatatype@
- Description: @Include useful description@
- Private: Select "Private"
- Include all branches: SELECT
- Click green "Create repository from template" button at bottom

```
cd $GEOIPS_TESTDATA_DIR      # By standard convention, place within
                             # $GEOIPS_TESTDATA_DIR/test_data_@mydatatype@
↪on the filesystem.
git clone https://github.com/NRLMMD-GEOIPS/test_data_@mydatatype@
```

Follow instructions in template README

Create GeoIPS plugin repository for readers / products

Use the "template_basic_plugin" template on github

- https://github.com/NRLMMD-GEOIPS/template_basic_plugin
- Click green "Use this template" button top right
- Owner: GEOIPS
  - NOTE if you do not have permissions to create a repo, contact geoips@nrlmry.navy.mil, or create it under your own organization / user.
- Repository name: geoips_@mydatatype@
- Description: @Include useful description@
- Private: Select "Private"
- Include all branches: SELECT
- Click green "Create repository from template" button at bottom

```
  cd $GEOIPS_PACKAGES_DIR
  git clone https://github.com/NRLMMD-GEOIPS/geoips_@mydatatype@
##########################################################################
↪##
```

Follow instructions in template README

Distribution Statement A. Approved for public release. Distribution unlimited.
# # #
Author:
Naval Research Laboratory, Marine Meteorology Division
# # #
This program is free software: you can redistribute it and/or modify it under
the terms of the NRLMMD License included with this program. This program is
distributed WITHOUT ANY WARRANTY; without even the implied warranty of

MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the included license
for more details. If you did not receive the license, for more information see:
https://github.com/U-S-NRL-Marine-Meteorology-Division/

5.4 Building documentation

Distribution Statement A. Approved for public release. Distribution unlimited.
# # #
Author:
Naval Research Laboratory, Marine Meteorology Division
# # #
This program is free software: you can redistribute it and/or modify it under
the terms of the NRLMMD License included with this program. This program is
distributed WITHOUT ANY WARRANTY; without even the implied warranty of
MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the included license
for more details. If you did not receive the license, for more information see:
https://github.com/U-S-NRL-Marine-Meteorology-Division/

5.5 GeoIPS git workflow

Follow these steps when making modifications to geoips-based packages or plugins

5.5.1 GeoIPS GitHub Issue Creation Workflow

FROM WEB: Create an Issue for something that needs to be done

- Select an appropriate Issue template
  - Navigate to: https://github.com/NRLMMD-GEOIPS/geoips/issues
    - NOTE: you can create Issues on repos besides "geoips" if desired.
  - Click green "New Issue" button in top right
  - Click green "Get started" box to the right of the desired template
- Populate Issue contents appropriately

  - **Title**: Short descriptive name for the Issue (example: "Update GEOIPS_REPO_URL to GitHub")
  - **Write**: Follow template within "Write" tab to populate with appropriate content
  - **Assignees**: Add Assignees as appropriate
  - **Labels**: Add descriptive labels as appropriate
  - **Projects**: Link to "GeoIPS - All Repos and all Functionality", other Projects as appropriate
- Click "Submit new issue"

FROM WEB: MEMBERS: Create Branch from Existing Issue

NOTE: Those who are NOT members of the GeoIPS organization will fork, not branch. Skip to NON MEMBERS section

- Navigate to Issue you would like to resolve
- Click on Development->Create Branch
  - **Branch name** Use auto-populated default branch name
  - **Repository Destination** Select repository to which you would like to make changes
    * NOTE you can create branches on repositories outside the repository the Issue resides in
  - **Change branch source** optional (defaults to "main")
  - Select **"Checkout locally"**
  - Click **"Create branch"**
  - Copy and paste the resulting "git fetch" and "git checkout" commands

FROM WEB: NON-MEMBERS: Create fork of repo

- **NOTE: NRLMMD-GEOIPS members will branch following steps above, **skip this section if you are a member**
- Navigate to desired repository
- Click drop down next to "Fork"
- Click "+ Create a new fork"
- Select appropriate owner/organization to own the fork (could be your individual github username)
- Uncheck "copy main branch only"

- Click "create fork"

5.5.2 GeoIPS command line workflow

FROM COMMAND LINE: Switch to new branch, Make changes as usual

- Navigate to repository of your choice
  - Issue only needs to be created on a single repository
  - You can create branches and make changes on any number of repos, as appropriate.
  - Related changes on different repositories will all be linked to the same Issue.
- Switch to new branch, and make changes as appropriate
  - *Ensure you copy and paste git fetch and git checkout commands when creating branch above*
  - Switch to new branch: Paste git fetch / git checkout commands specified when creating branch from Issue
    - git fetch origin
    - git checkout <new_branch_name>
  - <Make changes to code>
  - git commit # Frequently commit your changes
- Use enforced commit message format for all commits
  - Please follow Commit Message Template
  - Summary line <= 120 characters
  - Blank line (if commit message is more than one line)
  - OPTIONAL: additional details
  - Issue ID
- Update CHANGELOG.md in each repository with changes related to this Issue
  - Before pushing your final changes to GitHub and creating a pull request, you MUST update CHANGELOG.md appropriately
  - Please follow CHANGELOG Template
  - You will Copy and paste CHANGELOG modifications directly into the "Summary" section of pull request.
  - If CHANGELOG.md is not updated appropriately, pull request will be rejected.

- **Create test scripts and associated outputs for any new functionality**
  - **Ensure any new functionality is tested in:**
    - <repo>/tests/scripts/<test_name>.sh
  - **Ensure new test scripts are included in:**
    - <repo>/tests/test_all.sh

Push changes to github

- **From command line**: When you have made all of the changes required for the current Issue, push changes to GitHub
  - Perform once for each repository with changes related to this Issue
  - git push

5.5.3 GeoIPS GitHub Pull Request workflow

FROM WEB: Create pull request from new ticket branch to "dev" branch

Follow these instructions for each repo that requires changes for a given Issue.

- IF NEEDED: Navigate to Issue URL via web browser, and finalize with any last minute notes or resolutions
  - IF APPROPRIATE: Summary of overall changes
  - IF APPROPRIATE: Complete testing instructions (if multiple repostiories involved)
  - IF APPROPRIATE: Complete test output (if multiple repositories involved)
- Create a Pull Request on each repo with changes associated with the current Issue ID
  - Click on "Pull requests" tab within current repo
  - Click green "New pull request" button
  - **Source**: <new_branch_name>
  - **Destination**: main
  - Click green "Create pull request" button
- Fill Auto-populated template with appropriate content:
  - Generated from Global Pull Request Template
  - **Important to follow template title and contents directions for ease of review**
  - **Pull request will be denied if template is not followed appropriately**

- Ensure appropriate tags and attributes are set on the pull request
  - **Reviewers**: *Identify at least two Reviewers*
  - **Labels**: As appropriate
  - **Projects**: **VERY IMPORTANT**: *Select "GeoIPS - * *All Repos and All Functionality*, additional Projects if desired.
- Click "Create"
  - Now wait for the automated emails from GitHub saying your changes have been approved and merged.

5.5.4 GeoIPS Merge PR and Close Issue workflow

FROM WEB: Merge pull requests

This could include related pull requests from multiple repositories.

- Navigate to GeoIPS Project
- Find current Issue card - Issue will have all associated PRs linked
- CTRL-Click on each linked pull request
  - Click green "Merge branch" from each PR
  - Click "Delete head branch" from each PR
- This should automatically update the Project so all related PRs and Issues are moved to the "Done" columnn

FROM WEB: Ensure all Issues and Pull Requests were successfully closed

- Could require approvals / merges from multiple pull requests in multiple repos before closing Issue
- Navigate to GeoIPS Project
- Ensure all related Issue and Pull Request cards were automatically moved to the "Done" column

  # # # Distribution Statement A. Approved for public release. Distribution unlimited.
  # # #
  # # # Author:
  # # # Naval Research Laboratory, Marine Meteorology Division
  # # #
  # # # This program is free software: you can redistribute it and/or modify it under

the terms of the NRLMMD License included with this program. This program is
distributed WITHOUT ANY WARRANTY; without even the implied warranty of
MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the included license
for more details. If you did not receive the license, for more information see:
https://github.com/U-S-NRL-Marine-Meteorology-Division/

5.6 Documentation and Style Strategy

GeoIPS uses Sphinx with the Napoleon extension for automated documentation generation.

https://www.sphinx-doc.org/en/master/usage/extensions/napoleon.html

The **geoips/docs** directory contains high level restructured text (rst) format documentation (including this page), along with a **build_docs.sh** script that will setup sphinx and build complete documentation from the high level rst files as well as docstrings contained within the GeoIPS source code.

5.6.1 GeoIPS Syntax and Style Checking

GeoIPS uses the NumPy docstring format within the code base for simplicity:

https://numpydoc.readthedocs.io/en/latest/format.html

bandit, flake8, and black are used to enforce appropriate style, security, and syntax usage. flake8-rst and flake8-rst-docstring plugins are used to enforce numpy docstring formatting.

The installation script called from **geoips/README.md** contains steps for setting up VIM8 with automated syntax checking and highlighting (including automated flake8, pylint, and bandit error / warning highlighting), to help enforce desired style guides.

VSCode plugins are also available to provide automated syntax checking and highlighting.

Distribution Statement A. Approved for public release. Distribution unlimited.
# # #
Author:
Naval Research Laboratory, Marine Meteorology Division
# # #
This program is free software: you can redistribute it and/or modify it under
the terms of the NRLMMD License included with this program. This program is
distributed WITHOUT ANY WARRANTY; without even the implied warranty of

MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the included license
for more details. If you did not receive the license, for more information see:
https://github.com/U-S-NRL-Marine-Meteorology-Division/

5.7 Xarray and NetCDF Metadata Standards

All GeoIPS readers read data into xarray Datasets - a separate dataset for each shape/resolution of data - and contain standard metadata information for standardized processing.

Readers should return a dictionary of the resulting xarray Datasets, with human readable keys for the different datasets (no standard for dictionary key names).

5.7.1 Xarray Standard Variables

Internal to GeoIPS, our xarray standards include the following variables for automated temporal and spatial sectoring.

- 'latitude' - REQUIRED 2d array the same shape as data variables
- 'longitude' - REQUIRED 2d array the same shape as data variables
- 'time' - OPTIONAL 2d array the same shape as data variables

NOTE: Additional methods of storing spatial and temporal information will be implemented in the future for efficiency, but currently latitude and longitude arrays are strictly required, and time array is required for automated temporal sectoring

5.7.2 Xarray Standard Attributes

The following standard attributes are used internally to GeoIPS for consistent generation of titles, legends, regridding, etc

- 'source_name' - REQUIRED
- 'platform_name' - REQUIRED
- 'data_provider' - REQUIRED
- 'start_datetime' - REQUIRED
- 'end_datetime' - REQUIRED
- 'interpolation_radius_of_influence' - REQUIRED used for pyresample-based interpolation

The following optional attributes can be used within processing if available.

- 'source_file_names' - OPTIONAL
  - list of strings containing names of all files that went into the current dataset. To ensure consistent output between users, these file names can either be
    - base paths, including only the filename and excluding the path altogether, or
    - full paths with GeoIPS environment variables replacing specific paths (ie, $GEOIPS_OUTDIRS, $GEOIPS_TESTDATA_DIR, etc)
- 'source_file_attributes' - OPTIONAL
  - attribute associated with the list of source files.
  - dictionary with name of each source file as keys, and attributes specific to that source file as values.
- 'source_file_datetimes' - OPTIONAL
  - list of datetime objects corresponding to the datetime listed in each of the 'source_file_names'. List must be same length as 'source_file_names'
- 'area_definition' - OPTIONAL
  - specify area_definition current dataset is registered to, if applicable
- 'registered_dataset' - OPTIONAL
  - True if current dataset is registered to a specific area_definition, False otherwise
- 'minimum_coverage' - OPTIONAL
  - if specified, products will not be generated with coverage < minimum_coverage
- 'sample_distance_km' - OPTIONAL
  - if specified, sample_distance_km can be used to produce a "minimum" sized image. Web images are often up sampled to provide a conveniently sized image for viewing with titles/legends, this allows producing minimal sized "clean" imagery for overlaying in external viewers (such as the Automated Tropical Cyclone Forecasting System)

5.7.3 NetCDF CF Standards

All additional attributes should follow the **NetCDF Climate and Forecast (CF) Conventions**.

Attributes and metadata on output NetCDF files should follow the **CF Metadata Conventions**

- http://cfconventions.org/Data/cf-conventions/cf-conventions-1.8/cf-conventions.html

Names of attributes describing individual products and variables in output NetCDF files should use **CF Standard Names** when available

- http://cfconventions.org/Data/cf-standard-names/76/build/cf-standard-name-table.html

```
# # # Distribution Statement A. Approved for public release. Distribution unlimited.
# # #
# # # Author:
# # # Naval Research Laboratory, Marine Meteorology Division
# # #
# # # This program is free software: you can redistribute it and/or modify it under
# # # the terms of the NRLMMD License included with this program. This program is
# # # distributed WITHOUT ANY WARRANTY; without even the implied warranty
of
# # # MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See
the included license
# # # for more details. If you did not receive the license, for more information see:
# # # https://github.com/U-S-NRL-Marine-Meteorology-Division/
```

CHAPTER

SIX

API REFERENCE

6.1 geoips package

6.1.1 Subpackages

geoips.commandline package

Submodules

geoips.commandline.args module

Command line script for kicking off geoips based procflows.

geoips.commandline.args.**add_args**(*parser*, *arglist=None*)

List of available standard arguments for calling data processing command line.

Parameters

- **parser** (*ArgumentParser*) – argparse ArgumentParser to add appropriate arguments
- **arglist** (*list, optional*) – list of requested arguments to add to the ArgumentParser, default None. if None, include all arguments

Return type

No return values (parser modified in place)

geoips.commandline.args.**check_command_line_args**(*arglist*, *argdict*)

Check formatting of command line arguments.

Parameters

- **arglist** (*list of str*) – List of desired command line arguments to check within argdict for appropriate formatting
- **argdict** (*dict*) – Dictionary of command line arguments

Returns
Return True if all arguments are of appropriate formatting.

Return type
bool

Raises
TypeError – Incorrect command line formatting

geoips.commandline.args.**get_command_line_args**(*arglist=None*, *description=None*, *add_args_func=None*, *check_args_func=None*)

Parse command line arguments specified by the requested list of arguments.

Parameters

- **arglist** (*list, optional*) – list of requested arguments to add to the ArgumentParser, default None. if None, include all arguments
- **description** (*str, optional*) – String description of arguments, default None
- **add_args_func** (*function, optional*) – Alternative "add_args" function, default None If None, use internal "add_args"
- **check_args_func** (*function, optional*) – Alternative "check_args" function, default None If None, use internal "check_args"

Returns
Dictionary of command line arguments

Return type
dict

geoips.commandline.list_available_modules module

Simple script to list available modules for each interface.

This includes both dev and stable interfaces. Note this will be deprecated with v2.0, replaced with a new class-based interface implementation.

geoips.commandline.list_available_modules.**main**()

Script to list all modules available in the current geoips instantiation.

geoips.commandline.log_setup module

Geoips module for setting up logging handlers with a specified verbosity.

geoips.commandline.log_setup.**setup_logging**(*verbose=True*)

Set up logging handler.

If you do this the first time with no argument, it sets up the logging for all submodules. Subsequently, in submodules, you can just do LOG = logging.getLogger(__name__)

geoips.commandline.run_procflow module

Command line script for kicking off geoips based procflows.

MUST call with –procflow.

geoips.commandline.run_procflow.**main**(*get_command_line_args_func=None*)

Script to kick off processing based on command line args.

Parameters

get_command_line_args_func (*function, optional*) – Function to use in place of "get_command_line_args", default None If None, use geoips.commandline.args.get_command_line_args

geoips.commandline.test_interfaces module

Simple test script to run "test_<interface>_interface" for each interface.

This includes both dev and stable interfaces. Note this will be deprecated with v2.0 - replaced with a new class-based interface implementation.

geoips.commandline.test_interfaces.**main**()

Script to test all dev and stable interfaces.

geoips.commandline.update_tc_tracks_database module

Command line script for updating the TC database.

geoips.commandline.update_tc_tracks_database.**main**()

Update TC tracks database via command line.

Module contents

geoips.commandline init file.

geoips.data_manipulations package

Submodules

geoips.data_manipulations.conversions module

Routines for converting between units.

geoips.data_manipulations.conversions.**unit_conversion**(*data_array*, *input_units=None*, *output_units=None*)

Convert array in units 'input_units' to units 'output_units'.

Parameters

- **data_array** (*ndarray*) – numpy.ndarray or numpy.MaskedArray of data values to be converted
- **input_units** (*str, optional*) – Units of input data array, defaults to None
- **output_units** (*str, optional*) – Units of output data array, defaults to None

Returns

Return numpy.ma.MaskedArray, with units converted from 'input_units' to 'output_units'

Return type

MaskedArray

geoips.data_manipulations.corrections module

Apply min/max values, normalize, and invert data arrays.

geoips.data_manipulations.corrections.**apply_data_range**(*data*, *min_val=None*, *max_val=None*, *min_outbounds='crop'*, *max_outbounds='crop'*, *norm=True*, *inverse=False*)

Apply minimum and maximum values to an array of data.

Normalize, invert, and handle out of bounds data as requested.

Parameters

- **data** (*numpy.ndarray or numpy.ma.MaskedArray*) – data values to which the data range will be applied.
- **min_val** (*float, default None*) –
  - The minimum bound to be applied to the input data as a scalar,
  - If None, use data.min().
- **max_val** (*float, default=None*) –
  - The maximum bound to be applied to the input data as a scalar.
  - If None, use data.max().
- **min_outbounds** (*str, default='crop'*) – Method to use when applying bounds as a string. Valid values are:
  - retain: keep all pixels as is
  - mask: mask all pixels that are out of range.
  - crop: set all out of range values to min_val
- **max_outbounds** (*str, default='crop'*) – Method to use when applying bounds as a string. Valid values are:
  - retain: keep all pixels as is
  - mask: mask all pixels that are out of range.
  - crop: set all out of range values to max_val
- **norm** (*bool, default=True*) – Boolean flag indicating whether to normalize (True) or not (False).
  - If True, returned data will be in the range from 0 to 1.
  - If False, returned data will be in the range min_val to max_val.
- **inverse** (*bool, default=False*) – Boolean flag indicating whether to invert data (True) or not (False).
  - If True, returned data will be inverted
  - If False, returned data will not be inverted

Returns

Return numpy.ndarray or numpy.ma.MaskedArray Input data array with values above 'max_val' or below 'min_val' retained, cropped, or masked.

Return type
numpy.ndarray

geoips.data_manipulations.corrections.**apply_gamma**(*data_array*, *gamma*)

Apply gamma correction to all values in the data array.

Gamma correction applied as: data_array ** (1.0 / float(gamma))

Parameters

- **data_array** (*numpy.ndarray or numpy.ma.MaskedArray*) – data array to which gamma will be applied
- **gamma** (*float*) – gamma correction value

Returns
Return numpy.ndarray or numpy.ma.MaskedArray if data_array was MaskedArray with gamma correction applied data_array ** (1.0 / float(gamma))

Return type
numpy.ndarray

geoips.data_manipulations.corrections.**apply_maximum_value**(*data*, *max_val*, *outbounds*)

Apply maximum value to an array of data.

Parameters

- **data** (*numpy.ndarray or numpy.ma.MaskedArray*) – data values to which the maximum value will be applied.
- **max_val** (*float*) – The maximum bound to be applied to the input data as a scalar.
- **outbounds** (*str*) –

  Method to use when applying bounds as a string. Valid values are:
  retain: keep all pixels as is mask: mask all pixels that are out of range. crop: set all out of range values to max_val.

Returns
Return numpy.ndarray or numpy.ma.MaskedArray Input data array with values above 'max_val' retained, cropped, or masked appropriately.

Return type
numpy.ndarray

geoips.data_manipulations.corrections.**apply_minimum_value**(*data*, *min_val*, *outbounds*)

Apply minimum values to an array of data.

Parameters

- **data** (*numpy.ndarray or numpy.ma.MaskedArray*) – data values to which the minimum value will be applied.
- **min_val** (*float*) – The minimum bound to be applied to the input data as a scalar.
- **outbounds** (*str*) –

  Method to use when applying bounds as a string. Valid values are:
  retain: keep all pixels as is mask: mask all pixels that are out of range. crop: set all out of range values to min_val.

Returns

Return numpy.ndarray or numpy.ma.MaskedArray Input data array with values below 'min_val' retained, cropped, or masked appropriately.

Return type

numpy.ndarray

geoips.data_manipulations.corrections.**apply_offset**(*data_array*, *offset*)

Apply offset to all values in data_array.

Offset applied as: data_array + offset

Parameters

- **data_array** (*numpy.ndarray or numpy.ma.MaskedArray*) – data values to which offset will be applied.
- **scale_factor** (*float*) – requested offset.

Returns

Return numpy.ndarray or numpy.ma.MaskedArray Input data array with offset applied data_array + offset

Return type

numpy.ndarray

geoips.data_manipulations.corrections.**apply_scale_factor**(*data_array*, *scale_factor*)

Apply scale factor to all values in data_array.

Scale factor applied as: data_array * scale_factor

Parameters

- **data_array** (*numpy.ndarray or numpy.ma.MaskedArray*) – data values to be scaled
- **scale_factor** (*float*) – requested scale factor

Returns
Return numpy.ndarray or numpy.ma.MaskedArray Input data array with scale factor applied data_array * scale_factor

Return type
numpy.ndarray

geoips.data_manipulations.corrections.**apply_solar_zenith_correction**(*data_array*, *sunzen_array*)

Apply solar zenith angle correction to all values in data_array.

Solar zenith correction applied as: data / cos(sunzen)

Parameters

- **data_array** (*numpy.ndarray or numpy.ma.MaskedArray*) – data values to be masked
- **sunzen_array** (*numpy.ndarray or numpy.ma.MaskedArray*) – solar zenith angles of the same shape as the data array.

Returns
Return numpy.ndarray or numpy.ma.MaskedArray if original data_array was MaskedArray with each value in the data_array divided by cos(sunzen).

Return type
numpy.ndarray

geoips.data_manipulations.corrections.**invert_data_range**(*data*, *min_val=None*, *max_val=None*)

Invert data range to an array of data.

Parameters

- **data** (*numpy.ndarray or numpy.ma.MaskedArray*) – data values to which the data range will be applied.
- **min_val** (*float, optional*) – The minimum bound to be applied to the input data as a scalar, by default None, which results in data.min().
- **max_val** (*float, optional*) – The maximum bound to be applied to the input data as a scalar. by default None, which results in data.max().

Returns
Return numpy.ndarray or numpy.ma.MaskedArray Input data array with values inverted.

Return type
numpy.ndarray

geoips.data_manipulations.corrections.**mask_day**(*data_array*, *sunzen_array*, *max_zenith=90*)

Mask where solar zenith angle less than the maxinum specified value.

Mask all pixels within the data array where the solar zenith angle is less than the maxinum specified value.

Parameters

- **data_array** (*numpy.ndarray or numpy.ma.MaskedArray*) – data values to be masked
- **sunzen_array** (*numpy.ndarray*) – numpy.ndarray or numpy.ma.MaskedArray of solar zenith angles, of the same shape as the data array
- **max_zenith** (*float, optional*) – Mask all locations in data_array where sunzen_array is less than max_zenith, by default 90

Returns

Data array with all locations corresponding to a solar zenith angle less than max_zenith masked.

Return type

numpy.ma.MaskedArray

geoips.data_manipulations.corrections.**mask_night**(*data_array*, *sunzen_array*, *min_zenith=90*)

Mask where solar zenith angle greater than the minimum specified value.

Mask all pixels within the data array where the solar zenith angle is greater than the mininum specified value.

Parameters

- **data_array** (*numpy.ndarray or numpy.ma.MaskedArray*) – data values to be masked.
- **sunzen_array** (*numpy.ndarray or numpy.ma.MaskedArray*) – array of solar zenith angles, same shape as the data array.
- **min_zenith** (*float, optional*) – Mask all locations in data_array where sunzen_array is greater than min_zenith, by default 90.

Returns

Data array with all locations corresponding to a solar zenith angle greater than min_zenith masked.

Return type

numpy.ma.MaskedArray

geoips.data_manipulations.corrections.**normalize**(*data*, *min_val=None*, *max_val=None*, *min_bounds='crop'*, *max_bounds='crop'*)

Normalize data array with min_val and max_val to range 0 to 1.

Default to cropping outside requested data range.

Parameters

- **data** (*numpy.ndarray or numpy.ma.MaskedArray*) – data values to which the data range will be applied.
- **min_val** (*float, default=None*) –
  - The minimum bound to be applied to the input data as a scalar,
  - If None, use data.min().
- **max_val** (*float, default=None*) –
  - The maximum bound to be applied to the input data as a scalar.
  - If None, use data.max().
- **min_outbounds** (*str, default='crop'*) – Method to use when applying bounds as a string. Valid values are:
  - retain: keep all pixels as is
  - mask: mask all pixels that are out of range.
  - crop: set all out of range values to min_val
- **max_outbounds** (*str, default='crop'*) – Method to use when applying bounds as a string. Valid values are:
  - retain: keep all pixels as is
  - mask: mask all pixels that are out of range.
  - crop: set all out of range values to max_val

Returns

Return numpy.ndarray or numpy.ma.MaskedArray Input data array normalized between 0 and 1, with values above 'max_val' or below 'min_val' retained, cropped, or masked.

Return type

numpy.ndarray

geoips.data_manipulations.info module

Introspection functions on data arrays.

geoips.data_manipulations.info.**percent_not_nan**(*data_array*)

Determine percent of a numpy.ndarray that is not NaN values.

Parameters
data_array (*numpy.ndarray*) – Final processed array from which to determine coverage, invalid values specified by "numpy.nan".

Returns
percent of input data array that is not numpy.nan.

Return type
float

geoips.data_manipulations.info.**percent_unmasked**(*data_array*)

Determine percent of a numpy.ma.Masked array that is not masked.

Parameters
data_array (*numpy.ma.MaskedArray*) – Final processed array from which to determine coverage

Returns
percent of input data array that is not masked.

Return type
float

geoips.data_manipulations.merge module

Utilities for merging granules into a single data array.

These utilities can apply to potentially different data sources - spanning a variety of sensors and platforms into a single final dataset.

geoips.data_manipulations.merge.**daterange**(*start_date*, *end_date*)

Check one day at a time.

If end_date - start_date is between 1 and 2, days will be 1, and range(1) is 0. So add 2 to days to set range.

geoips.data_manipulations.merge.**find_datafiles_in_range**(*sector_name*, *platform_name*, *source_name*, *min_time*, *max_time*, *basedir*, *product_name*, *every_min=True*, *verbose=False*, *time_format='%H%M'*, *actual_datetime=None*, *single_match=False*)

Find datafiles from a specified set of parameters.

Parameters

- **sector_name** (*str*) – Sector of interest
- **platform_name** (*str*) – platform of interest
- **source_name** (*str*) – Source of interest
- **min_time** (*datetime.datetime*) – Minimum time to search
- **max_time** (*datetime.datetime*) – Maximum time to search
- **basedir** (*str*) – Base directory to search
- **product_name** (*str*) – Product of interest
- **every_min** (*bool, optional*) – Check every minute, by default True
- **verbose** (*bool, optional*) – Print a lot of log output during the search, by default False
- **time_format** (*str, optional*) – Format of time information in filenames, by default "%H%M"
- **actual_datetime** (*datetime.datetime, optional*) – Actual datetime of the requested data, required if single_match is True, by default None
- **single_match** (*bool, optional*) – Only return the closest matching file if True, else return all matching files, by default False

Returns

List of all filenames matching the given parameters (list of length 1 if single_match is True, all matching files if single_match is false)

Return type

list of str

geoips.data_manipulations.merge.**get_matching_files**(*primary_sector_name*, *subsector_names*, *platforms*, *sources*, *max_time_diffs*, *basedir*, *merge_datetime*, *product_name*, *time_format='%H%M'*, *buffer_mins=30*, *verbose=False*, *single_match=False*)

Given the current set of parameters, find all matching files.

Given the current primary sector, and associated subsectors, platforms, and sources, find all matching files.

Parameters

- **primary_sector_name** (*str*) – The final sector that all data will be stitched into. ie 'GlobalGlobal'
- **subsector_names** (*list of str*) – List of all subsectors that will be merged into the final sector. (potentially including the full primary_sector_name.) ie ['GlobalGlobal', 'GlobalAntarctic', 'GlobalArctic']
- **platforms** (*list of str*) – List of all desired platforms. platforms, sources, and max_time_diffs correspond to one another and should be the same length and in the same order.
- **sources** (*list of str*) – List of all desired sources. platforms, sources, and max_time_diffs correspond to one another and should be the same length and in the same order.
- **max_time_diffs** (*list of int*) – Minutes. List of allowed time diffs for given platform/source. Matches max_time_diff before the requested merge_datetime argument. platforms, sources, and max_time_diffs correspond to one another and should be the same length and in the same order.
- **basedir** (*str*) – Base directory in which to look for the matching files.
- **merge_datetime** (*datetime*) – Attempt matching max_time_diff prior to merge_datetime
- **product_name** (*str*) – product_name string found in matching files
- **time_format** (*str, optional*) – Requested time format for filenames (strptime format string), by default '%H%M'
- **verbose** (*bool, optional*) – Print a lot of log output during the search, by default False

- **single_match** (*bool, optional*) – Only return the closest matching file if True, else return all matching files, by default False

Returns
List of all filenames matching the given parameters (list of length 1 if single_match is True, all matching files if single_match is false)

Return type
list of str

geoips.data_manipulations.merge.**hourrange**(*start_date*, *end_date*)

Check one hour at a time.

geoips.data_manipulations.merge.**minrange**(*start_date*, *end_date*)

Check one minute at a time.

Module contents

geoips.data_manipulations init file.

geoips.dev package

Submodules

geoips.dev.output_config module

Interpolation interface will be deprecated v2.0.

Wrapper functions for geoips output_config specifications.

This functionality will be replaced with a class-based implementation v2.0, and deprecated at that time.

geoips.dev.output_config.**get_filename_formatter_kwargs**(*filename_formatter*, *output_dict*)

Interface will be deprecated v2.0.

Return dictionary of filename_formatters_kwargs.

based on what was passed in via the YAML output config dictionary, as well as default kwargs.

If "filename_formatter_kwargs (singular) is passed command line, use that to override ALL filename_formatters_kwargs specified in YAML output config.

geoips.dev.output_config.**get_filename_formatters**(*output_dict*)

Interface will be deprecated v2.0.

geoips.dev.output_config.**get_metadata_filename_formatter**(*filename_formatter*, *output_dict*)

Interface will be deprecated v2.0.

geoips.dev.output_config.**get_metadata_filename_formatter_kwargs**(*filename_formatter*, *output_dict*)

Interface will be deprecated v2.0.

Return dictionary of filename_formatters_kwargs.

based on what was passed in via the YAML output config dictionary, as well as default kwargs

geoips.dev.output_config.**get_metadata_output_formatter**(*output_dict*)

Interface will be deprecated v2.0.

geoips.dev.output_config.**get_metadata_output_formatter_kwargs**(*output_dict*)

Interface will be deprecated v2.0.

geoips.dev.output_config.**get_minimum_coverage**(*product_name*, *output_dict*)

Interface will be deprecated v2.0.

geoips.dev.output_config.**get_output_config_type**(*output_config_dict*)

Interface will be deprecated v2.0.

Retrieve output_config_type of the passed output_config_dict, found in:
output_config_dict['output_config_type']

See: geoips.dev.output_config.is_valid_output_config for full list of supported output_config types.

Parameters
output_config_dict (*dict*) – dictionary of complete output config parameters

Returns
(str)

Return type
output_config type, found in output_config_dict['output_config_type']

geoips.dev.output_config.**get_output_formatter**(*output_dict*)

Interface will be deprecated v2.0.

geoips.dev.output_config.**get_output_formatter_kwargs**(*output_dict*, *xarray_obj=None*, *area_def=None*, *sector_type=None*, *bg_files=None*, *bg_xarrays=None*, *bg_product_name=None*)

Interface will be deprecated v2.0.

geoips.dev.output_config.**is_valid_output_config**(*output_config_dict*)

Interface will be deprecated v2.0.

Check that requested output_config dictionary is properly formatted.

The dictionary of output_config parameters fully determines the outputs required for a given set of data files.

Dictionary of output_config parameters currently specified by a full path to a YAML file: and requested via commandline with: `--output_config <full_path_to_YAML_output_config>`

Parameters

output_config_dict (*dict*) – Dictionary of output config parameters

Returns

is_valid –

- True if `output_config_dict` is a properly formatted dictionary of output parameters.
- **False if output_config_dict:**
  - does not contain supported `output_config_type`,
  - does not contain all `required` fields,
  - contains non-supported `optional` fields

Return type

bool

Notes

output_config_types currently one of:

- `single_source`
- `fused`

geoips.dev.output_config.**produce_current_time**(*config_dict*, *metadata_xobj*, *output_dict_keys=None*)

Interface will be deprecated v2.0.

Determine if the current data file needs to be processed,
based on the requested times.

If output_dict_key is included, apply to only the currently
requested output_dict.

If output_dict_key is None, check ALL outputs to determine if
ANY need the current time.

geoips.dev.output_config.**set_lonlat_spacing**(*gridline_annotator*, *area_def*)

Interface will be deprecated v2.0.

geoips.dev.output_config.**test_output_config_interface**(*output_config_dict*)

Interface will be deprecated v2.0.

Finds and opens every product params dict available within the current geoips instantiation

See geoips.dev.output_config.is_valid_output_config?
for a list of available product params dict types and associated call signatures / return values.

Returns
List of all successful output_config information

Return type
list

geoips.dev.product module

Interpolation interface will be deprecated v2.0.

Wrapper functions for geoips product specifications.

This functionality will be replaced with a class-based implementation v2.0, and deprecated at that time.

geoips.dev.product.**get_cmap_from_product**(*prod_plugin*, *output_dict=None*)

Interface will be deprecated v2.0.

Retrieve colormapper information, based on requested product and source

Parameters

- **product_name** (*str*) – Name of requested product (ie, 'IR-BD', '89H', 'color89Nearest', etc)

- **source_name** (*str*) – Name of requested source (ie, 'ahi', 'modis', etc)

Returns
cmap_func(cmap_args)** – Return actual colormapper information

Return type
function

See also:

geoips.dev.check_cmap_func
additional information on colormapper types, arguments, and return values

geoips.dev.product.**get_covg_args_from_product**(*prod_plugin*, *output_dict=None*, *covg_field=None*)

Interface will be deprecated v2.0.

Retrieve coverage check function args, based on requested product and source

Parameters

- **product_name** (*str*) – Name of requested product (ie, 'IR-BD', '89H', 'color89Nearest', etc)
- **source_name** (*str*) – Name of requested source (ie, 'ahi', 'modis', etc)

Returns
covg_func_name – Return dictionary of coverage args required for given product/source

Return type
str

See also:

geoips.dev.check_cmap_func
additional information on colormapper types, arguments, and return values

geoips.dev.product.**get_covg_from_product**(*prod_plugin*, *output_dict=None*, *covg_field=None*)

Interface will be deprecated v2.0.

Retrieve coverage check function name, based on requested product and source

Parameters

- **product_name** (*str*) – Name of requested product (ie, 'IR-BD', '89H', 'color89Nearest', etc)
- **source_name** (*str*) – Name of requested source (ie, 'ahi', 'modis', etc)

Returns
covg_func_name – Return name of coverage function required for given product/source

Return type
str

See also:

`geoips.dev.check_cmap_func`
additional information on colormapper types, arguments, and return values

`geoips.dev.product.get_data_range(prod_plugin, output_dict=None)`

Interface will be deprecated v2.0.

Retrieve required data range for requested product

Parameters
product_name (*str*) – Name of requested product (ie, 'IR-BD', '89H', 'color89Nearest', etc)

Returns
data_range – List of float specifying min and max value for the output product `<geoips_package>.algorithms.<algorithm_name>.alg_params['output_data_range']`

Return type
list

`geoips.dev.product.get_product_display_name(prod_plugin, output_dict=None)`

Interface will be deprecated v2.0.

Retrieve product display name. For titles, etc.

Parameters

- **product_name** (*str*) – Name of requested product (ie, 'IR-BD', '89H', 'color89Nearest', etc)
- **source_name** (*str*) – Name of requested source (ie, 'ahi', 'modis', etc)

Returns
product_display_name – Return display name for given product

Return type
str

See also:

`geoips.dev.check_cmap_func`
additional information on colormapper types, arguments, and return values

geoips.dev.product.**get_requested_datasets_for_variables**(*prod_plugin*)

Interface will be deprecated v2.0.

Retrieve required datasets if specified for product variables, based on requested product and source

Within product_inputs YAML specifications, variables can be requested with <DATASET>:<VARNAME> if you need a particular variable from a specific dataset.

If <DATASET>: is not specified, the first variable found when looping through the datasets is used.

Parameters

- **product_name** (*str*) – Name of requested product (ie, 'IR-BD', '89H', 'color89Nearest', etc)
- **source_name** (*str*) – Name of requested source (ie, 'ahi', 'modis', etc)

Returns

datasets_for_variable –

- Dictionary of
  - {'<variable_name>': '<requested_dataset>'} OR
  - {'variable_type': {'<variable_name>': '<requested_dataset>'}

Return type
dict

geoips.dev.product.**get_required_variables**(*prod_plugin*)

Interface will be deprecated v2.0.

Return required variables names for the input product plugin. If variables are combined with their dataset name, the dataset name will be stripped and only the variable names will be returned.

Parameters
prod_plugin (*ProductPlugin*) – An instance of the GeoIPS ProductPlugin class.

Returns

required_variables –

- If list: List of strings specifying required variables.
- If dict: Dictionary of variable types of lists of variable names
  - {'<variable_type>': ['var1', 'var2', ... , 'varn']}

Return type
list or dict

Module contents

dev init file.

geoips.filenames package

Submodules

geoips.filenames.base_paths module

Collection of base path names used throughout GeoIPS.

Everything defaults to subdirectories relative to the REQUIRED environment variable GEOIPS_OUTDIRS.

Individual GEOIPS_OUTDIRS relative paths can be overridden by setting appropriate environment variables.

`geoips.filenames.base_paths.`**`make_dirs`**`(`*`path`*`)`

Make directories, catching exceptions if directory already exists.

Parameters
path (*str*) – Path to directory to create

Returns
Path if successfully created

Return type
str

geoips.filenames.duplicate_files module

Module to handle removing duplicate files, based on filename formats.

If an individual filename format has a method named "<filename_formatter>_remove_duplicates" defined, use that method to remove duplicates for the given current filename.

`geoips.filenames.duplicate_files.`**`remove_duplicates`**`(`*`fnames`*`, `*`remove_files=False`*`)`

Remove duplicate files from all filenames included in dict fnames.

Parameters

- **fnames** (*dict*) – Dictionary with individual filenames as keys, and a field named "filename_formatter" which indicates the filename format used to generate the given filename.
- **remove_files** (*bool, optional*) – Specify whether to remove files (True), or just list what would have been removed, default to False

Returns

- **removed_files** (*list*) – List of files that were removed.
- **saved_files** (*list*) – List of files that were not removed.

Module contents

geoips.filenames init file.

geoips.image_utils package

Submodules

geoips.image_utils.colormap_utils module

Module for generating specific colormaps on the fly.

geoips.image_utils.colormap_utils.**create_linear_segmented_colormap**(*cmapname*, *min_val*, *max_val*, *transition_vals*, *transition_colors*)

Create a LinearSegmentedColormap instance.

Parameters

- **cmapname** (*str*) – Name to attach to the matplotlib.color ColorMap object
- **min_val** (*float*) – Overall minimum value for the colormap Range must be normalized between 0 and 1
- **max_val** (*float*) – Overall maximum value for the colormap Range must be normalized between 0 and 1

- **transition_vals** (*array-like*) – A list of value ranges specified as tuples for generating a specific range of colors ie [(0, 10), (10, 30), (30, 60)]
- **transition_colors** (*array-like*) – A list of color ranges specified as tuples for generating a specific range of colors corresponding to the transition_vals (see Notes below)

Returns
cm – matplotlib colormap object

Return type
LinearSegmentedColormap

Notes

Transition colors specified as:

```
[('yellow', 'orange'),
 ('pink', 'red'),
 ('violet', 'purple')]
```

Where:

```
TRANSITIONPOINT1 = 0.0
TRANSITIONPOINT4 = 1.0
cmdict = { 'red' :  ((TRANSITIONPOINT1, IGNORED, 1to2STARTCOLOR),
                     (TRANSITIONPOINT2, 1to2ENDCOLOR, 2to3STARTCOLOR),
                     (TRANSITIONPOINT3, 2to3ENDCOLOR, 3to4STARTCOLOR),
                     (TRANSITIONPOINT4, 3to4ENDCOLOR, IGNORED)),
         'green' :  ((TRANSITIONPOINT1, IGNORED, 1to2STARTCOLOR),
                     (TRANSITIONPOINT2, 1to2ENDCOLOR, 2to3STARTCOLOR),
                     (TRANSITIONPOINT3, 2to3ENDCOLOR, 3to4STARTCOLOR),
                     (TRANSITIONPOINT4, 3to4ENDCOLOR, IGNORED)),

         'blue' :  ((TRANSITIONPOINT1, IGNORED, 1to2STARTCOLOR),
                     (TRANSITIONPOINT2, 1to2ENDCOLOR, 2to3STARTCOLOR),
                     (TRANSITIONPOINT3, 2to3ENDCOLOR, 3to4STARTCOLOR),
                     (TRANSITIONPOINT4, 3to4ENDCOLOR, IGNORED)),
    }
```

geoips.image_utils.colormap_utils.**from_ascii**(*fname*, *cmap_name=None*, *reverse=False*)

Create a ListedColormap instance from an ASCII file of RGB values.

Parameters

- **fname** (*str*) – Full path to ascii RGB colortable file
- **cmap_name** (*str, default=None (basename(fname))*) – Identifying name of colormap - if None, default to basename(fname)
- **reverse** (*bool, default=False*) – If True, reverse the colormap

Returns
cmap – If cmap_name not specified, the colormap name will be the os.path.basename of the file.

Return type
ListedColormap object

Notes

- Lines preceded by '#' are ignored.
- 0-255 or 0-1.0 RGB values (0-255 values are normalized to 0-1.0 for matplotlib usage)
- One white space delimited RGB value per line

geoips.image_utils.colormap_utils.**set_matplotlib_colors_rgb**()

Create matplotlib Colors parameters dictionary.

For rgb imagery, we require no color information (it is entirely specified by the RGB(A) arrays).

Returns
mpl_colors_info – Specifies matplotlib Colors parameters for use in both plotting and colorbar generation. For RGBA arrays, all fields are "None".

Return type
dict

geoips.image_utils.colormap_utils.**set_matplotlib_colors_standard**(*data_range*, *cmap_name='Greys'*, *cbar_label=None*, *create_colorbar=True*)

Set the matplotlib colors information.

For use in colorbar and image production.

Parameters
- **data_range** (*list*) – the minimum and maximum value for the data range [min_val, max_val]
- **cmap_name** (*str, default='Greys'*) – Specify the standard matplotlib colormap

- **cbar_label** (*str, optional*) – If specified, use cbar_label string as colorbar label
- **create_colorbar** (*bool, default=True*) – Specify whether the image should contain a colorbar or not

Returns

mpl_colors_info – Specifies matplotlib Colors parameters for use in both plotting and colorbar generation. See geoips.image_utils.mpl_utils.create_colorbar for field descriptions.

Return type

dict

geoips.image_utils.colormap_utils.**set_mpl_colors_info_dict**(*cmap*, *norm*, *cbar_ticks*, *cbar_tick_labels=None*, *boundaries=None*, *cbar_label=None*, *cbar_spacing='proportional'*, *create_colorbar=True*, *cbar_full_width=False*)

Create the mpl_colors_info dictionary directly from passed arguments.

Parameters

- **cmap** (*Colormap*) – This is a valid matplotlib cm Colormap object that can be used for both plotting and colorbar creation.
- **norm** (*Normalize*) – This is a valid matplotlib Normalize object that can be used for both plotting and colorbar creation.
- **cbar_ticks** (*list*) – List of values where tick marks should be placed on colorbar
- **cbar_tick_labels** (*list, optional*) – List of tick label values
- **boundaries** (*list, optional*) – List of boundaries to use in matplotlib plotting and colorbar creation
- **cbar_label** – The label for the colorbar
- **optional** – The label for the colorbar
- **cbar_spacing** (*str, default='proportional'*) – One of 'proportional' or 'uniform'
- **create_colorbar** (*bool, default=True*) – True if colorbar should be created with the set of color info, False otherwise

- **cbar_full_width** (*bool, default=False*) – True if colorbar should be full width of figure, center 50% if False

Returns
mpl_colors_info – Dictionary of mpl_colors_info for use in plotting and colorbar creation.

Return type
dict

geoips.image_utils.maps module

matplotlib geographic map (basemap or cartopy) utilities.

geoips.image_utils.maps.**check_feature_annotator**(*feature_annotator*)

Check that the provided feature_annotator plugin has all required fields.

Parameters
feature_annotator (*YamlPlugin*) – A feature annotator plugin.

Raises
ValueError – if any field is missing

See also:

geoips.image_utils.maps.get_feature_annotator
For complete list of fields, and appropriate defaults

geoips.image_utils.maps.**check_gridline_annotator**(*gridline_annotator*)

Check gridlines_info dictionary for that all required fields.

Parameters
gridline_annotator (*YamlPlugin*) – A gridline annotator plugin instance.

Raises
ValueError – If required field is missing

geoips.image_utils.maps.**compute_lat_auto_spacing**(*area_def*)

Compute automatic spacing for latitude lines based on area definition.

geoips.image_utils.maps.**compute_lon_auto_spacing**(*area_def*)

Compute automatic spacing for longitude lines based on area definition.

geoips.image_utils.maps.**draw_features**(*mapobj*, *curr_ax*, *feature_annotator*, *zorder=None*)

Draw cartopy features.

Draw features on specified cartopy map instance, based on specs found in the feature_annotator plugin.

Parameters

- **mapobj** (*map object*) – CRS object for plotting map features
- **curr_ax** (*matplotlib.axes._axes.Axes*) – matplotlib Axes object for plotting map features
- **feature_annotator** (*dict*) – Dictionary of parameters for plotting map features
- **zorder** (*int, optional*) – The matplotlib zorder

See also:

geoips.image_utils.maps.check_feature_annotator
for required dictionary entries and defaults.

geoips.image_utils.maps.**draw_gridlines**(*mapobj*, *area_def*, *curr_ax*, *gridline_annotator*, *zorder=None*)

Draw gridlines on map object.

Draw gridlines on a cartopy map object, as specified by a gridline_annotator plugin instance

Parameters

- **mapobj** (*map object*) – CRS object for plotting gridlines
- **area_def** (*AreaDefinition*) – pyresample AreaDefinition object
- **curr_ax** (*matplotlib.axes._axes.Axes*) – matplotlib Axes object for plotting gridlines
- **gridline_annotator** (*YamlPlugin*) – A gridline_annotator plugin instance
- **zorder** (*int, optional*) – The matplotlib zorder value

See also:

geoips.image_utils.maps.get_gridlines_info_dict
For complete list of fields, and appropriate default

geoips.image_utils.maps.**ellps2axis**(*ellps_name*)

Get semi-major and semi-minor axis from ellipsis definition.

Parameters
ellps_name (*str*) – Standard name of ellipsis

Returns

- **avar** (*float*) – semi-major axis
- **bvar** (*float*) – semi-minor axis

geoips.image_utils.maps.**is_crs**(*mapobj*)

Determine if the map object we are using is a cartopy crs instance.

Parameters
mapobj (*map object*) – Basemap or cartopy._PROJ4Projection object

Returns
crs – True if it is a CRS object, False otherwise.

Return type
bool

geoips.image_utils.maps.**meridians**(*area_def*, *grid_size*)

Calculate the meridians (longitude) that are within the input sector.

Parameters

- **area_def** (*AreaDefinition*) – pyresample AreaDefinition
- **grid_size** (*float*) – grid spacing in degrees

Returns
meridians_to_draw – longitude locations for gridlines

Return type
list

geoips.image_utils.maps.**parallels**(*area_def*, *grid_size*)

Calculate the parallels (latitude) that fall within the input sector.

Parameters

- **area_def** (*AreaDefinition*) – pyresample AreaDefinition
- **grid_size** (*float*) – grid spacing in degrees

Returns
lat_ticks – latitude locations for gridlines

Return type
list

geoips.image_utils.maps.**set_gridlines_info_dict**(*gridlines_info*, *area_def*)

Set plotting gridlines.

Set the final values for gridlines plotting params, pulling from argument and defaults.

Parameters

- **gridlines_info** (*dict*) – Dictionary of parameters for plotting gridlines. The following fields are available. If a field is not included in the dictionary, the field is added to the return dictionary and the default is used.
- **area_def** (*AreaDefinition*) – pyresample AreaDefinition

Returns

use_gridlines_info – gridlines_info dictionary, with fields as specified above.

Return type

dict

Notes

Defaults specified as:

```
gridlines_info['grid_lat_spacing']      default auto calculated 5␣
↪lat grid lines
gridlines_info['grid_lon_spacing']      default auto calculated 5␣
↪lon grid lines
gridlines_info['grid_lat_xoffset']      default None (0.01 * image␣
↪height)
gridlines_info['grid_lon_xoffset']      default None (0.01 * image␣
↪width)
gridlines_info['grid_lat_yoffset']      default None (0.01 * image␣
↪height)
gridlines_info['grid_lon_yoffset']      default None (0.01 * image␣
↪width)
gridlines_info['grid_lat_fontsize']     default None (plot fontsize)
gridlines_info['grid_lon_fontsize']     default None (plot fontsize)
gridlines_info['left_label']            default True
gridlines_info['right_label']           default False
gridlines_info['top_label']             default True
gridlines_info['bottom_label']          default False
gridlines_info['grid_lat_linewidth']    default 1
gridlines_info['grid_lon_linewidth']    default 1
gridlines_info['grid_lat_color']        default 'black'
gridlines_info['grid_lon_color']        default 'black'
gridlines_info['grid_lat_dashes']       default [4, 2]
gridlines_info['grid_lon_dashes']       default [4, 2]
```

geoips.image_utils.mpl_utils module

matplotlib utilities.

geoips.image_utils.mpl_utils.**alpha_from_masked_arrays**(*arrays*)

Convert from arrays to alpha.

Return an alpha transparency array based on the masks from a list of masked arrays. 0=transparent, 1=opaque

Parameters

arrays (*numpy.ndarray*) – list of numpy masked arrays, must all be the same shape

Returns

alp – the alpha transparency layer in matplotlib, values between 0 and 1, where 0 is fully transparent and 1 is fully opaque

Return type

numpy.ndarray

geoips.image_utils.mpl_utils.**create_colorbar**(*fig*, *mpl_colors_info*)

Routine to create a single colorbar.

Parameters

- **fig** (*matplotlib.figure.Figure*) – Figure object to attach the colorbar - the colorbar will create its own ax
- **mpl_colors_info** (*dict*) – Dictionary of matplotlib Color information, required fields in Notes below.

Returns

cbar – This will have all the pertinent information for ensuring plot and colorbar use the same parameters

Return type

matplotlib.colorbar.Colorbar

Notes

Required mpl_colors_info fields:

```
mpl_colors_info['cmap'] (Colormap):
    matplotlib.colors.Colormap object (LinearSegmentedColormap, etc)
    this is used to plot the image and to generated the colorbar
mpl_colors_info['norm'] (Normalize):
    matplotlib.colors.Normalize object (BoundaryNorm, Normalize, etc)
```

(continues on next page)

(continued from previous page)

```
    again, this should be used to plot the data also
mpl_colors_info['cbar_ticks'] (list):
    list of floats - values requiring tick marks on the colorbar
mpl_colors_info['cbar_tick_labels'] (list)
    list of values to use to label tick marks, if other than
    found in cbar_ticks
mpl_colors_info['boundaries'] (list):
    if cmap_norm is BoundaryNorm, list of boundaries for discrete␣
 ↪colors
mpl_colors_info['cbar_spacing'] (string):
    DEFAULT 'proportional', 'uniform' or 'proportional'
mpl_colors_info['cbar_label'] (string):
    string label for colorbar
mpl_colors_info['colorbar']: (bool)
    True if a colorbar should be included in the image, False if no␣
 ↪cbar
```

Colorbar set as:

```
cbar_ax = fig.add_axes([<cbar_start_pos>, <cbar_bottom_pos>,
                        <cbar_width>, <cbar_height>])
cbar = fig.colorbar(mappable=matplotlib.cm.ScalarMappable(norm=cmap_
 ↪norm,
                                                           cmap=mpl_
 ↪cmap),
                    cax=cbar_ax,
                    norm=cmap_norm,
                    boundaries=cmap_boundaries,
                    spacing=cbar_spacing,
                    **cbar_kwargs)
cbar.set_ticks(cbar_ticks, labels=cbar_tick_labels, **set_ticks_
 ↪kwargs)
if cbar_label:
    cbar.set_label(cbar_label, **set_label_kwargs)
```

geoips.image_utils.mpl_utils.**create_figure_and_main_ax_and_mapobj**(*x_size*, *y_size*, *area_def*, *font_size=None*, *existing_mapobj=None*, *noborder=False*)

Create a figure of x pixels horizontally and y pixels vertically.

Use information from matplotlib.rcParams.

Parameters

- **x_size** (*int*) – number pixels horizontally xsize = (float(x_size)/dpi)/(right_margin - left_margin)
- **y_size** (*int*) – number pixels vertically ysize = (float(y_size)/dpi)/(top_margin - bottom_margin)
- **font_size** (*int*) – matplotlib font size
- **area_def** (*AreaDefinition*) – pyresample AreaDefinition object - used for initializing map object (basemap or cartopy)
- **existing_mapobj** (*CRS or basemap, optional*) – If specified, do not regenerate mapobj. If None, create CRS or basemap object from specified area_def.
- **noborder** (*bool, default=False*) – If true, use [0, 0, 1, 1] for axes (allowing for image exact shape of sector).

Returns

- **fig** (*matplotlib.figure.Figure*) – matplotlib Figure object to subsequently use for plotting imagery / colorbars / etc
- **main_ax** (*matplotlib.axes._axes.Axes*) – matplotlib Axes object corresponding to the single main plotting area.
- **mapobj** (*mapobject*) – cartopy crs or Basemap object for plotting

geoips.image_utils.mpl_utils.**get_title_string_from_objects**(*area_def*, *xarray_obj*, *product_name_title*, *product_datatype_title=None*, *bg_xarray=None*, *bg_product_name_title=None*, *bg_datatype_title=None*, *title_copyright=None*, *title_formatter=None*)

Get the title from object information.

Parameters

- **area_def** (*AreaDefinition*) – pyresample AreaDefinition object specifying the area covered by the current plot

- **xarray_obj** (*xarray.Dataset*) – data used to produce product
- **product_name_title** (*str*) – name to display for the title
- **product_datatype_title** (*str, optional*) – the data type
- **bg_xarray** (*xarray, optional*) – data used for background
- **bg_product_name_title** (*str, optional*) – background product title
- **bg_datatype_title** (*str, optional*) – background data type
- **title_copyright** (*str, optional*) – string for copyright
- **title_formatter** (*str, optional*) – format for title

Returns
title_string – the title to use for matplotlib

Return type
str

geoips.image_utils.mpl_utils.**percent_unmasked_rgba**(*rgba*)

Convert to percent.

Return percentage of array that is NOT fully transparent / masked (ie, non-zero values in the 4th dimension)

Parameters
rgba (*numpy.ndarray*) – 4 dimensional array where the 4th dimension is the alpha transparency array: 1 is fully opaque, 0 is fully transparent

Returns
coverage – Coverage in percentage, between 0 and 100.

Return type
float

geoips.image_utils.mpl_utils.**plot_image**(*main_ax*, *data*, *mapobj*, *mpl_colors_info*, *zorder=None*)

Plot the "data" array and map in the matplotlib "main_ax".

Parameters

- **main_ax** (*Axes*) – matplotlib Axes object for plotting data and overlays
- **data** (*numpy.ndarray)*) – Numpy array of data to plot
- **mapobj** (*Map Object*) – Basemap or Cartopy CRS instance
- **mpl_colors_info** (*dict*) – Specifies matplotlib Colors parameters for use in both plotting and colorbar

See also:

geoips.image_utils.mpl_utils.create_colorbar
for field descriptions for mpl_colors_info

geoips.image_utils.mpl_utils.**plot_overlays**(*mapobj*, *curr_ax*, *area_def*, *feature_annotator=None*, *gridline_annotator=None*, *features_zorder=None*, *gridlines_zorder=None*)

Plot specified coastlines and gridlines on the matplotlib axes.

Parameters

- **mapobj** (*map object*) – Basemap or CRS object for boundary and gridline plotting.
- **ax** (*matplotlib.axes._axes.Axes*) – matplotlib Axes object for boundary and gridline plotting.
- **area_def** (*AreaDefinition*) – pyresample AreaDefinition object specifying the area covered by the current plot
- **feature_annotator** (*YamlPlugin*) – A feature annotator plugin instance.
- **gridline_annotator** (*YamlPlugin*) – A gridlines annotator plugin instance.

geoips.image_utils.mpl_utils.**remove_duplicates**(*fname*, *min_range*)

Not implemented.

geoips.image_utils.mpl_utils.**rgba_from_arrays**(*red*, *grn*, *blu*, *alp=None*)

Return rgba from red, green, blue, and alpha arrays.

Parameters

- **red** (*numpy.ndarray*) – red gun values
- **grn** (*numpy.ndarray*) – green gun values
- **blu** (*numpy.ndarray*) – blue gun values
- **alp** (*numpy.ndarray, optional*) – alpha values 1 is fully opaque, 0 is fully transparent If none, calculate alpha from red, grn, blu guns

Returns

rgba – 4 layer dimensional numpy.ndarray

Return type

numpy.ndarray

geoips.image_utils.mpl_utils.**save_image**(*fig*, *out_fname*, *is_final=True*, *image_datetime=None*, *remove_duplicate_minrange=None*, *savefig_kwargs=None*)

Save the image specified by the matplotlib figure "fig" to the filename out_fname.

Parameters

- **fig** (*matplotlib.figure.Figure*) – Figure object that needs to be written to a file.
- **out_fname** (*str*) – full path to the output filename
- **is_final** (*bool, default=True*) – Final imagery must set_axis_on for all axes. Non-final imagery must be transparent with set_axis_off for all axes, and no pad inches.

Notes

No return values (image is written to disk and IMAGESUCCESS is written to log file)

geoips.image_utils.mpl_utils.**set_fonts**(*figure_y_size*, *font_size=None*)

Set the fonts in the matplotlib.rcParams dictionary, using matplotlib.rc.

Parameters

figure_y_size (*int*) – Font size set relative to number of pixels in the y direction

geoips.image_utils.mpl_utils.**set_title**(*ax*, *title_string*, *figure_y_size*, *xpos=None*, *ypos=None*, *fontsize=None*)

Set the title on figure axis "ax" to string "title_string".

Parameters

- **ax** (*Axes*) – matplotlib.axes._axes.Axes object to add the title
- **title_string** (*str*) – string specifying title to attach to axis "ax"
- **figure_y_size** (*int*) – vertical size of the image, used to proportionally set the title size
- **xpos** (*float, optional*) – x position of the title
- **ypos** (*float, optional*) – y position of the title
- **fontsize** (*int, optional*) – matplotlib font size

Module contents

image_utils init file.

geoips.interfaces package

Subpackages

geoips.interfaces.module_based package

Submodules

geoips.interfaces.module_based.algorithms module

Algorithms interface module.

class geoips.interfaces.module_based.algorithms.**AlgorithmsInterface**

Bases: *BaseModuleInterface*

GeoIPS interface for algorithms plugins.

name = 'algorithms'

required_args = {'channel_combination': ['arrays'], 'list_numpy_to_numpy': ['arrays'], 'rgb': ['arrays'], 'scalar_to_scalar': [], 'single_channel': ['arrays'], 'xarray_dict_area_def_to_numpy': ['xarray_dict', 'area_def'], 'xarray_dict_dict_to_xarray': ['xarray_dict_dict'], 'xarray_dict_to_xarray': ['xarray_dict'], 'xarray_dict_to_xarray_dict': ['xarray_dict'], 'xarray_to_numpy': ['xobj']}

required_kwargs = {'channel_combination': [], 'list_numpy_to_numpy': [], 'rgb': [], 'scalar_to_scalar': ['value'], 'single_channel': [], 'xarray_dict_area_def_to_numpy': [], 'xarray_dict_dict_to_xarray': [], 'xarray_dict_to_xarray': [], 'xarray_dict_to_xarray_dict': [], 'xarray_to_numpy': []}

geoips.interfaces.module_based.colormappers module

Colormappers interface module.

class geoips.interfaces.module_based.colormappers.**ColormappersInterface**

Bases: *BaseModuleInterface*

GeoIPS interface for colormappers plugins.

allowable_kwargs = {'matplotlib': ['data_range', 'cmap_name', 'ascii_path', 'create_colorbar', 'cbar_label', 'cbar_ticks', 'cbar_tick_labels', 'cbar_spacing', 'cbar_full_width', 'colorbar_kwargs', 'set_ticks_kwargs', 'set_label_kwargs']}

name = 'colormappers'

required_args = {'matplotlib': []}

required_kwargs = {'matplotlib': []}

geoips.interfaces.module_based.coverage_checkers module

Interpolators interface module.

class geoips.interfaces.module_based.coverage_checkers.**CoverageCheckersInterface**

Bases: *BaseModuleInterface*

GeoIPS interface for coverage_checkers plugins.

allowable_kwargs = {'standard': {'area_def', 'radius_km'}}

get_plugin_for_product(*product*, *checker_field='coverage_checker'*)

Get plugin for product.

name = 'coverage_checkers'

required_args = {'standard': ['xarray_obj', 'variable_name']}

required_kwargs = {'standard': {}}

geoips.interfaces.module_based.filename_formatters module

Filename formatters interface module.

class geoips.interfaces.module_based.filename_formatters.**FilenameFormattersInterface**

Bases: *BaseModuleInterface*

GeoIPS interface for filename_formatters plugins.

find_duplicates(**args*, ***kwargs*)

Find duplicate files.

name = 'filename_formatters'

remove_duplicates()

Remove duplicate files.

required_args = {'data': ['area_def', 'xarray_obj', 'product_names'], 'standard': ['area_def', 'xarray_obj', 'product_name'], 'standard_metadata': ['area_def', 'xarray_obj', 'product_filename'], 'xarray_metadata_to_filename': ['xarray_obj']}

required_kwargs = {'data': ['coverage', 'output_type', 'output_type_dir', 'product_dir', 'product_subdir', 'source_dir', 'basedir'], 'standard': ['coverage', 'output_type', 'output_type_dir', 'product_dir', 'product_subdir', 'source_dir', 'basedir'], 'standard_metadata': ['metadata_dir', 'metadata_type', 'basedir'], 'xarray_metadata_to_filename': ['extension', 'basedir']}

geoips.interfaces.module_based.interpolators module

Interpolators interface module.

class geoips.interfaces.module_based.interpolators.**InterpolatorsInterface**

Bases: *BaseModuleInterface*

GeoIPS interface for interpolators plugins.

name = 'interpolators'

required_args = {'2d': ['area_def', 'input_xarray', 'output_xarray', 'varlist'], 'grid': ['area_def', 'input_xarray', 'output_xarray', 'varlist']}

required_kwargs = {'2d': ['array_num'], 'grid': ['array_num']}

geoips.interfaces.module_based.output_formatters module

Output formatters interface module.

class
geoips.interfaces.module_based.output_formatters.**OutputFormattersInterface**

Bases: *BaseModuleInterface*

GeoIPS interface for output_formatters plugins.

name = 'output_formatters'

required_args = {'image': ['area_def', 'xarray_obj', 'product_name', 'output_fnames'], 'image_multi': ['area_def', 'xarray_obj', 'product_names', 'output_fnames', 'mpl_colors_info'], 'image_overlay': ['area_def', 'xarray_obj', 'product_name', 'output_fnames'], 'standard_metadata': ['area_def', 'xarray_obj', 'metadata_yaml_filename', 'product_filename'], 'unprojected': ['xarray_obj', 'product_name', 'output_fnames'], 'xarray_data': ['xarray_obj', 'product_names', 'output_fnames'], 'xrdict_area_product_outfnames_to_outlist': ['xarray_dict', 'area_def', 'product_name', 'output_fnames'], 'xrdict_area_product_to_outlist': ['xarray_dict', 'area_def', 'product_name'], 'xrdict_area_varlist_to_outlist': ['xarray_dict', 'area_def', 'varlist'], 'xrdict_varlist_outfnames_to_outlist': ['xarray_dict', 'varlist', 'output_fnames']}

required_kwargs = {'image': ['product_name_title', 'mpl_colors_info', 'existing_image'], 'image_multi': ['product_name_titles'], 'image_overlay': ['product_name_title', 'clean_fname', 'mpl_colors_info', 'clean_fname', 'feature_annotator', 'gridline_annotator', 'clean_fname', 'product_datatype_title', 'clean_fname', 'bg_data', 'bg_mpl_colors_info', 'clean_fname', 'bg_xarray', 'bg_product_name_title', 'bg_datatype_title', 'clean_fname', 'remove_duplicate_minrange'], 'standard_metadata': ['metadata_dir', 'basedir', 'output_dict'], 'unprojected': ['product_name_title', 'mpl_colors_info'], 'xarray_data': [], 'xarray_dict_data': ['append', 'overwrite'], 'xarray_dict_to_image': [], 'xrdict_area_product_outfnames_to_outlist': [], 'xrdict_area_product_to_outlist': [], 'xrdict_area_varlist_to_outlist': [], 'xrdict_varlist_outfnames_to_outlist': []}

geoips.interfaces.module_based.procflows module

Procflows interface module.

class geoips.interfaces.module_based.procflows.**ProcflowsInterface**

Bases: *BaseModuleInterface*

GeoIPS interface for procflows plugins.

name = 'procflows'

required_args = {'standard': ['fnames']}

required_kwargs = {'standard': ['command_line_args']}

geoips.interfaces.module_based.readers module

Readers interface module.

class geoips.interfaces.module_based.readers.**ReadersInterface**

Bases: *BaseModuleInterface*

GeoIPS interface for readers plugins.

name = 'readers'

required_args = {'standard': ['fnames']}

required_kwargs = {'standard': ['metadata_only', 'chans', 'area_def', 'self_register']}

geoips.interfaces.module_based.sector_adjusters module

Sector adjusters interface module.

class geoips.interfaces.module_based.sector_adjusters.**SectorAdjustersInterface**

Bases: *BaseModuleInterface*

GeoIPS interface for sector_adjusters plugins.

name = 'sector_adjusters'

required_args = {'list_xarray_list_variables_to_area_def_out_fnames': ['xobjs', 'area_def', 'variables']}

```
required_kwargs =
{'list_xarray_list_variables_to_area_def_out_fnames':
['recenter_variables']}
```

geoips.interfaces.module_based.sector_metadata_generators module

Sector metadata generators interface module.

class geoips.interfaces.module_based.sector_metadata_generators.**SectorMetadataGeneratorsInterface**

Bases: *BaseModuleInterface*

GeoIPS interface for sector_metadata_generators plugins.

```
name = 'sector_metadata_generators'
```

```
required_args = {'tc':  ['trackfile_name']}
```

```
required_kwargs = {'tc':  []}
```

geoips.interfaces.module_based.sector_spec_generators module

Sector spec generators interface module.

class geoips.interfaces.module_based.sector_spec_generators.**SectorSpecGeneratorsInterface**

Bases: *BaseModuleInterface*

GeoIPS interface for sector_spec_generators plugins.

```
name = 'sector_spec_generators'
```

```
required_args = {'area_definition':  []}
```

```
required_kwargs = {'area_definition':  []}
```

geoips.interfaces.module_based.title_formatters module

Title formatters interface module.

class geoips.interfaces.module_based.title_formatters.**TitleFormattersInterface**

Bases: *BaseModuleInterface*

GeoIPS interface for title_formatters plugins.

```
name = 'title_formatters'

required_args = {'standard':  []}

required_kwargs = {'standard':  []}
```

Module contents

Module based interfaces init file.

geoips.interfaces.yaml_based package

Submodules

geoips.interfaces.yaml_based.feature_annotators module

Feature Annotator interface module.

class geoips.interfaces.yaml_based.feature_annotators.**FeatureAnnotatorsInterface**

Bases: *BaseYamlInterface*

Interface for feature annotator plugins.

name = 'feature_annotators'

geoips.interfaces.yaml_based.gridline_annotators module

Gridline Annotator interface module.

class geoips.interfaces.yaml_based.gridline_annotators.**GridlineAnnotatorsInterface**

Bases: *BaseYamlInterface*

Interface for gridline annotator plugins.

name = 'gridline_annotators'

geoips.interfaces.yaml_based.product_defaults module

Product Defaults interface module.

class geoips.interfaces.yaml_based.product_defaults.**ProductDefaultsInterface**

Bases: *BaseYamlInterface*

Default values that can be applied to products.

name = 'product_defaults'

geoips.interfaces.yaml_based.products module

Products interface module.

class geoips.interfaces.yaml_based.products.**ProductsInterface**

Bases: *BaseYamlInterface*

GeoIPS interface for Products plugins.

get_plugin(*source_name*, *name*, *product_spec_override=None*)

Retrieve a Product plugin by source_name, name, and product_spec_override.

If product_spec_override dict is passed, values contained within product_spec_override will be used in place of those found in products list and product_defaults.

product_spec_override[product_name] matches the format of the product "spec" field.

Additionall, if the special key product_spec_override["all"] is included, it will apply to all products not specified by name within the dictionary.

get_plugins()

Retrieve a plugin by name.

name = 'products'

plugin_is_valid(*source_name*, *name*)

Test that the named plugin is valid.

test_interface()

Test interface method.

validator = <geoips.interfaces.yaml_based.products.ProductsPluginValidator object>

class geoips.interfaces.yaml_based.products.**ProductsPluginValidator**

Bases: *YamlPluginValidator*

Validator for Products plugins.

This differs from other validators for two reasons:

1. Most plugins are identified solely by 'name'. Products are identified by 'source_name' and 'name.
2. Most plugins supply their 'family' directly. Products can supply it directly, but can, alternatively, specify a 'product_defaults' plugin from which to pull 'family' and most other properties. This validator handles filling in a product plugin based on its specified product defaults plugin.

validate(*plugin*, *validator_id=None*)

Validate a Products plugin against the relevant schema.

The relevant schema is determined based on the interface and family of the plugin.

validate_product(*product*)

Validate single product.

geoips.interfaces.yaml_based.sectors module

Sector interface module.

class geoips.interfaces.yaml_based.sectors.**SectorsInterface**

Bases: *BaseYamlInterface*

Interface for sector plugins.

name = 'sectors'

Module contents

YAML based interfaces init file.

Submodules

geoips.interfaces.base module

Base classes for interfaces, plugins, and plugin validation machinery.

class geoips.interfaces.base.**BaseInterface**

Bases: object

Base class for GeoIPS interfaces.

This class should not be instantiated directly. Instead, interfaces should be accessed by importing them from geoips.interfaces. For example: ` from geoips.interfaces import algorithms ` will retrieve an instance of AlgorithmsInterface which will provide access to the GeoIPS algorithm plugins.

class geoips.interfaces.base.**BaseModuleInterface**

Bases: *BaseInterface*

Base Class for GeoIPS Interfaces.

This class should not be instantiated directly. Instead, interfaces should be accessed by importing them from geoips.interfaces. For example: ` from geoips.interfaces import algorithms ` will retrieve an instance of AlgorithmsInterface which will provide access to the GeoIPS algorithm plugins.

get_plugin(*name*)

Retrieve a plugin from this interface by name.

Parameters
name (*str*) – The name the desired plugin.

Returns

- An object of type <interface>Plugin where <interface> is the name of
- *this interface.*

Raises
PluginError – If the specified plugin isn't found within the interface.

get_plugins()

Get a list of plugins for this interface.

plugin_is_valid(*name*)

Check that an interface is valid.

Check that the requested interface function has the correct call signature. Return values should be as specified below, but are not programmatically verified.

Parameters
name (*str*) – Name of the interface to be validated

Returns
True if valid, False if invalid

Return type
bool

`plugins_all_valid()`

Test the current interface by validating every Plugin.

Return type
True if all plugins are valid, False if any plugin is invalid.

`test_interface()`

Test the current interface by validating each Plugin and testing each method.

Test this interface by opening every Plugin available to the interface, and validating each plugin by calling *plugin_is_valid* for each. Additionally, ensure all methods of this interface work as expected:

- get_plugins
- get_plugin
- plugin_is_valid
- plugins_all_valid

Returns

- *A dictionary containing three keys*
- *'by_family', 'validity_check', 'func', and 'family'. The value for each*
- *of these keys is a dictionary whose keys are the names of the Plugins.*
- *- 'by_family' contains a dictionary of plugin names sorted by family.*
- *- 'validity_check' contains a dict whose keys are plugin names and whose* – values are bools where *True* indicates that the Plugin's function is valid according to *plugin_is_valid*.
- *- 'func' contains a dict whose keys are plugin names and whose values are* – the function for each Plugin.
- *- 'family' contains a dict whose keys are plugin names and whose vlaues* – are the contents of the 'family' attribute for each Plugin.

`class geoips.interfaces.base.BaseModulePlugin`

Bases: `object`

Base class for GeoIPS plugins.

`class geoips.interfaces.base.BaseYamlInterface`

Bases: *BaseInterface*

Base class for GeoIPS yaml-based plugin interfaces.

This class should not be instantiated directly. Instead, interfaces should be accessed by importing them from `geoips.interfaces`. For example: ` `from geoips.interfaces import products` ` will retrieve an instance of `ProductsInterface` which will provide access to the GeoIPS products plugins.

get_plugin(*name*)

Get a plugin by its name.

This default method can be overridden to provide different search functionality for an interface. An example of this is in the ProductsInterface which uses a tuple containing 'source_name' and 'name'.

get_plugins()

Retrieve a plugin by name.

plugin_is_valid(*name*)

Plugin is valid method.

plugins_all_valid()

Plugins all valid method.

test_interface()

Test interface method.

validator = <geoips.interfaces.base.YamlPluginValidator object>

class geoips.interfaces.base.**BaseYamlPlugin**(**args*, ***kwargs*)

Bases: `dict`

Base class for GeoIPS plugins.

class geoips.interfaces.base.**YamlPluginValidator**

Bases: `object`

PluginValidator class.

schemas = {'bases.docstring': {'$id': 'bases.docstring', 'description': 'A docstring that describes the plugin following numpy docstring style', 'type': 'string'}, 'bases.family': {'$id': 'bases.family', '$ref': 'bases.valid_identifier', 'description': 'The family that the plugin belongs to within its interface'}, 'bases.interface': {'$id': 'bases.interface', '$ref': 'bases.valid_identifier', 'description': 'The name of the interface that the plugin belongs to'}, 'bases.name': {'$id': 'bases.name', '$ref': 'bases.valid_identifier', 'description': 'The name of the plugin', 'type': 'string'}, 'bases.product_defaults': {'$id': 'bases.product_defaults', '$ref': 'bases.valid_identifier', 'description': 'The name of the product defaults plugin to use'}, 'bases.top': {'$id': 'bases.top', 'properties': {'abspath': {'type': 'string'}, 'docstring': {'$ref': 'bases.docstring'}, 'family': {'$ref': 'bases.family'}, 'interface': {'$ref': 'bases.interface'}, 'name': {'$ref': 'bases.name'}, 'package': {'type': 'string'}, 'relpath': {'type': 'string'}}, 'required': ['interface', 'family', 'name', 'docstring'], 'type': 'object'}, 'bases.valid_identifier': {'$id': 'bases.valid_identifier', 'description': 'A valid Python identifier', 'type': 'string'}, 'feature_annotators.cartopy': {'$id': 'feature_annotators.cartopy', '$ref': 'bases.top', 'description': 'Defines which geographical boundaries should be drawn over the output products and\nhow they should be drawn. Can enable coastlines, countries, states, and rivers and\ncontrol their line width and color.\n', 'properties': {'spec': {'default': {}, 'properties': {'borders': {'if': {'properties': {'enabled': {'const': True}}}, 'properties': {'enabled': {'type': 'boolean'}}, 'then': {'properties': {'edgecolor': {'type': 'string'}, 'linewidth': {'type': 'number'}}, 'required': ['edgecolor', 'linewidth']}, 'type': 'object', 'unevaluatedProperties': False}, 'coastline': {'if': {'properties': {'enabled': {'const': True}}}, 'properties': {'enabled': {'type': 'boolean'}}, 'then': {'properties': {'edgecolor': {'type': 'string'}, 'linewidth': {'type': 'number'}}, 'required': ['edgecolor', 'linewidth']}, 'type': 'object', 'unevaluatedProperties': False}, 'rivers': {'if': {'properties': {'enabled': {'const': True}}}, 'properties': {'enabled': {'type': 'boolean'}}, 'then': {'properties': {'edgecolor': {'type': 'string'}, 'linewidth': {'type': 'number'}}, 'required': ['edgecolor', 'linewidth']}, 'type': 'object', 'unevaluatedProperties': False}, 'states': {'if': {'properties': {'enabled': {'const': True}}}, 'properties': {'enabled': {'type': 'boolean'}}, 'then': {'properties': {'edgecolor': {'type': 'string'}, 'linewidth': {'type': 'number'}}, 'required': ['edgecolor', 'linewidth']}, 'type': 'object', 'unevaluatedProperties': False}}, 'required': ['coastline', 'borders', 'rivers', 'states'], 'type': 'object', 'unevaluatedProperties': False}}, 'required': ['spec'], 'title': 'Feature Annotators'}, 'gridline_annotators.cartopy': {'$id': 'gridline_annotators.cartopy', '$ref': 'bases.top', 'description':

validate(*plugin*, *validator_id=None*)

Validate a YAML plugin against the relevant schema.

The relevant schema is determined based on the interface and family of the plugin.

validate_list(*plugin*)

Validate a list of YAML plugins.

Some interfaces allow a 'list' family. These list plugins will contain a property that is the same as the interface's name. Under that is a list of individual plugins.

This function will add the interface property to each plugin in the list, then validate each plugin.

validators = {'bases.docstring': Draft202012Validator(schema={'$id': 'bases.docstring', 'description': 'A docstring ...cstring style', 'type': 'string'}, format_checker=None), 'bases.family': Draft202012Validator(schema={'$id': 'bases.family', '$ref': 'bases.valid_identifier', 'description': 'The family t...its interface'}, format_checker=None), 'bases.interface': Draft202012Validator(schema={'$id': 'bases.interface', '$ref': 'bases.valid_identifier', 'description': 'The name of ...in belongs to'}, format_checker=None), 'bases.name': Draft202012Validator(schema={'$id': 'bases.name', '$ref': 'bases.valid_identifier', 'description': 'The name of the plugin', 'type': 'string'}, format_checker=None), 'bases.product_defaults': Draft202012Validator(schema={'$id': 'bases.product_defaults', '$ref': 'bases.valid_identifier', 'description': 'The name of ...plugin to use'}, format_checker=None), 'bases.top': Draft202012Validator(schema={'$id': 'bases.top', 'properties': {'abspath': {'type': 'string'}, 'docstring': {'$ref': 'bases.docstring'}, 'family': {'$ref': 'bases.family'}, 'interface': {'$ref': 'bases.interface'}, ...}, 'required': ['interface', 'family', 'name', 'docstring'], 'type': 'object'}, format_checker=None), 'bases.valid_identifier': Draft202012Validator(schema={'$id': 'bases.valid_identifier', 'description': 'A valid Python identifier', 'type': 'string'}, format_checker=None), 'feature_annotators.cartopy': Draft202012Validator(schema={'$id': 'feature_annotators.cartopy', '$ref': 'bases.top', 'description': 'Defines whic... and color.\n', 'properties': {'spec': {'default': {}, 'properties': {'borders': {'if': {'properties': {...}}, 'properties': {'enabled': {...}}, 'then': {'properties': {...}, 'required': [...]}, 'type': 'object', ...}, 'coastline': {'if': {'properties': {...}}, 'properties': {'enabled': {...}}, 'then': {'properties': {...}, 'required': [...]}, 'type': 'object', ...}, 'rivers': {'if': {'properties': {...}}, 'properties': {'enabled': {...}}, 'then': {'properties': {...}, 'required': [...]}, 'type': 'object', ...}, 'states': {'if': {'properties': {...}}, 'properties': {'enabled': {...}}, 'then': {'properties': {...}, 'required': [...]}, 'type': 'object', ...}}, 'required': ['coastline', 'borders', 'rivers', 'states'], 'type': 'object', ...}}, ...}, format_checker=None), 'gridline_annotators.cartopy': Draft202012Validator(schema={'$id': 'gridline_annotators.cartopy', '$ref': 'bases.top', 'description': 'Defines lati...he imagery.\n', 'properties': {'spec': {'properties': {'labels': {'properties': {'bottom': {...}, 'left': {...}, 'right': {...}, 'top': {...}}, 'required': ['top', 'bottom', 'left', 'right'], 'type': 'object', 'unevaluatedProperties': False}, 'lines': {'properties': {'color': {...}, 'linestyle': {...}, 'linewidth': {...}}, 'required': ['color', 'linestyle', 'linewidth'], 'type': 'object', 'unevaluatedProperties': False}, 'spacing': {'properties': {'latitude': {...}, 'longitude': {...}}, 'required': ['latitude', 'longitude'], 'type': 'object', 'unevaluatedProperties': False}},

geoips.interfaces.base.**extend_with_default**(*validator_class*)

Extend a jsonschema validator to make it respect default values.

Note: This does not pollute the input validator object. Calling *jsonschema.validators.extend* returns a new object.

This will cause the validator to fill in fields that have default values. In cases where fields with default values are contained inside a mapping, that mapping must have *default: {}* and may not have *requires*.

geoips.interfaces.base.**get_schemas**(*path*, *validator*)

Collect all of the interface schema.

geoips.interfaces.base.**get_validators**(*schema_dict*, *validator_class*)

Create validators for each schema in *schema_dict*.

Parameters
schema_dict (*dict*) – A dictionary whose keys are schema *$id* and whose values are the full schema.

Returns
A dictionary whose keys are schema *$id* and whose values are *jsonschema* validator instances.

Return type
dict

geoips.interfaces.base.**plugin_module_to_obj**(*name*, *module*, *obj_attrs={}*)

Convert a module plugin to an object.

Convert the passed module plugin into an object and return it. The returned object will be derrived from a class named <interface>Plugin where interface is the interface specified by the plugin. This class is derrived from BasePlugin.

This function is used instead of predefined classes to allow setting __doc__ and __call__ on a plugin-by-plugin basis. This allows collecting __doc__ and __call__ from the plugin modules and using them in the objects.

For a module to be converted into an object it must meet the following requirements:

- The module must define a docstring. This will be used as the docstring for the plugin class as well as the docstring for the plugin when requested on the command line. The first line will be used as a "short" description, and the full docstring will be used as a more detailed discussion of the plugin.
- The following global attributes must be defined in the module: - interface: The name of the interface that the plugin belongs to. - family: The family of plugins that the plugin belongs to within the interface. - name: The name of the plugin which must be unique within the interface.

- A callable named *call* that will be called when the plugin is used.

Parameters

- **module** (*module*) – The imported plugin module.
- **obj_attrs** (*dict, optional*) – Additional attributes to be assigned to the plugin object.

Returns

- An object of type <interface>InterfacePlugin where <interface> is the name
- *of the interface that the desired plugin belongs to.*

geoips.interfaces.base.**plugin_repr**(*obj*)

Repr plugin string.

geoips.interfaces.base.**plugin_yaml_to_obj**(*name*, *yaml_plugin*, *obj_attrs={'__doc__': 'The default products_source_name fusion plugin configuration.\n', 'abspath': '/satops/users/surratt/public_test_install/template_fusion_ 'docstring': 'The default products_source_name fusion plugin configuration.\n', 'family': 'list', 'id': 'my_layered_list', 'interface': 'products', 'name': 'my_layered_list', 'package': 'my_fusion_package', 'relpath': 'yaml/products/my_layered.yaml', 'yaml': {'abspath': '/satops/users/surratt/public_test_install/template_fusion_ 'docstring': 'The default products_source_name fusion plugin configuration.\n', 'family': 'list', 'interface': 'products', 'name': 'my_layered_list', 'package': 'my_fusion_package', 'relpath': 'yaml/products/my_layered.yaml', 'spec': {'products': [{'name': 'My-Layered-Winds', 'source_names': ['my_layered_source'], 'docstring': 'Layered winds product using default 2 colorbar placement.\n\nThis example layered image includes default colorbar placement\nfor both windspeed and ir products, and no colorbar for windbarbs.\n', 'spec': {'coverage_checker': {'plugin': {'name': 'masked_arrays', 'arguments': {'variable_name': 'windspeed:wind_speed_kts'}}}, 'mpl_colors_info': {'windbarbs': {'colorbar': False, 'colorbar_positioning': {'start_x_pos': 0.7666666666666667, 'end_x_pos': 1.0, 'start_y_pos': -0.07792207792207793, 'end_y_pos': -0.05194805194805195}}, 'windspeed': {'colorbar': True, 'colorbar_positioning': {'start_x_pos': 0.26666666666666666, 'end_x_pos': 0.7333333333333334, 'start_y_pos': -0.07792207792207793, 'end_y_pos': -0.05194805194805195}}, 'ir': {'colorbar': True, 'colorbar_positioning': {'start_x_pos': 0.0, 'end_x_pos': 0.2333333333333334, 'start_y_pos': -0.07792207792207793, 'end_y_pos': -0.05194805194805195}, 'cbar_label': 'BT (degrees C)'}}}, 'interface': 'products', 'family': 'xarray_dict_to_output_format', 'package':*

Convert a yaml plugin to an object.

Convert the passed YAML plugin into an object and return it. The returned object will be derived from a class named `<interface>Plugin` where interface is the interface specified by the plugin. This class is derived from `BasePlugin`.

This function is used instead of predefined classes to allow setting `__doc__` on a plugin-by-plugin basis. This allows collecting `__doc__` and from the plugin and using them in the objects.

For a yaml plugin to be converted into an object it must meet the following requirements:

- Must match the jsonschema spec provided for its interface.
- The plugin must have the following top-level attributes and the must not be empty. - interface: The name of the interface that the plugin belongs to. - family: The family of plugins that the plugin belongs to within the interface. - name: The name of the plugin which must be unique within the interface. - docstring: A string to be used as the object's docstring.

Module contents

GeoIPS interface module.

geoips.plugins package

Subpackages

geoips.plugins.modules package

Subpackages

geoips.plugins.modules.algorithms package

Subpackages

geoips.plugins.modules.algorithms.pmw_tb package

Submodules

geoips.plugins.modules.algorithms.pmw_tb.pmw_37pct module

Passive Microwave 37 GHz Polarization Corrected Temperature.

Data manipulation steps for the "37pct" product. This algorithm expects Brightness Temperatures in units of degrees Kelvin.

geoips.plugins.modules.algorithms.pmw_tb.pmw_37pct.**call**(*arrays*, *output_data_range=None*, *min_outbounds='crop'*, *max_outbounds='mask'*, *norm=False*, *inverse=False*)

37pct product algorithm data manipulation steps.

This algorithm expects Brightness Temperatures in units of degrees Kelvin, and returns degrees Kelvin

Parameters

arrays (*list of numpy.ndarray*) –

- **numpy.ndarray or numpy.MaskedArray of channel data, in order of sensor**
  "channels" list
- Degrees Kelvin

Returns

numpy.ndarray or numpy.MaskedArray of appropriately scaled channel data, in degrees Kelvin.

Return type
numpy.ndarray

geoips.plugins.modules.algorithms.pmw_tb.pmw_89pct module

Passive Microwave 89 GHz Polarization Corrected Temperature.

Data manipulation steps for the "89pct" product. This algorithm expects Brightness Temperatures in units of degrees Kelvin

geoips.plugins.modules.algorithms.pmw_tb.pmw_89pct.**call**(*arrays*, *output_data_range*, *min_outbounds='crop'*, *max_outbounds='mask'*, *norm=False*, *inverse=False*)

89pct product algorithm data manipulation steps.

This algorithm expects Brightness Temperatures in units of degrees Kelvin, and returns degrees Kelvin

Parameters
arrays (*list of numpy.ndarray*) –

- **list of numpy.ndarray or numpy.MaskedArray of channel data**
  and other variables, in order of sensor "variables" list
- Channel data: Degrees Kelvin

Returns
numpy.ndarray or numpy.MaskedArray of appropriately scaled channel data, in degrees Kelvin.

Return type
numpy.ndarray

geoips.plugins.modules.algorithms.pmw_tb.pmw_color37 module

Passive Microwave 37 GHz Colorized Brightness Temperature.

Data manipulation steps for the "color37" product. This algorithm expects Brightness Temperatures in units of degrees Kelvin

geoips.plugins.modules.algorithms.pmw_tb.pmw_color37.**call**(*arrays*)

color37 product algorithm data manipulation steps.

This algorithm expects Brightness Temperatures in units of degrees Kelvin, and returns red green and blue gun arrays.

Parameters
data (*list of numpy.ndarray*) –

- **list of numpy.ndarray or numpy.MaskedArray of channel data,** in order of channels list above
- Degrees Kelvin

Returns
numpy.ndarray or numpy.MaskedArray of qualitative RGBA image output

Return type
numpy.ndarray

geoips.plugins.modules.algorithms.pmw_tb.pmw_color89 module

Passive Microwave 89 GHz Colorized Brightness Temperature.

Data manipulation steps for the "color89" product. This algorithm expects Brightness Temperatures in units of degrees Kelvin

geoips.plugins.modules.algorithms.pmw_tb.pmw_color89.**call**(*arrays*)

color89 product algorithm data manipulation steps.

This algorithm expects Brightness Temperatures in units of degrees Kelvin, and returns red green and blue gun arrays.

Parameters
arrays (*list of numpy.ndarray*) –

- **list of numpy.ndarray or numpy.MaskedArray of channel data and** other variables, in order of sensor "variables" list
- Channel data: Degrees Kelvin

Returns
numpy.ndarray or numpy.MaskedArray of qualitative RGBA image output

Return type
numpy.ndarray

Module contents

geoips pmw_tb algorithm init file.

geoips.plugins.modules.algorithms.sfc_winds package

Submodules

geoips.plugins.modules.algorithms.sfc_winds.windbarbs module

Surface Winds plotted as Barbs in Knots.

Data manipulation steps for surface winds products. This algorithm expects surface wind speeds in units of kts

geoips.plugins.modules.algorithms.sfc_winds.windbarbs.**call**(*arrays*, *output_data_range=None*, *input_units=None*, *output_units=None*, *min_outbounds='crop'*, *max_outbounds='crop'*, *norm=False*, *inverse=False*)

Windbarbs product algorithm data manipulation steps.

This algorithm expects input windspeed with units "kts" and returns in "kts"

Parameters

- **arrays** (*list of numpy.ndarray*) –
  - **list of numpy.ndarray or numpy.MaskedArray of channel data,** in order of sensor "channels" list
  - kts
- **output_data_range** (*list of float, default=None*) –
  - list of min and max value for wind speeds (kts)
  - defaults to None, which results in using data.min and data.max.
- **input_units** (*str, default=None*) –
  - Units of input data, for applying necessary conversions
  - defaults to None, resulting in no unit conversions.
- **output_units** (*str, default=None*) –

- Units of output data, for applying necessary conversions
- defaults to None, resulting in no unit conversions.

- **min_outbounds** (*str, default='crop'*) –
  - Method to use when applying bounds. Valid values are:
    - retain: keep all pixels as is
    - mask: mask all pixels that are out of range
    - crop: set all out of range values to either min_val or max_val as appropriate
- **max_outbounds** (*str, default='crop'*) –
  - Method to use when applying bounds. Valid values are:
    - retain: keep all pixels as is
    - mask: mask all pixels that are out of range
    - crop: set all out of range values to either min_val or max_val as appropriate
- **norm** (*bool, default=False*) –
  - Boolean flag indicating whether to normalize (True) or not (False)
    - If True, returned data will be in the range from 0 to 1
    - If False, returned data will be in the range from min_val to max_val
- **inverse** (*bool, default=False*) –
  - Boolean flag indicating whether to inverse (True) or not (False)
    - If True, returned data will be inverted
    - If False, returned data will not be inverted

Returns

numpy.ndarray or numpy.MaskedArray of appropriately scaled channel data, dstacked as follows:

- (spd, direction, rain_flag)
- spd in kts
- direction in degrees
- rain_flag 0 or 1

Return type
numpy.ndarray

Module contents

geoips sfc_winds algorithm init.

geoips.plugins.modules.algorithms.visir package

Submodules

geoips.plugins.modules.algorithms.visir.Night_Vis module

Data manipulation steps for "Night_Vis" product, standard Version.

This algorithm expects one VIIRS channel (DNBRad) for a single channel image.

geoips.plugins.modules.algorithms.visir.Night_Vis.**call**(*arrays*, *output_data_range=None*, *scale_factor=None*, *gamma_list=None*, *input_units=None*, *output_units=None*, *min_outbounds=None*, *max_outbounds=None*, *max_night_zen=None*, *norm=None*, *inverse=None*)

Night-Vis algorithm data manipulation steps, standard version.

DNB obs for visible product.

This algorithm expects radaiance, between 0 and 2.5*10^-8

This is only for nighttime product.

Parameters

arrays (*list of numpy.ndarray*) –

- list of numpy.ndarray or numpy.MaskedArray of channel data
- Channel data: Radiance, between 0 and 2.5*10^-8

Returns

numpy.ndarray or numpy.MaskedArray of appropriately scaled channel data

Return type

numpy.ndarray

Notes

Due to a relative maximum value of the DNBRad is much larger than that of the majority pixels in moonlight/lighting situation, it could lead to a black image if the original maximum is used to normalize the data (i.e., the normlized value is close to 0). Thus, we need to setup an tuning factor to normalize the DNBRad.

We start to use 0.05 to tune the val_max in moonlight/other lighing source, 0.5 for no lighting source.

We might have to generate night-vis product only when moonlight is present (TBD).

geoips.plugins.modules.algorithms.visir.Night_Vis_GeoIPS1 module

Data manipulation steps for "Night_Vis" product, GeoIPS 1 Version.

This algorithm expects one VIIRS channel (DNBRad) for a single channel image.

geoips.plugins.modules.algorithms.visir.Night_Vis_GeoIPS1.**call**(*arrays*, *min_outbounds='crop'*, *max_outbounds='crop'*, *max_night_zen=90*)

Night Vis product algorithm data manipulation steps, GeoIPS 1 version.

This algorithm expects DNBRad in reflectance, and returns the adjusted array.

Parameters

arrays (*list of numpy.ndarray*) –

- **list of numpy.ndarray or numpy.MaskedArray of channel data,** in order of sensor "channels" list
- Degrees Kelvin

Returns

numpy.ndarray or numpy.MaskedArray of adjusted DNB output.

Return type

numpy.ndarray

Notes

It will generate a product in daytime if we do not apply the daytime check. For now, it is for both day/night.

We will decide whether this product is only for nighttime. If so, a daytime check will be required.

We may focus only on nighttime product with moonlight after additional validation (TBD).

geoips.plugins.modules.algorithms.visir.Night_Vis_IR module

Data manipulation steps for "Night_Vis_IR" product.

This algorithm expects two VIIRS channels (DNBRad and M16BT) for a RGB image

`geoips.plugins.modules.algorithms.visir.Night_Vis_IR.call`(*arrays*)

Night_Vis_IR RGB product algorithm data manipulation steps.

This algorithm expects DNBRad in reflectance and M16BT Brightness Temperatures in units of degrees Kelvin, and returns red green and blue gun arrays.

Parameters

arrays (*list of numpy.ndarray*) –

- **list of numpy.ndarray or numpy.MaskedArray of channel data,** in order of sensor "channels" list
- Degrees Kelvin

Returns

numpy.ndarray or numpy.MaskedArray of qualitative RGBA image output

Return type

numpy.ndarray

Notes

It will generate a product in daytime if we do not apply the daytime check. For now, it is for both day/night.

We will decide whether this product is only for nighttime. If so, a daytime check will be required.

We may focus only on nighttime product with moonlight after additional validation (TBD).

geoips.plugins.modules.algorithms.visir.Night_Vis_IR_GeoIPS1 module

Data manipulation steps for "Night_Vis_IR" product, GeoIPS 1 Version.

This algorithm expects two VIIRS channels (DNBRad and M16BT) for a RGB image

`geoips.plugins.modules.algorithms.visir.Night_Vis_IR_GeoIPS1.call(`*arrays*, *max_night_zen=90*`)`

Night Vis IR RGB product algorithm data manipulation steps.

This algorithm expects DNBRad in reflectance and M16BT Brightness Temperatures in units of degrees Kelvin, and returns red green and blue gun arrays.

Parameters

arrays (*list of numpy.ndarray*) –

- **list of numpy.ndarray or numpy.MaskedArray of channel data,** in order of sensor "channels" list
- Degrees Kelvin

Returns

numpy.ndarray or numpy.MaskedArray of qualitative RGBA image output

Return type

numpy.ndarray

Notes

It will generate a product in daytime if we do not apply the daytime check. For now, it is for both day/night.

We will decide whether this product is only for nighttime. If so, a daytime check will be required.

We may focus only on nighttime product with moonlight after additional validation (TBD).

Module contents

geoips visir algorithm init file.

Submodules

geoips.plugins.modules.algorithms.single_channel module

Data manipulation steps for standard "single_channel" algorithm.

Generalized algorithm to apply data manipulation steps in a standard order to apply corrections to a single channel output product.

geoips.plugins.modules.algorithms.single_channel.**call**(*arrays*, *output_data_range=None*, *input_units=None*, *output_units=None*, *min_outbounds='crop'*, *max_outbounds='crop'*, *norm=False*, *inverse=False*, *sun_zen_correction=False*, *mask_night=False*, *max_day_zen=None*, *mask_day=False*, *min_night_zen=None*, *gamma_list=None*, *scale_factor=None*)

Apply data range and requested corrections to a single channel product.

Data manipulation steps for applying a data range and requested corrections to a single channel product

Parameters

- **arrays** (*list of numpy.ndarray*) –
  - list of numpy.ndarray or numpy.MaskedArray of channel data
  - MUST be length one for single_channel algorithm.
- **output_data_range** (*list of float, default=None*) –
  - list of min and max value for output data product.
  - This is applied LAST after all other corrections/adjustments
  - If None, use data min and max.
- **input_units** (*str, default=None*) –
  - Units of input data, for applying necessary conversions
  - If None, no conversion

- **output_units** (*str, default=None*) –
  - Units of output data, for applying necessary conversions
  - If None, no conversion
- **min_outbounds** (*str, default='crop'*) –
  - Method to use when applying bounds. Valid values are:
    - retain: keep all pixels as is
    - mask: mask all pixels that are out of range
    - crop: set all out of range values to either min_val or max_val as appropriate
- **max_outbounds** (*str, default='crop'*) –
  - Method to use when applying bounds. Valid values are:
    - retain: keep all pixels as is
    - mask: mask all pixels that are out of range
    - crop: set all out of range values to either min_val or max_val as appropriate
- **norm** (*bool, default=False*) –
  - Boolean flag indicating whether to normalize (True) or not (False)
    - If True, returned data will be in the range from 0 to 1
    - If False, returned data will be in the range from min_val to max_val
- **inverse** (*bool, default=False*) –
  - Boolean flag indicating whether to inverse (True) or not (False)
    - If True, returned data will be inverted
    - If False, returned data will not be inverted
- **sun_zenith_correction** (*bool, default=False*) –
  - Boolean flag indicating whether to apply solar zenith correction (True) or not (False)
    - If True, returned data will have solar zenith correction applied (see data_manipulations.corrections.apply_solar_zenith_correction)
    - If False, returned data will not be modified based on solar zenith angle

Notes

Order of operations, based on the passed arguments, is:

1. Mask night
2. Mask day
3. Apply solar zenith correction
4. Apply gamma values
5. Apply scale factor
6. Convert units
7. Apply data range.

NOTE: If "norm=True" is specified, the "output_data_range" will NOT match the actual range of the returned data, since the normalized data will be returned between 0 and 1.

If you require a different order of operations than that specified within "single_channel" algorithm, please create a new algorithm for your desired order of operations.

Returns
numpy.ndarray or numpy.MaskedArray of appropriately scaled channel data, in units "output_units".

Return type
numpy.ndarray

Module contents

geoips algorithms init file.

geoips.plugins.modules.colormappers package

Subpackages

geoips.plugins.modules.colormappers.pmw_tb package

Submodules

geoips.plugins.modules.colormappers.pmw_tb.cmap_150H module

Module containing colormap for ~150GHz PMW products.

geoips.plugins.modules.colormappers.pmw_tb.cmap_150H.**call**(*data_range=[110, 310]*, *cbar_label='TB (K)'*)

Colormap for displaying ~150GHz PMW data.

Parameters

data_range (*list of float, default=[110, 310]*) –

- Min and max value for colormap.
- Ensure the data range matches the range of the algorithm specified for use with this colormap
- This colormap MUST include 110 and 310

Returns

mpl_colors_info – Dictionary of matplotlib plotting parameters, to ensure consistent image output

Return type

dict

geoips.plugins.modules.colormappers.pmw_tb.cmap_37H module

Module containing colormap for ~37GHz PMW products.

geoips.plugins.modules.colormappers.pmw_tb.cmap_37H.**call**(*data_range=[125, 310]*, *cbar_label='TB (K)'*)

Colormap for displaying ~37GHz PMW data.

Parameters

data_range (*list of float, default=[125, 310]*) –

- Min and max value for colormap.
- Ensure the data range matches the range of the algorithm specified for use with this colormap
- The 37GHz colormap MUST include 125 and 310

Returns

mpl_colors_info – Dictionary of matplotlib plotting parameters, to ensure consistent image output

Return type

dict

geoips.plugins.modules.colormappers.pmw_tb.cmap_37H_Legacy module

Module containing Legacy colormap for ~37GHz PMW products.

geoips.plugins.modules.colormappers.pmw_tb.cmap_37H_Legacy.**call**(*data_range=[180, 280]*, *cbar_label='TB (K)'*)

Legacy Colormap for displaying ~37GHz PMW data.

Parameters

data_range (*list of float, default=[180, 280]*) –

- Min and max value for colormap.
- Ensure the data range matches the range of the algorithm specified for use with this colormap
- The 37GHz colormap MUST include 180 and 280

Returns

mpl_colors_info – Dictionary of matplotlib plotting parameters, to ensure consistent image output

Return type

dict

geoips.plugins.modules.colormappers.pmw_tb.cmap_37H_Physical module

Module containing colormap for ~37GHz PMW products.

geoips.plugins.modules.colormappers.pmw_tb.cmap_37H_Physical.**call**(*data_range=[125, 310]*, *cbar_label='TB (K)'*)

Colormap for displaying ~37GHz PMW data.

Parameters

data_range (*list of float, default=[125, 310]*) –

- Min and max value for colormap.
- Ensure the data range matches the range of the algorithm specified for use with this colormap
- This colormap MUST include 125 and 210

Returns
mpl_colors_info – Dictionary of matplotlib plotting parameters, to ensure consistent image output

Return type
dict

geoips.plugins.modules.colormappers.pmw_tb.cmap_37pct module

Module containing colormap for 37pct product.

geoips.plugins.modules.colormappers.pmw_tb.cmap_37pct.**call**(*data_range=[230, 280]*, *cbar_label='TB (K)'*)

Colormap for displaying 37pct PMW data.

Parameters
data_range (*list of float, default=[230, 280]*) –

- Min and max value for colormap.
- Ensure the data range matches the range of the algorithm specified for use with this colormap
- The 37pct colormap MUST include 230 and 280

Returns
mpl_colors_info – Dictionary of matplotlib plotting parameters, to ensure consistent image output

Return type
dict

geoips.plugins.modules.colormappers.pmw_tb.cmap_89H module

Module containing colormap for ~89GHz PMW products.

geoips.plugins.modules.colormappers.pmw_tb.cmap_89H.**call**(*data_range=[105, 305]*, *cbar_label='TB (K)'*)

Colormap for displaying ~89GHz PMW data.

Parameters
data_range (*list of float, default=[105, 305]*) –

- Min and max value for colormap.

- Ensure the data range matches the range of the algorithm specified for use with this colormap
- This colormap MUST include 105 and 305

Returns
mpl_colors_info – Dictionary of matplotlib plotting parameters, to ensure consistent image output

Return type
dict

geoips.plugins.modules.colormappers.pmw_tb.cmap_89HW module

Module containing colormap for ~89GHz PMW products, highlighting weak convection.

`geoips.plugins.modules.colormappers.pmw_tb.cmap_89HW.call`(*data_range=[220.0, 280.0]*, *cbar_label='TB (K)'*)

Colormap for displaying ~89GHz PMW data for weak TCs.

Parameters
data_range (*list of float, default=[220, 280]*) –

- Min and max value for colormap.
- Ensure the data range matches the range of the algorithm specified for use with this colormap
- This colormap MUST include 220 and 280

Returns
mpl_colors_info – Dictionary of matplotlib plotting parameters, to ensure consistent image output

Return type
dict

geoips.plugins.modules.colormappers.pmw_tb.cmap_89H_Legacy module

Module containing Legacy colormap for ~89GHz PMW products.

`geoips.plugins.modules.colormappers.pmw_tb.cmap_89H_Legacy.call`(*data_range=[180.0, 280.0]*, *cbar_label='TB (K)'*)

Legacy Colormap for displaying ~89GHz PMW data.

Parameters
data_range (*list of float, default=[180, 280]*) –

- Min and max value for colormap.
- Ensure the data range matches the range of the algorithm specified for use with this colormap
- This colormap MUST include 180 and 280

Returns
mpl_colors_info – Dictionary of matplotlib plotting parameters, to ensure consistent image output

Return type
dict

geoips.plugins.modules.colormappers.pmw_tb.cmap_89H_Physical module

Module containing colormap for ~89GHz PMW products.

geoips.plugins.modules.colormappers.pmw_tb.cmap_89H_Physical.**call**(*data_range=[105, 305]*, *cbar_label='TB (K)'*)

Colormap for displaying ~89GHz PMW data.

Parameters
data_range (*list of float, default=[105, 305]*) –

- Min and max value for colormap.
- Ensure the data range matches the range of the algorithm specified for use with this colormap
- This colormap MUST include 105 and 305

Returns
mpl_colors_info – Dictionary of matplotlib plotting parameters, to ensure consistent image output

Return type
dict

geoips.plugins.modules.colormappers.pmw_tb.cmap_89pct module

Module containing colormap for 89pct product.

geoips.plugins.modules.colormappers.pmw_tb.cmap_89pct.**call**(*data_range=[105, 280]*, *cbar_label='TB (K)'*)

Colormap for displaying ~89GHz PMW data for weak TCs.

Parameters

data_range (*list of float, default=[105, 280]*) –

- Min and max value for colormap.
- Ensure the data range matches the range of the algorithm specified for use with this colormap
- This colormap MUST include 105 and 280

Returns

mpl_colors_info – Dictionary of matplotlib plotting parameters, to ensure consistent image output

Return type

dict

geoips.plugins.modules.colormappers.pmw_tb.cmap_Rain module

Module containing colormap for Rain Rate products.

geoips.plugins.modules.colormappers.pmw_tb.cmap_Rain.**call**(*data_range=[0.05, 50.0]*, *cbar_label='Rainrate $(mm hr^{-1})$'*)

Colormap for displaying Rain Rate products.

Parameters

data_range (*list of float, default=[0.05, 50.0]*) –

- Min and max value for colormap.
- Ensure the data range matches the range of the algorithm specified for use with this colormap
- This colormap MUST include 0.05 and 50.0

Returns
mpl_colors_info – Dictionary of matplotlib plotting parameters, to ensure consistent image output

Return type
dict

Module contents

Geoips passive microwave brightness temperature colormaps init file.

geoips.plugins.modules.colormappers.tpw package

Submodules

geoips.plugins.modules.colormappers.tpw.tpw_cimss module

Module containing tpw_cimss ASCII palette based colormap.

`geoips.plugins.modules.colormappers.tpw.tpw_cimss.call()`
Colormap for displaying data using TPW CIMSS ascii colormap.

Data range of ASCII palette is 5 to 65 mm, with transitions at 15, 25, 35, 45, and 55.

Returns
mpl_colors_info – Dictionary of matplotlib plotting parameters, to ensure consistent image output

Return type
dict

See also:

API Reference
ASCII palette is found in image_utils/ascii_palettes/tpw_cimss.txt

geoips.plugins.modules.colormappers.tpw.tpw_purple module

Module containing tpw_purple ASCII palette based colormap.

`geoips.plugins.modules.colormappers.tpw.tpw_purple.call()`
Colormap for displaying data using purple TPW ascii colormap.

Data range of ASCII palette is 5 to 65 mm, with transitions at 15, 25, 35, 45, and 55.

Returns
mpl_colors_info – Dictionary of matplotlib plotting parameters, to ensure consistent image output

Return type
dict

See also:

API Reference
ASCII palette is found in image_utils/ascii_palettes/tpw_purple.txt

geoips.plugins.modules.colormappers.tpw.tpw_pwat module

Module containing tpw_pwat ASCII palette based colormap.

`geoips.plugins.modules.colormappers.tpw.tpw_pwat.call()`
Colormap for displaying data using TPW PWAT ascii colormap.

Data range of ASCII palette is 1 to 90 mm, with numerous transitions

Returns
mpl_colors_info – Dictionary of matplotlib plotting parameters, to ensure consistent image output

Return type
dict

See also:

API Reference
ASCII palette is found in image_utils/ascii_palettes/tpw_pwat.txt

Module contents

Geoips Total Precipitable Water colormap init file.

geoips.plugins.modules.colormappers.visir package

Submodules

geoips.plugins.modules.colormappers.visir.IR_BD module

Module containing user-specified IR-BD algorithm colormap.

geoips.plugins.modules.colormappers.visir.IR_BD.**call**(*data_range=[-90.0, 40.0]*)

Colormap for displaying algorithms/visir/IR_BD.py processed data.

Parameters

data_range (*list of float, default=[-90, 40]*) –

- Min and max value for colormap.
- Ensure the data range matches the range of the algorithm specified for use with this colormap
- This colormap MUST include -90 and 40

Returns

mpl_colors_info – Dictionary of matplotlib plotting parameters, to ensure consistent image output

Return type

dict

geoips.plugins.modules.colormappers.visir.Infrared module

Module containing Infrared algorithm colormap.

geoips.plugins.modules.colormappers.visir.Infrared.**call**(*data_range=[-90, 30]*)

Colormap for displaying algorithms/visir/Infrared.py processed data.

Parameters

data_range (*list of float, default=[-90, 30]*) –

- Min and max value for colormap.
- Ensure the data range matches the range of the algorithm specified for use with this colormap
- This colormap MUST include -90 and 30

Returns

mpl_colors_info – Dictionary of matplotlib plotting parameters, to ensure consistent image output

Return type

dict

geoips.plugins.modules.colormappers.visir.WV module

Module containing WV (water vapor) algorithm colormap.

geoips.plugins.modules.colormappers.visir.WV.**call**(*data_range=[-70.0, 0.0]*)

Colormap developed for displaying algorithms/WV.py processed data.

Parameters

data_range (*list of float, default=[-70, 0]*) –

- Min and max value for colormap.
- Ensure the data range matches the range of the algorithm specified for use with this colormap
- This colormap MUST include -70 and 0

Returns

mpl_colors_info – Dictionary of matplotlib plotting parameters, to ensure consistent image output

Return type

dict

Module contents

Geoips Visible/Infrared colormap init file.

geoips.plugins.modules.colormappers.winds package

Submodules

geoips.plugins.modules.colormappers.winds.wind_radii_transitions module

Module containing wind speed colormap with transitions at 34, 50, 64, and 80.

geoips.plugins.modules.colormappers.winds.wind_radii_transitions.**call**(*data_range=[0, 200]*)

Generate appropriate matplotlib colors for plotting standard wind speeds.

wind_radii_transitions contains hard coded transition values for different colors, in order to have consistent imagery across all sensors / products.

Parameters

data_range (*list of float, default=[0, 200]*) –

- Min and max value for colormap.

- Ensure the data range matches the range of the algorithm specified for use with this colormap
- This colormap MUST include 0 and 200

Returns
mpl_colors_info – Dictionary of matplotlib plotting parameters, to ensure consistent image output

Return type
dict

Module contents

Geoips surface winds colormap init file.

Submodules

geoips.plugins.modules.colormappers.cmap_rgb module

Module containing matplotlib information for RGB or RGBA imagery.

`geoips.plugins.modules.colormappers.cmap_rgb.call()`

For rgb imagery, we require no color information.

colormap is entirely specified by the RGB(A) arrays, so no specific matplotlib color information required.

Parameters
arguments (*No*) –

Returns

mpl_colors_info –

- Specifies matplotlib Colors parameters for use in both plotting and colorbar generation
- For RGBA arrays, all fields are "None"

Return type
dict

geoips.plugins.modules.colormappers.matplotlib_linear_norm module

Matplotlib information for standard imagery with an existing system colormap.

geoips.plugins.modules.colormappers.matplotlib_linear_norm.**call**(*data_range=None*, *cmap_name='Greys'*, *cmap_source='matplotlib'*, *cmap_path=None*, *create_colorbar=True*, *cbar_label=None*, *cbar_ticks=None*, *cbar_tick_labels=None*, *cbar_spacing='proportional'*, *cbar_full_width=False*, *colorbar_kwargs=None*, *set_ticks_kwargs=None*, *set_label_kwargs=None*)

Set the matplotlib colors information for matplotlib linear norm cmaps.

This information used in both colorbar and image production throughout GeoIPS image output specifications.

Parameters

- **data_range** (*list, default=None*) –
  - [min_val, max_val], matplotlib.colors.Normalize(vmin=min_val, vmax=max_val)
  - If data_range not specified, vmin and vmax both None.
- **cmap_name** (*str, default="Greys"*) –
  - Specify the name of the resulting matplotlib colormap.
  - If no ascii_path specified, will use builtin matplotlib colormap of name cmap_name.
- **ascii_path** (*str, default=None*) –
  - Specify full path to ASCII palette to convert to matplotlib colormap.
  - If not specified, use internal matplotlib colormap "cmap_name".
- **cbar_label** (*str, default=None*) –
  - Positional parameter passed to cbar.set_label
  - If specified, use cbar_label string as colorbar label.

- **create_colorbar** (*bool, default=True*) –
  - Specify whether the image should contain a colorbar or not.
- **cbar_ticks** (*list, default=None*) –
  - Positional parameter passed to cbar.set_ticks
  - Specify explicit list of ticks to include for colorbar.
  - None indicates ticks at int(min) and int(max) values
- **cbar_tick_labels** (*list, default=None*) –
  - "labels" argument to pass to cbar.set_ticks.
  - can also specify directly within "set_ticks_kwargs"
- **cbar_spacing** (*string, default="proportional"*) –
  - "spacing" argument to pass to fig.colorbar
  - can also specify directly within "colorbar_kwargs"
- **cbar_full_width** (*bool, default=True*) –
  - Extend the colorbar across the full width of the image.
- **colorbar_kwargs** (*dict, default=None*) –
  - keyword arguments to pass through directly to "fig.colorbar"
- **set_ticks_kwargs** (*dict, default=None*) –
  - keyword arguments to pass through directly to "cbar.set_ticks"
- **set_label_kwargs** (*dict, default=None*) –
  - keyword arguments to pass through directly to "cbar.set_label"

Returns

mpl_colors_info –

- Specifies matplotlib Colors parameters for use in both plotting and colorbar generation

Return type
dict

See also:

API Reference
See geoips.image_utils.mpl_utils.create_colorbar for field descriptions.

Module contents

Geoips colormappers init file.

geoips.plugins.modules.coverage_checkers package

Submodules

geoips.plugins.modules.coverage_checkers.center_radius module

Coverage check routine for center radius coverage checks.

geoips.plugins.modules.coverage_checkers.center_radius.**call**(*xarray_obj*, *variable_name*, *area_def=None*, *radius_km=300*)

Coverage check routine for xarray objects with masked projected arrays.

Parameters

- **xarray_obj** (*xarray.Dataset*) – xarray object containing variable "variable_name"
- **variable_name** (*str*) – variable name to check percent unmasked
- **radius_km** (*float*) – Radius of center disk to check for coverage

Returns
Percent coverage of variable_name

Return type
float

geoips.plugins.modules.coverage_checkers.center_radius.**create_radius**(*temp_arr*, *radius_pixels=300*, *x_center=0*, *y_center=0*)

Create a radius around given x,y coordinates in the 2d array.

Given the radius and the x,y coordinates it creates a circle around those points using the skimage.draw library

Parameters

- **temp_arr** (*int*) – The 2D array.

- **radius** (*int, optional*) – The radius of the circle. 500 is default value.
- **x** (*int, optional*) – The x coordinate of middle circle point. 0 is default value.
- **y** (*int, optional*) – The x coordinate of middle circle point. 0 is default value.

Returns
2D array with circle created at the x,y coordinate with the given radius All circles are marked as 1.

Return type
numpy.ndarray

geoips.plugins.modules.coverage_checkers.center_radius.**plot_coverage**(*main_ax*, *area_def*, *covg_args*)

Plot the coverage specified by the 'center_radius' function.

Parameters

- **main_ax** (*matplotlib.axis*) – Axis on which to plot coverage representation
- **area_def** (*pyresample.AreaDefinition*) – area def for current plot
- **covg_args** (*dict*) – product params dictionary for current product - to ensure we plot the correct coverage params

Return type
No return value

geoips.plugins.modules.coverage_checkers.center_radius_rgba module

Coverage check routine for RGBA center radius coverage checks.

geoips.plugins.modules.coverage_checkers.center_radius_rgba.**call**(*xarray_obj*, *variable_name*, *area_def=None*, *radius_km=300*)

Coverage check routine for xarray objects with masked projected arrays.

Only calculates coverage within a "radius_km" radius of center.

Parameters

- **xarray_obj** (*xarray.Dataset*) – xarray object containing variable "variable_name"
- **variable_name** (*str*) – variable name to check percent unmasked radius_km (float) : Radius of center disk to check for coverage

Returns
Percent coverage of variable_name

Return type
float

geoips.plugins.modules.coverage_checkers.masked_arrays module

Coverage check routine for masked arrays.

geoips.plugins.modules.coverage_checkers.masked_arrays.**call**(*xarray_obj*, *variable_name*, *area_def=None*)

Coverage check routine for xarray objects with masked projected arrays.

Parameters

- **xarray_obj** (*xarray.Dataset*) – xarray object containing variable "variable_name"
- **variable_name** (*str*) – variable name to check percent unmasked

Returns
Percent coverage of variable_name

Return type
float

geoips.plugins.modules.coverage_checkers.numpy_arrays_nan module

Coverage check routine for masked arrays.

geoips.plugins.modules.coverage_checkers.numpy_arrays_nan.**call**(*xarray_obj*, *variable_name*, *area_def=None*)

Coverage check routine for xarray objects with projected numpy arrays.

Parameters

- **xarray_obj** (*xarray.Dataset*) – xarray object containing variable "variable_name"

- **variable_name** (*str*) – variable name to check percent unmasked

Returns
Percent coverage of variable_name

Return type
float

geoips.plugins.modules.coverage_checkers.rgba module

Coverage check routine for RGBA arrays.

geoips.plugins.modules.coverage_checkers.rgba.**call**(*xarray_obj*, *variable_name*, *area_def=None*)

Coverage check routine for xarray objects with projected RGBA arrays.

Parameters

- **xarray_obj** (*xarray.Dataset*) – xarray object containing variable "variable_name"
- **variable_name** (*str*) – variable name to check percent unmasked

Returns
Percent coverage of variable_name

Return type
float

geoips.plugins.modules.coverage_checkers.windbarbs module

Coverage check routine for windbarb xarrays.

geoips.plugins.modules.coverage_checkers.windbarbs.**call**(*xarray_obj*, *variable_name*, *area_def=None*)

Coverage check routine for wind barb xarray object.

Parameters

- **xarray_obj** (*xarray.Dataset*) – xarray object containing variable "variable_name"
- **variable_name** (*str*) – variable name to check percent unmasked.

Returns
Percent coverage of variable_name over area_def

Return type
float

Module contents

geoips coverage_checkers init file.

geoips.plugins.modules.filename_formatters package

Subpackages

geoips.plugins.modules.filename_formatters.utils package

Submodules

geoips.plugins.modules.filename_formatters.utils.tc_file_naming module

Utilities for TC filenaming, for use within geoips filename formatters.

geoips.plugins.modules.filename_formatters.utils.tc_file_naming.**get_storm_subdir**(*basin_*
base_t
tc_stor
out-
put_di
sec-
tor_inf

Get the TC storm subdirectory.

geoips.plugins.modules.filename_formatters.utils.tc_file_naming.**tc_storm_basedir**(*basedi*
tc_yea
tc_bas
tc_stor
out-
put_di
sec-
tor_inf

Produce base storm directory for TC web output.

Parameters

- **basedir** (*str*) – base directory
- **tc_year** (*int*) – Full 4 digit storm year

- **tc_basin** (*str*) –

  2 character basin designation
  SH Southern Hemisphere WP West Pacific EP East Pacific CP Central Pacific IO Indian Ocean AL Atlantic

- **tc_stormnum** (*int*) –

  2 digit storm number
  90 through 99 for invests 01 through 69 for named storms

Returns
path – Path to base storm web directory

Return type
str

geoips.plugins.modules.filename_formatters.utils.tc_file_naming.**update_extra_field**(*output_xarray_area_product_extra_existing_extra_extra_extra_include*

Finalize extra field using standard geoips arguments.

Module contents

geoips filename formatters utils package init file.

Submodules

geoips.plugins.modules.filename_formatters.basic_fname module

Filename specification using minimal basic attributes, and no subdirs.

geoips.plugins.modules.filename_formatters.basic_fname.**call**(*xarray_obj*, *area_def*, *product_name*, *output_type='.png'*, *basedir='/users/surratt/geoips/out*
extra_field=None)

Create basic filename, in a specific directory (with no subdirectories).

This filename format includes only the start time, platform name, source name, product name, sector name, and data provider, and a full path to basedir with no additional subdirectories.

geoips.plugins.modules.filename_formatters.geoips_fname module

Standard geoips filename production.

geoips.plugins.modules.filename_formatters.geoips_fname.**assemble_geoips_fname**(*basedir*, *product_name*, *source_na* *platform_name* *sector_name*, *coverage*, *resolution*, *product_dateti* *output_type=* *data_provi* *extra=None*, *product_dir=N* *source_dir* *continent=None* *country=None*, *area=None* *subarea=None* *state=None* *city=None*

Produce full output product path from product / sensor specifications.

standard web paths are of the format:

<basedir>/<continent>-<country>-<area>/<subarea>-<state>-<city>/ <productname>/<sensorname>``

standard filenames are of the format:

<date{%Y%m%d}>.<time{%H%M%S}>.<satname>.<sensorname>.<productname>.<sectorname>.<coverage>.<dataprovider>.<extra>

Parameters

- **basedir** (*str*) – Full path to base directory of final product.
- **product_name** (*str*) – Name of product
- **source_name** (*str*) – Name of data source (sensor)
- **platform_name** (*str*) – Name of platform (satellite)
- **coverage** (*float*) – Image coverage, float between 0.0 and 100.0
- **resolution** (*float*) – Image resolution, float greater than 0.0
- **product_datetime** (*datetime.datetime*) – Datetime object - start time of data used to generate product.
- **output_type** (*str, optional*) – file extension type, default is png
- **data_provider** (*str, optional*) – String to include in filename "data_provider" field
- **extra** (*str, optional*) – String to include in filename "extra" field, default is None If None, use fillval of 'x'
- **continent** (*str, optional*) – String to include in filename "continent" field, default is None If None, use fillval of 'x'
- **country** (*str, optional*) – String to include in filename "country" field, default is None If None, use fillval of 'x'
- **area** (*str, optional*) – String to include in filename "area" field, default is None If None, use fillval of 'x'
- **subarea** (*str, optional*) – String to include in filename "subarea" field, default is None If None, use fillval of 'x'
- **state** (*str, optional*) – String to include in filename "state" field, default is None If None, use fillval of 'x'
- **city** (*str, optional*) – String to include in filename "city" field, default is None If None, use fillval of 'x'

geoips.plugins.modules.filename_formatters.geoips_fname.**call**(*area_def*, *xarray_obj*, *product_name*, *coverage=None*, *output_type='png'*, *output_type_dir=None*, *product_dir=None*, *product_subdir=None*, *source_dir=None*, *basedir='/users/surratt/geoips/ou*

Create GeoIPS standard filenames, sector-based subdirs.

This uses the sector specification (continent, country, area, subarea, state, city), product name, source name, and platform name to generate a full unique path, as well as additional attributes to create a fully unique file name.

geoips.plugins.modules.filename_formatters.geoips_fname.**geoips_fname_remove_duplicates**

remove_duplicates function currently not defined.

If defined, this function will identify duplicate files, and remove appropriately. It should identify duplicates specifically based on the format/location of the geoips_fname formatted files.

Currently returns empty lists. When defined, will return a list of deleted files and a list of saved files.

geoips.plugins.modules.filename_formatters.geoips_netcdf_fname module

Standard GeoIPS NetCDF filename production.

geoips.plugins.modules.filename_formatters.geoips_netcdf_fname.**assemble_geoips_netcdf_**

Produce full output product path from product / sensor specifications.

netcdf paths are of the format:

<basedir>/<product_name>/<source_name>/<platform_name>/
<sector_name>/date{%Y%m%d}

netcdf filenames are of the format:

<date{%Y%m%d>.<time{%H%M%S}>.<platform_name>.<product_name>.
<sector_name>.nc

Parameters

- **basedir** (*str*) – Base directory (additional subdirectories assembled below basedir)
- **product_name** (*str*) – Name of product, used in path and filename
- **source_name** (*str*) – Name of data source (sensor), used in path and filename
- **platform_name** (*str*) – Name of platform (satellite), used in path and filename
- **coverage** (*float*) – Image coverage, float between 0.0 and 100.0, used in filename
- **product_datetime** (*datetime*) – Datetime object - start time of data used to generate product, used in filename

geoips.plugins.modules.filename_formatters.geoips_netcdf_fname.**call**(*area_def*, *xarray_obj*, *product_names*, *coverage=None*, *output_type='nc'*, *output_type_dir=None*, *product_dir=None*, *product_subdir=None*, *source_dir=None*, *basedir=None*)

Filename formatting for standard GeoIPS-style NetCDF outputs.

This uses the "assemble_geoips_netcdf_fname" function to appropriately assemble the filename from a base directory, product name, source name, platform nae, sector name, and product time, to allow reuse of this basic filename format from multiple filename formatter plugins.

geoips.plugins.modules.filename_formatters.geotiff_fname module

Standard GeoIPS GEOTIFF filename formatter.

geoips.plugins.modules.filename_formatters.geotiff_fname.**call**(*area_def*, *xarray_obj*, *product_name*, *coverage=None*, *output_type='tif'*, *output_type_dir=None*, *product_dir=None*, *product_subdir=None*, *source_dir=None*, *basedir='/users/surratt/geoips/*
output_dict=None)

GEOTIFF filename formatter.

This uses the standard "geoips_fname" formatter, but with a default output type of "tif".

geoips.plugins.modules.filename_formatters.metadata_default_fname module

Default TC metadata filename formatter.

geoips.plugins.modules.filename_formatters.metadata_default_fname.**call**(*area_def*, *xarray_obj*, *product_filename*, *metadata_dir='metadata*, *metadata_type='sector_*, *basedir='/users/sur*, *output_dict=None*)

Generate TC metadata filenames.

This uses attributes on both the xarray and area_def in order to produce the YAML metadata output specifically for TC sectors. Not valid for other sector types.

This uses the "tc_storm_basedir" utility to ensure a consistent path to the storm directory (so products and metadata end up in the same location)

geoips.plugins.modules.filename_formatters.tc_clean_fname module

Clean TC filename production (no backgrounds or overlays).

geoips.plugins.modules.filename_formatters.tc_clean_fname.**call**(*area_def*, *xarray_obj*, *product_name*, *coverage=None*, *output_type='png'*, *output_type_dir=None*, *product_dir=None*, *product_subdir=None*, *source_dir=None*, *basedir='/users/surratt/geoips*, *output_dict=None*)

Clean TC product filename formatter (no gridlines, titles, etc).

This ensures output ends up in "png_clean" directory, with "-clean" appended to the extra field, to avoid conflict with tc_fname based annotated imagery. Uses "tc_fname" module as a base.

Parameters

- **area_def** (*pyresample AreaDefinition*) – Contains metadata regarding sector
- **xarray_obj** (*xarray Dataset*) – Contains metadata regarding dataset
- **product_name** (*str*) – String product_name specification for use in filename
- **coverage** (*float*) – Percent coverage, for use in filename
- **output_type** (*str, optional*) – Requested output format, ie png, jpg, tif, etc, defaults to None.
- **output_type_dir** (*str, optional*) – Directory name for given output type (ie png_clean, png, etc), defaults to None.
- **product_dir** (*str, optional*) – Directory name for given product, defaults to None.
- **product_subdir** (*str, optional*) – Subdir name for given product, if any, defaults to None.
- **source_dir** (*str, optional*) – Directory name for given source, defaults to None.

- **basedir** (*str, optional*) – Base directory, defaults to $TCWWW.

Returns
Full path to output "clean" filename - with "-clean" appended to extra field, and "_clean" appended to output_type_dir.

Return type
str

geoips.plugins.modules.filename_formatters.tc_fname module

Standard TC filename formatter.

geoips.plugins.modules.filename_formatters.tc_fname.**assemble_tc_fname**(*basedir*, *tc_year*, *tc_basin*, *tc_stormnum*, *output_type*, *product_name*, *source_name*, *platform_name*, *coverage*, *product_datetime*, *intensity=None*, *extra=None*, *output_type_dir=None*, *product_dir=None*, *product_subdir=None*, *output_dict=None*, *sector_info=None*)

Produce full output product path from product / sensor specifications.

tc web paths are of the format:

<basedir>/tc<tc_year>/<tc_basin>/<tc_basin><tc_stormnum><tc_year>/
<output_type>/<product_name>/<platform_name>/

tc web filenames are of the format:

<date{%Y%m%d%H%M>_<tc_basin><tc_stormnum><tc_year>_<source_name>_
<platform_name>_<product_name>_<intensity>_<coverage>_ <extra>.<output_type>

Parameters

- **basedir** (*str*) – Base directory for output file.
- **tc_year** (*int*) – Full 4 digit storm year
- **tc_basin** (*str*) –

  2 character basin designation
  SH Southern Hemisphere WP West Pacific EP East Pacific CP Central Pacific IO Indian Ocean AL Atlantic

- **tc_stormnum** (*int*) –

  2 digit storm number
  90 through 99 for invests 01 through 69 for named storms

- **output_type** (*str*) – file extension type
- **product_name** (*str*) – Name of product
- **source_name** (*str*) – Name of data source (sensor)
- **platform_name** (*str*) – Name of platform (satellite)
- **coverage** (*float*) – Image coverage, float between 0.0 and 100.0
- **product_datetime** (*datetime.datetime*) – Datetime object - start time of data used to generate product
- **output_type_dir** (*str*) – Default output_type, dir name
- **product_dir** (*str*) – Default product_name, dir name

Returns
full path to output file

Return type
str

geoips.plugins.modules.filename_formatters.tc_fname.**call**(*area_def*, *xarray_obj*, *product_name*, *coverage=None*, *output_type='png'*, *output_type_dir=None*, *product_dir=None*, *product_subdir=None*, *source_dir=None*, *basedir='/users/surratt/geoips/outdirs/*, *extra_field=None*, *output_dict=None*)

Create standard TC filenames.

See also:

geoips.plugins.modules.filename_formatters.tc_fname.assemble_tc_fname
This uses the shared utility "assemble_tc_fname", such that a common filename can be used by related filename formatters.

geoips.plugins.modules.filename_formatters.tc_fname.**tc_fname_remove_duplicates**(*fname*, *mins_to_remove_file*

Remove tc_fname duplicate files.

Matches storm name, sensor name, platform name, product name, and resolution for all files within "mins_to_remove" minutes of the current file. All other fields are wild carded during the matching process.

geoips.plugins.modules.filename_formatters.text_winds_day_fname module

Filename formatter for full-day text windspeed products.

geoips.plugins.modules.filename_formatters.text_winds_day_fname.**call**(*xarray_obj*, *extension='.txt'*, *basedir='/users/surrat*

Create full-day text windspeed filenames.

text_winds_day_fname includes only YYYYMMDD in the filename, so all data for a full day is appended into a single file.

See also:

geoips.plugins.modules.filename_formatters.text_winds_full_fname. assemble_text_windspeeds_text_full_fname Shared utility for generating similarly formatted windspeed filenames.

geoips.plugins.modules.filename_formatters.text_winds_full_fname module

Filename formatter for text windspeed products.

geoips.plugins.modules.filename_formatters.text_winds_full_fname.**assemble_windspeeds_t**

Produce full output product path using product / sensor specifications.

Parameters

- **basedir** (*str*) – base directory
- **source_name** (*str*) – Name of source (sensor)
- **platform_name** (*str*) – Name of platform (satellite)
- **data_provider** (*str*) – Name of data provider
- **product_datetime** (*datetime.datetime*) – Start time of data used to generate product
- **dt_format** (*str, default="%Y%m%d.%H%M"*) – Format used to display product_datetime within filename
- **extension** (*str, default=".txt"*) – File extension, specifying type.
- **creation_time** (*datetime.datetime, default=None*) – Include given creation_time of file in filename If None, do not include creation time.

Returns

full path of output filename of the format:

<basedir>/<source_name>_<data_provider>_<platform_name>_ surface_winds_<YYYYMMDDHHMN>

Return type
str

Examples

```
>>> startdt = datetime.strptime('20200216T001412', '%Y%m%dT%H%M%S')
>>> assemble_windspeeds_text_full_fname(
...     '/outdir',
...     'smap-spd',
...     'smap',
...     'remss',
...     startdt,
...     '%Y%m%d'
...     )
'/outdir/smap-spd_remss_smap_surface_winds_20200216'
```

geoips.plugins.modules.filename_formatters.text_winds_full_fname.**call**(*xarray_obj*, *extension='.txt'*, *basedir='/users/surra*

Create a single text winds file for all data in the current xarray.

This text windspeed filename includes YYYYMMDD.HHMN in the filename in order to include only the current datafile in the file.

See also:

geoips.plugins.modules.filename_formatters.text_winds_full_fname.
assemble_windspeeds_text_full_fname Shared utility to create filenames with similar formatting requirements.

geoips.plugins.modules.filename_formatters.text_winds_tc_fname module

Filename formatter for TC-specific text windspeed outputs.

geoips.plugins.modules.filename_formatters.text_winds_tc_fname.**assemble_windspeeds_tex**

Produce full output product path from product / sensor specifications.

Parameters

- **basedir** (*str*) – base directory
- **tc_year** (*int*) – Full 4 digit storm year
- **tc_basin** (*str*) –

  2 character basin designation
  SH Southern Hemisphere WP West Pacific EP East Pacific CP Central Pacific IO Indian Ocean AL Atlantic

- **tc_stormnum** (*int*) –

  2 digit storm number
  90 through 99 for invests 01 through 69 for named storms

- **platform_name** (*str*) – Name of platform (satellite)
- **product_datetime** (*datetime*) – Start time of data used to generate product

Returns

full path of output filename of the format:

<basedir>/tc<tc_year>/<tc_basin>/<tc_basin><tc_stormnum><tc_year>/
txt/<source_name>_<platform_name>_surface_winds_<data_provider>_
<YYYYMMDDHHMN>

Return type
str

Examples

```
>>> startdt = datetime.strptime('20200216T001412', '%Y%m%dT%H%M%S')
>>> assemble_windspeeds_text_tc_fname('/outdir',
...     2020,
...     'SH',
...     16,
...     'smap-spd',
...     'smap',
...     startdt,
...     'remss')
'/outdir/tc2020/SH/SH162020/txt/
```

geoips.plugins.modules.filename_formatters.text_winds_tc_fname.**call**(*xarray_obj*, *extension='.txt'*, *basedir='/users/surratt/*, *output_dict=None*)

Create TC-specific text windspeed filename.

See also:

geoips.plugins.modules.filename_formatters.text_winds_tc_fname.
assemble_windspeeds_text_tc_fname Shared utility to facilitate creating multiple similar filenames from the same code.

Module contents

geoips filename_formatters init file.

geoips.plugins.modules.interpolators package

Subpackages

geoips.plugins.modules.interpolators.pyresample_wrappers package

Submodules

geoips.plugins.modules.interpolators.pyresample_wrappers.interp_gauss module

Geoips plugin for driving pyresample Gaussian interpolation.

geoips.plugins.modules.interpolators.pyresample_wrappers.interp_gauss.**call**(*area_def*, *input_xarray*, *output_xarray*, *varlist*, *array_num=Non*, *sigmaval=None*, *drop_nan=Fal*

Pyresample interp_kd_tree gaussian interpolation GeoIPS plugin.

geoips.plugins.modules.interpolators.pyresample_wrappers.interp_nearest module

Geoips plugin for driving pyresample Nearest Neighbor interpolation.

geoips.plugins.modules.interpolators.pyresample_wrappers.interp_nearest.**call**(*area_def*, *input_xarray*, *output_xarray*, *varlist*, *array_num=N*

Pyresample interp_kd_tree nearest neighbor GeoIPS plugin.

geoips.plugins.modules.interpolators.pyresample_wrappers.interp_nearest.**get_final_roi**(

Get the final interpolation Radius of Influence.

This takes the maximum of the xarray attribute, area_def pixel width, and area_def pixel height.

Module contents

Geoips pyresample interpolators init file.

geoips.plugins.modules.interpolators.scipy_wrappers package

Submodules

geoips.plugins.modules.interpolators.scipy_wrappers.interp_grid module

Geoips plugin for driving scipy griddata interpolation.

geoips.plugins.modules.interpolators.scipy_wrappers.interp_grid.**call**(*area_def*, *input_xarray*, *output_xarray*, *varlist*, *array_num=None*, *method=None*)

Scipy griddata interpolation GeoIPS plugin.

Module contents

Geoips scipy-based interpolators init file.

geoips.plugins.modules.interpolators.utils package

Submodules

geoips.plugins.modules.interpolators.utils.boxdefinitions module

Classes for geometry operations allowing masked data in swath corners.

This package uses the pyresample geometry package as base classes:

pyresample, Resampling of remote sensing image data in python # # Copyright (C) 2010-2015 # # Authors: # Esben S. Nielsen # Thomas Lavergne

class geoips.plugins.modules.interpolators.utils.boxdefinitions.**Line**(*start*, *end*)

Bases: object

A Line between two lat/lon points.

end = None

intersection(*other*)

Identify intersection between two lines.

Says where, if two lines defined by the current line and the *other_line* intersect.

intersects(*other_line*)

Test two lines intersect.

Says if two lines defined by the current line and the *other_line* intersect. A line is defined as the shortest tracks between two points.

start = None

class geoips.plugins.modules.interpolators.utils.boxdefinitions.**MaskedCornersSwathDefi**

Bases: SwathDefinition

Swath defined by lons and lats.

Allows datasets with potentially masked data in the corners.

Parameters

- **lons** (*numpy array*) – Longitude values
- **lats** (*numpy array*) – Latitude values
- **nprocs** (*int, optional*) – Number of processor cores to be used for calculations.

shape

Swath shape

Type
tuple

size

Number of elements in swath

Type
int

ndims

Swath dimensions

Type
int

Properties

---

lons

Swath lons

Type
object

lats

Swath lats

Type
object

cartesian_coords

Swath cartesian coordinates

Type
object

property corners

Return current area corners.

get_bounding_box_lonlats(*npts=100*)

Return lon/lats along bounding Arcs.

Parameters
npts (*int*) – Number of points to return along each line

Returns

- **(top, right, bottom, left)** (*4 tuples containing lists*) – of len npts of lons/lats
- *retval = (list(tplons),list(tplats)),* – (list(rtlons),list(rtlats)), (list(btlons),list(btlats)), (list(ltlons),list(ltlats))
- *eg for n=3* – ([tplon0,tplon1,tplon2],[tplat0,tplat1,tplat2]), ([rtlon0,rtlon1,rtlon2],[rtlat0,rtlat1,rtlat2]), ([btlon0,btlon1,btlon2],[btlat0,btlat1,btlat2]), ([ltlon0,ltlon1,ltlon2],[ltlat0,ltlat1,ltlat2]),

intersection(*other*)

Return current area intersection polygon corners.

other allows for potentially masked data in the corners.

Parameters
other (*object*) – Instance of subclass of BaseDefinition

Returns
(corner1, corner2, corner3, corner4)

Return type
tuple of points

overlaps_minmaxlatlon(*other*)

Test current area overlaps *other* area.

This is based solely on the min/max lat/lon of areas, assuming the
boundaries to be along lat/lon lines.

Parameters
other (*object*) – Instance of subclass of BaseDefinition

Returns
overlaps

Return type
bool

class geoips.plugins.modules.interpolators.utils.boxdefinitions.**PlanarPolygonDefinitio**

Bases: CoordinateDefinition

Planar polygon definition.

property corners

Return corners.

get_bounding_box_lonlats(*npts=100*)

Return array of lon/lats along the bounding lat/lon lines.

Parameters
npts (*int*) – Number of points to return along each line

Returns

- **(top, right, bottom, left)** (*4 tuples containing lists*) – of len npts of lons/lats

- *retval = (list(tplons),list(tplats)),* – (list(rtlons),list(rtlats)), (list(btlons),list(btlats)), (list(ltlons),list(ltlats))
- *eg for n=3* – ([tplon0,tplon1,tplon2],[tplat0,tplat1,tplat2]), ([rtlon0,rtlon1,rtlon2],[rtlat0,rtlat1,rtlat2]), ([btlon0,btlon1,btlon2],[btlat0,btlat1,btlat2]), ([ltlon0,ltlon1,ltlon2],[ltlat0,ltlat1,ltlat2]),

intersection(*other*)

Return current area intersection polygon corners against other.

Parameters
other (*object*) – Instance of subclass of BaseDefinition

Returns
(corner1, corner2, corner3, corner4)

Return type
tuple of points

overlaps(*other*)

Test if the current area overlaps the *other* area.

This is based solely on the corners of areas, assuming the boundaries to be straight lines.

Parameters
other (*object*) – Instance of subclass of BaseDefinition

Returns
overlaps

Return type
bool

overlaps_minmaxlatlon(*other*)

Determine if overlaps.

geoips.plugins.modules.interpolators.utils.boxdefinitions.**get_2d_false_corners**(*box_def*)

Identify false corners.

geoips.plugins.modules.interpolators.utils.boxdefinitions.**planar_intersection_polygon**(

Get the intersection polygon between two areas.

geoips.plugins.modules.interpolators.utils.boxdefinitions.**planar_point_inside**(*point*, *corners*)

Identify point inside 4 corners.

This DOES NOT USE great circle arcs as area boundaries.

geoips.plugins.modules.interpolators.utils.interp_pyresample module

Interpolation methods using pyresample routines.

geoips.plugins.modules.interpolators.utils.interp_pyresample.**get_data_box_definition**(*s*
l
l

Obtain pyresample geometry definitions.

For use with pyresample based reprojections

Parameters

- **source_name** (*str*) – geoips source_name for data type
- **lons** (*ndarray*) – Numpy array of longitudes, 0 to 360
- **lats** (*ndarray*) – Numpy array of latitudes, -90 to 90

geoips.plugins.modules.interpolators.utils.interp_pyresample.**interp_kd_tree**(*list_of_array*
area_definiti
data_box_de
ra-
dius_of_influ
in-
terp_type='n
sig-
mas=None,
neigh-
bours=None,
nprocs=None
fill_value=N

Interpolate using pyresample's kd_tree.resample_nearest method.

Parameters

- **list_of_array** (*list of numpy.ndarray*) – list of arrays to be interpolated
- **area_definition** (*areadef*) – pyresample area_definition object of current region of interest.
- **data_box_definition** (*datadef*) – pyresample/geoips data_box_definition specifying region covered by source data.
- **radius_of_influence** (*float*) – radius of influence for interpolation
- **interp_type** (*str, default='nearest'*) – One of 'nearest' or 'gauss' - kd_tree resampling methods.
- **sigmas** (*int, default=None*) –

Used for interp_type 'gauss' - multiplication factor for sigmas option:

- sigmas = [sigmas]*len(list_of_arrays)

geoips.plugins.modules.interpolators.utils.interp_scipy module

Interpolation routines from the scipy package.

geoips.plugins.modules.interpolators.utils.interp_scipy.**interp_gaussian_kde**(*data_lons*, *data_lats*, *target_lons*, *target_lats*, *vw_method=*

Interpolate a given array of non-uniform data using scipy.stats.gaussian_kde.

This is not finalized.

Parameters

- **data_array** (*numpy.ma.core.MaskedArray*) – numpy array of data to interpolate
- **data_lons** (*numpy.ma.core.MaskedArray*) – numpy array of longitudes corresponding to original data, same shape as data_array
- **data_lats** (*numpy.ma.core.MaskedArray*) – numpy array of latitudes corresponding to original data, same shape as data_array
- **target_lons** (*numpy.ma.core.MaskedArray*) – 2d numpy array of desired longitudes
- **target_lats** (*numpy.ma.core.MaskedArray*) – 2d numpy array of desired latitudes
- **bw_method** (*str*) – Bandwidth selection method (see scipy.stats.gaussian_kde)|

See also:

scipy.stats.gaussian_kde

geoips.plugins.modules.interpolators.utils.interp_scipy.**interp_griddata**(*data_array*, *data_lons*, *data_lats*, *min_gridlon*, *max_gridlon*, *min_gridlat*, *max_gridlat*, *numx_grid*, *numy_grid*, *method='linear'*)

Interpolate a given array of non-uniform data to a specified grid.

Uses scipy.interpolate.griddata

Parameters

- **data_array** (*numpy.ma.core.MaskedArray*) – numpy array of original data to be interpolated
- **data_lons** (*numpy.ma.core.MaskedArray*) – numpy array of longitudes corresponding to original data, same shape as data_array
- **data_lats** (*numpy.ma.core.MaskedArray*) – numpy array of latitudes corresponding to original data, same shape as data_array
- **min_gridlon** (*float*) –

  minimum desired lon for the output grid

  - -180.0 < min_gridlon < 180.0
- **max_gridlon** (*float*) –

  maximum desired lon for the output grid

  - -180.0 < max_gridlon < 180.0
- **min_gridlat** (*float*) –

  minimum desired lat for the output grid

  - -90.0 < min_gridlat < 90.0
- **max_gridlat** (*float*) –

  maximum desired lat for the output grid

  - -90.0 < max_gridlat < 90.0
- **numx_grid** (*int*) – number desired longitude points in the output grid
- **numy_grid** (*int*) – number desired latitude points in the output grid

- **method** (*str, default='linear'*) – A string specifying the interpolation method to use for scipy.interpolate.griddata. One of 'nearest', 'linear' or 'cubic'

Module contents

Geoips interpolators utils init file.

Module contents

Geoips interpolators init file.

geoips.plugins.modules.output_formatters package

Submodules

geoips.plugins.modules.output_formatters.full_disk_image module

Full disk image matplotlib-based output format.

geoips.plugins.modules.output_formatters.full_disk_image.**call**(*area_def*, *xarray_obj*, *product_name*, *output_fnames*, *clean_fname=None*, *product_name_title=None*, *mpl_colors_info=None*, *feature_annotator=None*, *gridline_annotator=None*, *product_datatype_title=None*, *bg_data=None*, *bg_mpl_colors_info=None*, *bg_xarray=None*, *bg_product_name_title=None*, *bg_datatype_title=None*, *remove_duplicate_minrange=Nor*

Plot full disk image.

geoips.plugins.modules.output_formatters.geotiff_standard module

Geotiff image rasterio-based output format.

geoips.plugins.modules.output_formatters.geotiff_standard.**call**(*area_def*, *xarray_obj*, *product_name*, *output_fnames*, *product_name_title=None*, *mpl_colors_info=None*, *existing_image=None*)

Create standard geotiff output using rasterio.

geoips.plugins.modules.output_formatters.geotiff_standard.**get_rasterio_cmap_dict**(*mpl_cr*
*scale_*
*scale_*

Get rasterio cmap dict.

geoips.plugins.modules.output_formatters.geotiff_standard.**scale_geotiff_data**(*plot_data*,
*mpl_colors_*
*scale_data_*
*scale_data_*
missing_value=

Scale geotiff data.

geoips.plugins.modules.output_formatters.imagery_annotated module

Matplot-lib based annotated image output.

geoips.plugins.modules.output_formatters.imagery_annotated.**call**(*area_def*, *xarray_obj*, *product_name*, *output_fnames*, *clean_fname=None*, *product_name_title=None*, *mpl_colors_info=None*, *feature_annotator=None*, *gridline_annotator=None*, *product_datatype_title=None*, *bg_data=None*, *bg_mpl_colors_info=None*, *bg_xarray=None*, *bg_product_name_title=Non*
bg_datatype_title=None, *remove_duplicate_minrange=N*
title_copyright=None, *title_formatter=None*, *output_dict=None*)

Plot annotated imagery.

geoips.plugins.modules.output_formatters.imagery_clean module

Matplotlib-based clean image output.

geoips.plugins.modules.output_formatters.imagery_clean.**call**(*area_def*, *xarray_obj*, *product_name*, *output_fnames*, *product_name_title=None*, *mpl_colors_info=None*, *existing_image=None*, *remove_duplicate_minrange=None*, *fig=None*, *main_ax=None*, *mapobj=None*)

Plot clean image on matplotlib figure.

geoips.plugins.modules.output_formatters.imagery_windbarbs module

Matplotlib-based windbarb annotated image output.

geoips.plugins.modules.output_formatters.imagery_windbarbs.**call**(*area_def*, *xarray_obj*, *product_name*, *output_fnames*, *clean_fname=None*, *product_name_title=None*, *mpl_colors_info=None*, *feature_annotator=None*, *gridline_annotator=None*, *product_datatype_title=None*, *bg_data=None*, *bg_mpl_colors_info=None*, *bg_xarray=None*, *bg_product_name_title=Non*
bg_datatype_title=None, *remove_duplicate_minrange=N*
title_copyright=None, *title_formatter=None*)

Plot annotated windbarbs on matplotlib figure.

geoips.plugins.modules.output_formatters.imagery_windbarbs.**format_windbarb_data**(*xarray_*
product_nam

Format windbarb data before plotting.

geoips.plugins.modules.output_formatters.imagery_windbarbs.**output_clean_windbarbs**(*area_*
clean
mpl_c
image_d
formatted_d
fig=N
*main_*
mapo

Plot and save "clean" windbarb imagery.

No background imagery, coastlines, gridlines, titles, etc.

Returns
Full paths to all resulting output files.

Return type
list of str

geoips.plugins.modules.output_formatters.imagery_windbarbs.**plot_barbs**(*main_ax*, *mapobj*, *mpl_colors_info*, *formatted_data_dict*)

Plot windbarbs on matplotlib figure.

geoips.plugins.modules.output_formatters.imagery_windbarbs_clean module

Matplotlib-based windbarb clean image output (no overlays or backgrounds).

geoips.plugins.modules.output_formatters.imagery_windbarbs_clean.**call**(*area_def*, *xarray_obj*, *product_name*, *output_fnames*, *product_name_title=None*, *mpl_colors_info=No*, *existing_image=None*, *remove_duplicate_mini*, *fig=None*, *main_ax=None*, *mapobj=None*)

Plot clean windbarb imagery on matplotlib figure.

geoips.plugins.modules.output_formatters.metadata_default module

Default YAML metadata output format.

geoips.plugins.modules.output_formatters.metadata_default.**call**(*area_def*, *xarray_obj*, *metadata_yaml_filename*, *product_filename*, *metadata_dir='metadata'*, *basedir='/users/surratt/geoips*, *output_dict=None*, *include_metadata_filename=Fa*

Produce metadata yaml file of sector info associated with final_product.

Parameters

- **area_def** (*AreaDefinition*) – Pyresample AreaDefintion object
- **final_product** (*str*) – Product that is associated with the passed area_def
- **metadata_dir** (*str, default='metadata'*) – Subdirectory name for metadata (using non-default allows for non-operational outputs)

Returns

Metadata yaml filename, if one was produced.

Return type

str

geoips.plugins.modules.output_formatters.metadata_default.**output_metadata_yaml**(*metadata area_def xarray_obj*, *product_filena output_dict= include_me*

Write out yaml file "metadata_fname" of sector info found in "area_def".

Parameters

- **metadata_fname** (*str*) – Path to output metadata_fname
- **area_def** (*AreaDefinition*) – Pyresample AreaDefinition of sector information
- **xarray_obj** (*xarray.Dataset*) – xarray Dataset object that was used to produce product
- **productname** (*str*) – Full path to full product filename that this YAML file refers to

Returns
Path to metadata filename if successfully produced.

Return type
str

geoips.plugins.modules.output_formatters.metadata_default.**update_sector_info_with_defa**

Update sector info found in "area_def" with standard metadata output.

Parameters

- **area_def** (*AreaDefinition*) – Pyresample AreaDefinition of sector information
- **xarray_obj** (*xarray.Dataset*) – xarray Dataset object that was used to produce product
- **product_filename** (*str*) – Full path to full product filename that this YAML file refers to

Returns

sector_info dict with standard metadata added

- bounding box
- product filename with wildcards
- basename of original source filenames

Return type
dict

geoips.plugins.modules.output_formatters.metadata_tc module

TC product YAML metadata output format.

geoips.plugins.modules.output_formatters.metadata_tc.**call**(*area_def*, *xarray_obj*, *metadata_yaml_filename*, *product_filename*, *metadata_dir='metadata'*, *basedir='/users/surratt/geoips/outdir*, *output_dict=None*, *metadata_fname_dict=None*, *output_fname_dict=None*)

Produce metadata yaml file of sector info associated with final_product.

Parameters

- **area_def** (*AreaDefinition*) – Pyresample AreaDefintion object
- **final_product** (*str*) – Product that is associated with the passed area_def
- **metadata_dir** (*str, default="metadata"*) – Subdirectory name for metadata (using non-default allows for non-operational outputs)

Returns

Metadata yaml filename, if one was produced.

Return type

str

geoips.plugins.modules.output_formatters.metadata_tc.**output_tc_metadata_yaml**(*metadata_fr*, *area_def*, *xarray_obj*, *product_filenam*, *output_dict=No*, *metadata_fname*, *output_fname_*

Write out yaml file "metadata_fname" of sector info found in "area_def".

Parameters

- **metadata_fname** (*str*) – Path to output metadata_fname
- **area_def** (*AreaDefinition*) – Pyresample AreaDefinition of sector information
- **xarray_obj** (*xarray.Dataset*) – xarray Dataset object that was used to produce product
- **productname** (*str*) – Full path to full product filename that this YAML file refers to

Returns
Path to metadata filename if successfully produced.

Return type
str

geoips.plugins.modules.output_formatters.metadata_tc.**update_sector_info_with_coverage**(

Update sector info with coverage, for YAML metadata output.

geoips.plugins.modules.output_formatters.metadata_tc.**update_sector_info_with_data_time**

Update sector info with data times, for YAML metadata output.

geoips.plugins.modules.output_formatters.netcdf_geoips module

Geoips style NetCDF output format.

geoips.plugins.modules.output_formatters.netcdf_geoips.**call**(*xarray_obj*, *product_names*, *output_fnames*)

Write GeoIPS style NetCDF to disk.

geoips.plugins.modules.output_formatters.netcdf_xarray module

Standard xarray-based NetCDF output format.

geoips.plugins.modules.output_formatters.netcdf_xarray.**call**(*xarray_obj*, *product_names*, *output_fnames*)

Write xarray-based NetCDF outputs to disk.

geoips.plugins.modules.output_formatters.netcdf_xarray.**write_xarray_netcdf**(*xarray_obj*, *ncdf_fname*, *clobber=False*)

Write out xarray_obj to netcdf file named ncdf_fname.

geoips.plugins.modules.output_formatters.text_winds module

Routines for outputting formatted text wind speed and vector data files.

geoips.plugins.modules.output_formatters.text_winds.**call**(*xarray_dict*, *varlist*, *output_fnames*, *append=False*, *overwrite=True*, *source_names=None*)

Write text windspeed output file.

geoips.plugins.modules.output_formatters.text_winds.**write_text_winds**(*xarray_obj*, *varlist*, *output_fnames*, *append=False*, *overwrite=True*, *source_names=None*)

Write out TC formatted text file of wind speeds.

Parameters

- **text_fname** (*str*) – String full path to output filename
- **speed_array** (*ndarray*) – array of windspeeds
- **time_array** (*ndarray*) – array of POSIX time stamps same length as speed_array

- **lon_array** (*ndarray*) – array of longitudes of same length as speed_array
- **lat_array** (*ndarray*) – array of latitudes of same length as speed_array
- **platform_name** (*str*) – String platform name

geoips.plugins.modules.output_formatters.unprojected_image module

Matplotlib-based unprojected image output.

geoips.plugins.modules.output_formatters.unprojected_image.**call**(*xarray_obj*, *product_name*, *output_fnames*, *product_name_title=None*, *mpl_colors_info=None*, *x_size=None*, *y_size=None*, *savefig_kwargs=None*)

Plot unprojected image to matplotlib figure.

Module contents

Geoips output formatter init file.

geoips.plugins.modules.procflows package

Submodules

geoips.plugins.modules.procflows.config_based module

Processing workflow for config-based processing.

geoips.plugins.modules.procflows.config_based.**call**(*fnames*, *command_line_args=None*)

Workflow for efficiently running all required outputs.

Includes all sectors and products specified in a YAML output config file. Specified via a YAML config file

Parameters

- **fnames** (*list*) – List of strings specifying full paths to input file names to process
- **command_line_args** (*dict*) – dictionary of command line arguments

Returns

0 for successful completion, non-zero for error (incorrect comparison, or failed run)

Return type

int

geoips.plugins.modules.procflows.config_based.**get_area_def_list_from_dict**(*area_defs*)

Get a list of actual area_defs from full dictionary.

Dict returned from get_area_defs_from_available_sectors

geoips.plugins.modules.procflows.config_based.**get_area_defs_from_available_sectors**(*avai com man xobj vari able*

Get all required area_defs for the given set of parameters.

YAML config parameters (config_dict), command_line_args, xobjs, and required variables. Command line args override config specifications.

Parameters

- **available_sectors_dict** (*dict*) – Dictionary of all requested sector_types (specified in YAML config)
- **command_line_args** (*dict*) – Dictionary of command line arguments - any command line argument that is also a key in available_sectors_dict[<sector_type>] will replace the value in the available_sectors_dict[<sector_type>]
- **xobjs** (*dict*) – Dictionary of xarray datasets, used in determining start/end time of data files for identifying dynamic sectors
- **variables** (*list*) – List of required variables, for determining center coverage for TCs

Returns

Dictionary of required area_defs, with area_def.name as the dictionary keys. Based on YAML config-specified available_sectors, and command line args

Return type

dict

Notes

- Each area_def.name key has one or more "sector_types" associated with it.
- Each sector_type dictionary contains the actual "requested_sector_dict" from the YAML config, and the actual AreaDefinition object that was returned.
  - `area_defs[area_def.name][sector_type]['requested_sector_dict']`
  - `area_defs[area_def.name][sector_type]['area_def']`

geoips.plugins.modules.procflows.config_based.**get_bg_xarray**(*sect_xarrays*, *area_def*, *prod_plugin*, *resampled_read=False*)

Get background xarray.

geoips.plugins.modules.procflows.config_based.**get_config_dict**(*config_yaml_file*)

Populate the full config dictionary from a given YAML config file.

Includes both sector and output specifications.

Parameters

config_yaml_file (*str*) – Full path to YAML config file, containing sector and output specifications. YAML config files support environment variables in entries flagged with !ENV

Returns

Return dictionary of both sector and output specifications, as found in config_yaml_file. The output dictionary references the "sector_types" found in the available_sectors dictionary, each output_type requests a specific "sector_type" to be used for processing.

Return type

dict

geoips.plugins.modules.procflows.config_based.**get_required_outputs**(*config_dict*, *sector_type*)

Get only the required outputs from the current sector_type.

geoips.plugins.modules.procflows.config_based.**get_resampled_read**(*config_dict*, *area_defs*, *area_def_id*, *sector_type*, *reader*, *fnames*, *variables*)

Return dictionary of xarray datasets for a given area def.

Xarrays resampled to area_def

geoips.plugins.modules.procflows.config_based.**get_sectored_read**(*config_dict*, *area_defs*, *area_def_id*, *sector_type*, *reader*, *fnames*, *variables*)

Return dictionary of xarray datasets for a given area def.

Xarrays sectored to area_def

geoips.plugins.modules.procflows.config_based.**get_variables_from_available_outputs_dic**

Get required variables for all outputs for a given "source_name".

Outputs specified within the YAML config.

Parameters

- **available_outputs_dict** (*dict*) – Dictionary of all requested output_types (specified in YAML config)
- **source_name** (*str*) – Find all required variables for the passed "source_name"
- **sector_types** (*list, default=None*) – if sector_types list of strings is passed, only include output_types that require one of the passed "sector_types"

Returns

List of all required variables for all output products for the given source_name

Return type

list

geoips.plugins.modules.procflows.config_based.**initialize_final_products**(*final_products*, *cpath*, *cmodule*)

Initialize the final_products dictionary with cpath dict key if needed.

Parameters

- **final_products** (*dict*) – Dictionary of final products, with keys of final required "compare_path" Products with no compare_path specified are stored with the key "no_comparison"
- **cpath** (*str*) – Key to add to final_products dictionary

Returns

Return final_products dictionary, updated with current "cpath" key: final_products[cpath]['files'] = <list_of_files_in_given_cpath>

Return type

dict

geoips.plugins.modules.procflows.config_based.**is_required_sector_type**(*available_outputs_di*
sector_type)

Check if current sector is required for any outputs.

Check if a given sector_type is required for any currently requested output_types

Parameters

- **available_outputs_dict** (*dict*) – Dictionary of all requested output_types (specified in YAML config)
- **sector_type** (*str*) – Determine if any output_types require the currently requested "sector_type"

Returns

- True if any output_types require the passed "sector_type"
- False if no output_types require the passed "sector_type"

Return type

bool

geoips.plugins.modules.procflows.config_based.**process_unsectored_data_outputs**(*final_prod*
available_outpu
available_secto
xobjs,
variables,
command_line
write_to_p

Process unsectored data output.

Loop through all possible outputs, identifying output types that require unsectored data output. Produce all required unsectored data output, update final_products dictionary accordingly, and return final_products dictionary with the new unsectored outputs.

Parameters

- **final_products** (*dict*) – Dictionary of final products, with keys of final required "compare_path" Products with no compare_path specified are stored with the key "no_comparison"
- **available_outputs_dict** (*dict*) – Dictionary of all available output product specifications
- **available_sectors_dict** (*dict*) – Dictionary of available sector types - we are looking for available sectors that contain the "unsectored" keyword.
- **xobjs** (*dict*) – Dictionary of xarray datasets, for use in producing unsectored output formats
- **variables** (*list*) – List of strings of required variables in the given product.

Returns

Return final_products dictionary, updated with current "cpath" key: final_products[cpath]['files'] = <list_of_files_in_given_cpath>

Return type

dict

geoips.plugins.modules.procflows.config_based.**requires_bg**(*available_outputs_dict*, *sector_type*)

Check if current sector requires background imagery.

Check if a given sector_type is requested for any product_types that also require background imagery.

Parameters

- **available_outputs_dict** (*dict*) – Dictionary of all requested output_types (specified in YAML config)
- **sector_type** (*str*) – sector_type to determine if any output_types that require background imagery also request the passed sector_type

Returns

- True if any output_types that require background imagery require the passed "sector_type"
- False if no output_types require both background imagery and the passed "sector_type"

Return type
bool

geoips.plugins.modules.procflows.config_based.**set_comparison_path**(*output_dict*, *product_name*, *output_type*, *command_line_args=None*)

Replace variables specified by <varname> in compare_path.

Parameters

- **config** (*dict*) – Dictionary of output specifications, containing key "compare_path"
- **product_name** (*str*) – Current requested product name, all instances of <product> in compare_path replaced with product_name argument
- **output_type** (*str*) – Current requested output type, all instances of <output> in compare_path replaced with output argument

Returns
Return a single string with the fully specified comparison path for current product

Return type
str

geoips.plugins.modules.procflows.config_based.**update_output_dict_from_command_line_arg**

Update output dict from command line args.

geoips.plugins.modules.procflows.single_source module

Processing workflow for single data source processing.

geoips.plugins.modules.procflows.single_source.**add_filename_extra_field**(*xarray_obj*, *field_name*, *field_value*)

Add filename extra field.

geoips.plugins.modules.procflows.single_source.**call**(*fnames*, *command_line_args=None*)

Workflow for running products from a single data source.

Parameters

- **fnames** (*list*) – List of strings specifying full paths to input file names to process
- **command_line_args** (*dict*) – dictionary of command line arguments

Returns

Return list of strings specifying full paths to output products that were produced

Return type

list

See also:

geoips.commandline.args

Complete list of available command line args.

geoips.plugins.modules.procflows.single_source.**combine_filename_extra_fields**(*source_xarr*
dest_xarray

Combine filename extra fields.

geoips.plugins.modules.procflows.single_source.**get_alg_xarray**(*sect_xarrays*, *area_def*, *prod_plugin*, *resector=True*, *resampled_read=False*, *variable_names=None*)

Get alg xarray.

geoips.plugins.modules.procflows.single_source.**get_area_defs_from_command_line_args**(*co*
xo
va
ab
fil-
ter

Get area def from command line args.

geoips.plugins.modules.procflows.single_source.**get_filename**(*filename_formatter*, *prod_plugin=None*, *alg_xarray=None*, *area_def=None*, *supported_filenamer_types=None*, *output_dict=None*, *filename_formatter_kwargs=None*)

Get filename.

geoips.plugins.modules.procflows.single_source.**get_output_filenames**(*fname_formats*, *output_dict*, *prod_plugin*, *xarray_obj=None*, *area_def=None*, *supported_filenamer_types*

Get output filenames.

geoips.plugins.modules.procflows.single_source.**output_all_metadata**(*output_dict*, *output_fnames*, *metadata_fnames*, *xarray_obj*, *area_def=None*)

Output all metadata.

geoips.plugins.modules.procflows.single_source.**pad_area_definition**(*area_def*, *source_name=None*, *force_pad=False*, *x_scale_factor=1.5*, *y_scale_factor=1.5*)

Pad area definition.

geoips.plugins.modules.procflows.single_source.**plot_data**(*output_dict*, *alg_xarray*, *area_def*, *prod_plugin*, *output_kwargs*, *fused_xarray_dict=None*, *no_output=False*)

Plot data.

alg_xarray used for filename formats, etc. If included, fused_xarray_dict used for output format call

geoips.plugins.modules.procflows.single_source.**print_area_def**(*area_def*, *print_str*)

Print area def.

geoips.plugins.modules.procflows.single_source.**process_sectored_data_output**(*xobjs*, *variables*, *prod_plugin*, *output_dict*, *area_def=No*

Process sectored data output.

If product family is 'sectored_xarray_dict_to_output_format', call 'process_xarray_dict_to_output_format', store the result in a list, and return it.

geoips.plugins.modules.procflows.single_source.**process_xarray_dict_to_output_format**(*xo*
va
ab
pr
ou
pu
ar

Process xarray dict to output format.

geoips.plugins.modules.procflows.single_source.**remove_unsupported_kwargs**(*module*, *requested_kwargs*)

Remove unsupported keyword arguments.

geoips.plugins.modules.procflows.single_source.**verify_area_def**(*area_defs*, *check_area_def*, *data_start_datetime*, *data_end_datetime*, *time_range_hours=3*)

Verify current area definition is the closest to the actual data time.

When looping through multiple dynamic area definitions for a full data file that temporally covers more than one dynamic area_def, there is no way of knowing which dynamic area_def has the best coverage until AFTER we have actually sectored the data to the specific area_def.

Call this utility on the current area_def (check_area_def) for the sectored data file, plus the full list of area definitions (area_defs) that cover the FULL data file.

Returns

- True if the current area definition is NOT dynamic
- True if the current area definition IS dynamic and is the closest temporally to the sectored data.
- False if the current area definition is removed when filtering the list of area definitions based on the actual sectored data time.

Return type
bool

Module contents

Geoips procflow init file.

geoips.plugins.modules.readers package

Subpackages

geoips.plugins.modules.readers.utils package

Submodules

geoips.plugins.modules.readers.utils.geostationary_geolocation module

Generalized geolocation calculations for geostationary satellites.

exception geoips.plugins.modules.readers.utils.geostationary_geolocation.**AutoGenError**

Bases: Exception

Raise exception on auto generated geolocation error.

geoips.plugins.modules.readers.utils.geostationary_geolocation.**calculate_solar_angles**(

Calculate solar angles.

geoips.plugins.modules.readers.utils.geostationary_geolocation.**get_geolocation**(*dt*, *gmd*, *fldk_lats*, *fldk_lons*, *BAD-VALS*, *area_def*=

Gather and return the geolocation data for the input metadata.

Input metadata should be the metadata for a single ABI data file.

If latitude/longitude have not been calculated with the metadata form the input data file they will be recalculated and stored for future use. They shouldn't change often. This will be slow the first time it is called after a metadata update, but fast thereafter.

The same is true for satellite zenith and azimuth angles.

Solar zenith ang azimuth angles are always calculated on the fly. This is because they actually change. This may be slow for full-disk images.

geoips.plugins.modules.readers.utils.geostationary_geolocation.**get_geolocation_cache_f**

Set the location and filename format for the cached geolocation files.

There is a separate filename format for satellite latlons and sector latlons

Notes

Changing geolocation filename format will force recreation of all files, which can be problematic for large numbers of sectors.

geoips.plugins.modules.readers.utils.geostationary_geolocation.**get_indexes**(*metadata*, *lats*, *lons*, *area_def*)

Return two 2-D arrays containing the X and Y indexes.

These are indices that should be used from the raw data for the input sector definition.

geoips.plugins.modules.readers.utils.geostationary_geolocation.**get_satellite_angles**(*me*
lat
lon
BA
VA
sec

Get satellite angles.

geoips.plugins.modules.readers.utils.hrit_reader module

Utility for reading HRIT datasets.

class geoips.plugins.modules.readers.utils.hrit_reader.**HritDtype**

Bases: object

HRIT data type.

```
types = {'byte':  'B', 'int16':  '>i2', 'int32':  '>i4', 'int64':
'>i8', 'int8':  'i1', 'uint16':  '>u2', 'uint32':  '>u4', 'uint64':
'>u8', 'uint8':  'u1', 'unicode':  'U'}
```

exception geoips.plugins.modules.readers.utils.hrit_reader.**HritError**(*msg*, *code=None*)

Bases: Exception

Raise exception when errors occur in reading xRIT data files.

class geoips.plugins.modules.readers.utils.hrit_reader.**HritFile**(*fname*)

Bases: object

Hrit File class.

property annotation_metadata

Return annotation metadata (ie, platform, start time, etc).

property band

Return band specified in block_128, if it exists.

property basename

Return file basename.

property block_info

Block info.

property block_map

Return block map.

property compressed

Return True if compressed, False if not.

decompress(*outdir*)

Decompress an xRIT file and return a file handle.

The file will be decompressed to *outdir* and read from there.

Returns an HritFile instance for the decompressed file. If already decompressed, raises an HritError.

property dirname
Return file dirname.

property epilogue
Return epilogue.

property file_type
Return file_type.

property geolocation_metadata
Return geolocation metadata.

property metadata
Return metadata.

property name
Return name.

property prologue
Return prologue.

property segment
Return segment specified in block_128, if it exists.

property start_datetime
Return start_datetime.

geoips.plugins.modules.readers.utils.hrit_reader.**read10bit**(*buff*)
Read 10 bit little endian data from a buffer.

Returns
16 bit unsigned int.

Return type
int

geoips.plugins.modules.readers.utils.remss_reader module

Read derived surface winds from REMSS SMAP, WINDSAT, and AMSR netcdf data.

geoips.plugins.modules.readers.utils.remss_reader.**read_remss_data**(*wind_xarray*, *data_type*)
Reformat SMAP or WindSat xarray object appropriately.

variables: latitude, longitude, time, wind_speed_kts attributes: source_name, platform_name, data_provider, interpolation_radius_of_influence

Module contents

geoips reader utils init file.

Submodules

geoips.plugins.modules.readers.abi_l2_netcdf module

ABI Level 2 NetCDF reader.

geoips.plugins.modules.readers.abi_l2_netcdf.**calculate_abi_geolocation**(*metadata*, *area_def*)

Calculate ABI geolocation.

geoips.plugins.modules.readers.abi_l2_netcdf.**call**(*fnames*, *area_def=None*, *metadata_only=False*, *chans=False*, *self_register=False*)

Read ABI Level 2 NetCDF data from a list of filenames.

Parameters

- **fnames** (*list*) –
  - List of strings, full paths to files
- **metadata_only** (*bool, default=False*) –
  - Return before actually reading data if True
- **chans** (*list of str, default=None*) –
  - List of desired channels (skip unneeded variables as needed).
  - Include all channels if None.
- **area_def** (*pyresample.AreaDefinition, default=None*) –
  - Specify region to read
  - Read all data if None.
- **self_register** (*str or bool, default=False*) –
  - register all data to the specified dataset id (as specified in the return dictionary keys).
  - Read multiple resolutions of data if False.

Returns

- dictionary of xarray.Dataset objects with required Variables and Attributes.
- Dictionary keys can be any descriptive dataset ids.

Return type
dict of xarray.Datasets

See also:

Xarray and NetCDF Metadata Standards
Additional information regarding required attributes and variables for GeoIPS-formatted xarray Datasets.

`geoips.plugins.modules.readers.abi_l2_netcdf.get_metadata(fname)`
Get metadata.

geoips.plugins.modules.readers.abi_netcdf module

Standard GeoIPS xarray dictionary based ABI NetCDF data reader.

`geoips.plugins.modules.readers.abi_netcdf.call(fnames, metadata_only=False, chans=None, area_def=None, self_register=False)`
Read ABI NetCDF data from a list of filenames.

Parameters

- **fnames** (*list*) –
  - List of strings, full paths to files
- **metadata_only** (*bool, default=False*) –
  - Return before actually reading data if True
- **chans** (*list of str, default=None*) –
  - List of desired channels (skip unneeded variables as needed).
  - Include all channels if None.
- **area_def** (*pyresample.AreaDefinition, default=None*) –
  - Specify region to read
  - Read all data if None.
- **self_register** (*str or bool, default=False*) –
  - register all data to the specified dataset id (as specified in the return dictionary keys).

– Read multiple resolutions of data if False.

Returns

- dictionary of xarray.Dataset objects with required Variables and Attributes.
- Dictionary keys can be any descriptive dataset ids.

Return type
dict of xarray.Datasets

See also:

Xarray and NetCDF Metadata Standards
Additional information regarding required attributes and variables for GeoIPS-formatted xarray Datasets.

geoips.plugins.modules.readers.abi_netcdf.**get_band_metadata**(*all_metadata*)

Get band metadata.

This method basically just reformats the all_metadata dictionary that is set based on the metadata found in the netcdf object itself to reference channel names as opposed to filenames as the dictionary keys.

geoips.plugins.modules.readers.abi_netcdf.**get_data**(*md*, *gvars*, *rad=False*, *ref=False*, *bt=False*)

Read data for a full channel's worth of files.

geoips.plugins.modules.readers.abi_netcdf.**get_latitude_longitude**(*metadata*, *BADVALS*, *sect=None*)

Get latitudes and longitudes.

This routine accepts a dictionary containing metadata as read from a NCDF4 format file, and returns latitudes and longitudes for a full disk.

geoips.plugins.modules.readers.abi_netcdf.**metadata_to_datetime**(*metadata*)

Use information from the metadata to get the image datetime.

geoips.plugins.modules.readers.ahi_hsd module

Advanced Himawari Imager Data Reader.

exception geoips.plugins.modules.readers.ahi_hsd.**AutoGenError**

Bases: `Exception`

Raise exception on geolocation autogeneration error.

geoips.plugins.modules.readers.ahi_hsd.**call**(*fnames*, *metadata_only=False*, *chans=None*, *area_def=None*, *self_register=False*)

Read AHI HSD data data from a list of filenames.

Parameters

- **fnames** (*list*) –
  - List of strings, full paths to files
- **metadata_only** (*bool, default=False*) –
  - Return before actually reading data if True
- **chans** (*list of str, default=None*) –
  - List of desired channels (skip unneeded variables as needed).
  - Include all channels if None.
- **area_def** (*pyresample.AreaDefinition, default=None*) –
  - Specify region to read
  - Read all data if None.
- **self_register** (*str or bool, default=False*) –
  - register all data to the specified dataset id (as specified in the return dictionary keys).
  - Read multiple resolutions of data if False.

Returns

- dictionary of xarray.Dataset objects with required Variables and Attributes.
- Dictionary keys can be any descriptive dataset ids.

Return type
dict of xarray.Datasets

See also:

Xarray and NetCDF Metadata Standards
Additional information regarding required attributes and variables for GeoIPS-formatted xarray Datasets.

geoips.plugins.modules.readers.ahi_hsd.**findDiff**(*d1*, *d2*, *path=''*)

Find diff.

geoips.plugins.modules.readers.ahi_hsd.**get_band_metadata**(*all_metadata*)

Get band metadata.

This method basically just reformats the all_metadata dictionary that is set based on the metadata found in the netcdf object itself to reference channel names as opposed to filenames as the dictionary keys.

geoips.plugins.modules.readers.ahi_hsd.**get_data**(*md*, *gvars*, *rad=False*, *ref=False*, *bt=False*, *zoom=1.0*)

Read data for a full channel's worth of files.

geoips.plugins.modules.readers.ahi_hsd.**get_latitude_longitude**(*metadata*, *BADVALS*, *area_def=None*)

Get latitudes and longitudes.

This routine accepts a dictionary containing metadata as read from an HSD format file and returns latitudes and longitudes for a full disk image.

Note: This code has been adapted from Dan Lindsey's Fortran90 code. This was done in three steps that ultimately culminated in faster, but more difficult to understand code. If you plan to edit this, I recommend that you return to Dan's original code, then explore the commented code here, then finally, look at the single-command statements that are currently being used.

geoips.plugins.modules.readers.ahi_hsd.**metadata_to_datetime**(*metadata*, *time_var='ob_start_time'*)

Use information from block_01 to get the image datetime.

geoips.plugins.modules.readers.ahi_hsd.**set_variable_metadata**(*xobj_attrs*, *band_metadata*, *dsname*, *varname*)

Set variable metadata.

MLS 20180914 Setting xobj_attrs at the variable level for the associated channel metadata pulled from the actual netcdf file. This will now be accessible from the scifile object. Additionally, pull out specifically the band_wavelength and attach it to the _varinfo at the variable level - this is automatically pulled from the xobj_attrs dictionary and set in the variable._varinfo dictionary in scifile/scifile.py and scifile/containers.py (see empty_varinfo at the beginning of containers.py for dictionary fields that are automatically pulled from the appropriate location in the xobj_attrs dictionary and set on the _varinfo dictionary)

geoips.plugins.modules.readers.ahi_hsd.**sort_by_band_and_seg**(*metadata*)

Sort by band and segment.

geoips.plugins.modules.readers.amsr2_netcdf module

Read AMSR2 data products.

geoips.plugins.modules.readers.amsr2_netcdf.**call**(*fnames*, *metadata_only=False*, *chans=None*, *area_def=None*, *self_register=False*)

Read AMSR2 netcdf data products.

Parameters

- **fnames** (*list*) –
  - List of strings, full paths to files
- **metadata_only** (*bool, default=False*) –
  - Return before actually reading data if True
- **chans** (*list of str, default=None*) –
  - List of desired channels (skip unneeded variables as needed).
  - Include all channels if None.
- **area_def** (*pyresample.AreaDefinition, default=None*) –
  - NOT YET IMPLEMENTED
  - Specify region to read
  - Read all data if None.
- **self_register** (*str or bool, default=False*) –
  - NOT YET IMPLEMENTED
  - register all data to the specified dataset id (as specified in the return dictionary keys).
  - Read multiple resolutions of data if False.

Returns

- dictionary of xarray.Dataset objects with required Variables and Attributes.
- Dictionary keys can be any descriptive dataset ids.

Return type
dict of xarray.Datasets

See also:

Xarray and NetCDF Metadata Standards
Additional information regarding required attributes and variables for GeoIPS-formatted xarray Datasets.

geoips.plugins.modules.readers.amsr2_netcdf.**read_amsr_data**(*full_xarray*, *chans*)
Read non-AMSR2_OCEAN data.

geoips.plugins.modules.readers.amsr2_netcdf.**read_amsr_mbt**(*full_xarray*, *varname*, *time_array=None*)
Reformat AMSR xarray object appropriately.

- variables: latitude, longitude, time, brightness temperature variables
- attributes: source_name, platform_name, data_provider, interpolation_radius_of_influence

geoips.plugins.modules.readers.amsr2_netcdf.**read_amsr_winds**(*wind_xarray*)
Reformat AMSR xarray object appropriately.

- variables: latitude, longitude, time, wind_speed_kts
- attributes: source_name, platform_name, data_provider, interpolation_radius_of_influence

geoips.plugins.modules.readers.amsr2_remss_winds_netcdf module

Read derived surface winds from REMSS AMSR netcdf data.

geoips.plugins.modules.readers.amsr2_remss_winds_netcdf.**call**(*fnames*, *metadata_only=False*, *chans=None*, *area_def=None*, *self_register=False*)
Read REMSS AMSR2 derived winds from netcdf data.

Parameters

- **fnames** (*list*) –
  - List of strings, full paths to files
- **metadata_only** (*bool, default=False*) –
  - NOT YET IMPLEMENTED
  - Return before actually reading data if True
- **chans** (*list of str, default=None*) –

- NOT YET IMPLEMENTED
- List of desired channels (skip unneeded variables as needed).
- Include all channels if None.

- **area_def** (*pyresample.AreaDefinition, default=None*) –
  - NOT YET IMPLEMENTED
  - Specify region to read
  - Read all data if None.
- **self_register** (*str or bool, default=False*) –
  - NOT YET IMPLEMENTED
  - register all data to the specified dataset id (as specified in the return dictionary keys).
  - Read multiple resolutions of data if False.

Returns

- dictionary of xarray.Dataset objects with required Variables and Attributes.
- Dictionary keys can be any descriptive dataset ids.

Return type
dict of xarray.Datasets

See also:

Xarray and NetCDF Metadata Standards
Additional information regarding required attributes and variables for GeoIPS-formatted xarray Datasets.

geoips.plugins.modules.readers.amsub_hdf module

Read AMSU-B and MHS passive microwave data files.

This reader is desgined for importing NOAA Advanced Microwave Sounding Unit (AMSU)-B/Microwave Humidity Sounder (HMS) and EUMETSAQT MHS data files in hdf format, such as

- NPR.MHOP.NP.D20153.S2046.E2230.B5832021.NS (N19),
- NPR.MHOP.NN.D20153.S1927.E2105.B7748081.NS (N18),
- NPR.MHOP.M1.D20153.S2229.E2318.B3998282.NS (METOP).

V1.0: Initial version, NRL-MRY, June 1, 2020

Basic information on AMSU-B product file:

```
Input SD Variables
(nscan, npix):
    npix=90 pixels per scan;
    nscan: vary with orbit
chan-1 AT:  89 GHz as ch16 anttenna temperature at V-pol   FOV 16km at␣
↪nadir
chan-2 AT:  150 (157) GHz as ch17 the number in bracket is for MHS from
                                  metops at V-pol, 16km at nadir
chan-3 AT:  183.31 +-1 GHz as ch18    at H-pol, 16km
chan-4 AT:  183.31 +-3 GHz as ch19    at H-pol, 16km
chan-5 AT:  183.31 +-7 (190.3) GHz as ch20    at V-pol, 16km
lat:     -90, 90     deg
lon:     -180, 180   deg
RR:  surface rainrate (mm/hr)
Snow: surafce snow cover
IWP:  ice water path (unit?)
SWE:  snow water equvelent  (unit)
Sfc_type:  surface type
Orbit_mode:   -1: ascending, 1: decending, 2: both
SFR:  snowfall rate (unit?)
LZ_angle:  local zinath angle (deg)
SZ_angle:  solar zinath angle (deg)

Vdata info (definition of AMSU-B date):
ScanTime_year
ScanTime_month
ScanTime_day
ScanTime_hour
ScanTime_minute
ScanTime_second
ScanTime_Jday
Time
```

geoips.plugins.modules.readers.amsub_hdf.**call**(*fnames*, *metadata_only=False*, *chans=None*, *area_def=None*, *self_register=False*)

Read AMSU-B hdf data products.

Parameters

- **fnames** (*list*) –
  - List of strings, full paths to files
- **metadata_only** (*bool, default=False*) –

  - NOT YET IMPLEMENTED
  - Return before actually reading data if True
- **chans** (*list of str, default=None*) –
  - NOT YET IMPLEMENTED
  - List of desired channels (skip unneeded variables as needed).
  - Include all channels if None.
- **area_def** (*pyresample.AreaDefinition, default=None*) –
  - NOT YET IMPLEMENTED
  - Specify region to read
  - Read all data if None.
- **self_register** (*str or bool, default=False*) –
  - NOT YET IMPLEMENTED
  - register all data to the specified dataset id (as specified in the return dictionary keys).
  - Read multiple resolutions of data if False.

Returns

- dictionary of xarray.Dataset objects with required Variables and Attributes.
- Dictionary keys can be any descriptive dataset ids.

Return type
dict of xarray.Datasets

See also:

Xarray and NetCDF Metadata Standards
Additional information regarding required attributes and variables for GeoIPS-formatted xarray Datasets.

geoips.plugins.modules.readers.amsub_mirs module

Read AMSU-B and MHS MIRS NetCDF data files.

This reader is desgined for importing the Advanced Microwave Sounding Unit (AMSU)-B/Microwave Humidity Sounder (HMS) and EUMETSAQT MHS data files in hdf5 from NOAA MIRS. These new data files have a different file name convention and and data structure from previous MSPPS. Example of MIRIS files:

- NPR-MIRS-IMG_v11r4_ma1_s202101111916000_e202101112012000_c202101112047200.nc
  - (ma1 is for metop-B)
- NPR-MIRS-IMG_v11r4_ma2_s202101111715000_e202101111857000_c202101111941370.nc
  - (ma2 is for metop-A)
- NPR-MIRS-IMG_v11r4_n19_s202101111730000_e202101111916000_c202101112001590.nc
  - (NOAA-19)

AMSU-A channel information:

```
Chan# / Freq(GHz) / bands / Bandwidth(MHz) / Beamwidth(deg) / NE#T(K) /
                          (Spec.) Polarization at nadir / Instrument␣
↪Component
1   23.800  1       270     3.3     0.30    V       A2
2   31.400  1       180     3.3     0.30    V       A2
3   50.300  1       180     3.3     0.40    V       A1-2
4   52.800  1       400     3.3     0.25    V       A1-2
5   53.596 +-115    2       170     3.3     0.25    H       A1-2
6   54.400  1       400     3.3     0.25    H       A1-1
7   54.940  1       400     3.3     0.25    V       A1-1
8   55.500  1       330     3.3     0.25    H       A1-2
9   f0=57,290.344   1       330     3.3     0.25    H       A1-1
10  f0+-217 2       78      3.3     0.40    H       A1-1
11  f0+-322.2+-48   4       36      3.3     0.40    H       A1-1
12  f0+-322.2+-22   4       16      3.3     0.60    H       A1-1
13  f0+-322.2+-10   4       8       3.3     0.80    H       A1-1
14  f0+-322.2+-4.5  4       3       3.3     1.20    H       A1-1
15  89,000  1       <6,000  3.3     0.50    V       A1-1
```

AMSU-B/MHS channel information:

```
Channel / Centre Frequency (GHz) / Bandwidth (MHz) / NeDT (K) /
                                 Calibration Accuracy (K) / pol. angle␣
↪(degree)
16  89.0    <6000   1.0     1.0     90-q                    (Vertical pol)
```

(continues on next page)

(continued from previous page)

```
17  150      <4000    1.0      1.0      90-q                  (Vertical)
18  183.31+-1.00     500      1.1      1.0      nospec       (Horizontal)
19  183.31+-3.00     1000     1.0      1.0      nospec       (Horizontal)
20  190.31+17.00     2000     1.2      1.0      90-q         (Vertical)
```

Since AMSU-A sensor is no longer available, we select only the AMSU-B/MHS channels. We decide to use the same names of the five channels used for previous NOAA MSPPS data files. i.e., select frequench index 15-19 (start from 0).

V1.0: Initial version, NRL-MRY, January 26, 2021

Dataset information:

```
Basic information on AMSU-B product file
index: 1     2    3    4      5      6    7
freq: 23.8,31.4,50.3,52.799,53.595,54.4,54.941,
55.499,57.29,57.29,57.29,57.29,57.29,57.29,
index:   8     9      10    11   12   13     14
freq: 55.499,57.29,57.29,57.29,57.29,57.29,57.29,
index: 15  16  17    18      19     20
freq: 89.,89.,157.,183.311,183.311,190.311

dimensions:
      Scanline = 2370 ;
      Field_of_view = 90 ;
      P_Layer = 100 ;
      Channel = 20 ;
      Qc_dim = 4 ;
variables:
      Freq(Channel): Central Frequencies (GHz)
      Polo(Channel): Polarizations
      ScanTime_year(Scanline): Calendar Year 20XX
      ScanTime_doy(Scanline: julian day 1-366
      ScanTime_month(Scanline): Calendar month 1-12
      ScanTime_dom(Scanline): Calendar day of the month 1-31
      ScanTime_hour(Scanline: hour of the day 0-23
      ScanTime_minute(Scanline): minute of the hour 0-59
      ScanTime_second(Scanline): second of the minute 0-59
      ScanTime_UTC(Scanline): Number of seconds since 00:00:00 UTC
      Orb_mode(Scanline): 0-ascending,1-descending
      Latitude(Scanline, Field_of_view):Latitude of the view (-90,90),
                                        unit: degree
      Longitude(Scanline, Field_of_view):Longitude of the view (-180,180),
                                         unit: degree
```

(continues on next page)

(continued from previous page)

```
     Sfc_type(Scanline, Field_of_view):type of surface:0-ocean,1-sea ice,
                                       2-land,3-snow
     Atm_type(Scanline, Field_of_view): type of atmosphere:currently␣
↪missing
                                    ( note: not needed for geoips␣
↪products)
     Qc(Scanline, Field_of_view, Qc_dim): Qc: 0-good, 1-usable with␣
↪problem,
                                          2-bad
     ChiSqr(Scanline, Field_of_view): Convergence rate: <3-good,>10-bad
     LZ_angle(Scanline, Field_of_view): Local Zenith Angle degree
     RAzi_angle(Scanline, Field_of_view):Relative Azimuth Angle 0-360␣
↪degree
     SZ_angle(Scanline, Field_of_view):Solar Zenith Angle (-90,90) degree
     BT(Scanline, Field_of_view, Channel): Channel Temperature (K)
     YM(Scanline, Field_of_view, Channel): UnCorrected Channel␣
↪Temperature(K)
     ChanSel(Scanline, Field_of_view, Channel):Channels Selection Used in
                                               Retrieval
     TPW(Scanline, Field_of_view): Total Precipitable Water (mm)
     CLW(Scanline, Field_of_view):Cloud liquid Water (mm)
     RWP(Scanline, Field_of_view): Rain Water Path (mm)
     LWP(Scanline, Field_of_view): Liquid Water Path (mm)
     SWP(Scanline, Field_of_view): Snow Water Path (mm)
     IWP(Scanline, Field_of_view): Ice Water Path (mm)
     GWP(Scanline, Field_of_view): Graupel Water Path (mm)
     RR(Scanline, Field_of_view): Rain Rate (mm/hr)
     Snow(Scanline, Field_of_view): Snow Cover (range: 0-1) i.e., 1 ->␣
↪100%
     SWE(Scanline, Field_of_view): Snow Water Equivalent (cm)
     SnowGS(Scanline, Field_of_view):Snow Grain Size (mm)
     SIce(Scanline, Field_of_view):Sea Ice Concentration (%)
     SIce_MY(Scanline, Field_of_view): Multi-Year Sea Ice Concentration (
↪%)
     SIce_FY(Scanline, Field_of_view): First-Year Sea Ice Concentration (
↪%)
     TSkin(Scanline, Field_of_view): Skin Temperature (K)
     SurfP(Scanline, Field_of_view): Surface Pressure (mb)
     Emis(Scanline, Field_of_view, Channel):Channel Emissivity
                                    (unit:1,  Emis:scale_factor = 0.
↪0001)
     SFR(Scanline, Field_of_view): Snow Fall Rate in mm/hr
     CldTop(Scanline, Field_of_view): Cloud Top Pressure (scale_factor =␣
```

(continues on next page)

(continued from previous page)

```
↪0.1)
     CldBase(Scanline, Field_of_view): Cloud Base Pressure
                                        (scale_factor = 0.1)
     CldThick(Scanline, Field_of_view): Cloud Thickness (scale_factor =␣
↪0.1)
     PrecipType(Scanline, Field_of_view): Precipitation Type (Frozen/
↪Liquid)
     RFlag(Scanline, Field_of_view): Rain Flag
     SurfM(Scanline, Field_of_view): Surface Moisture (scale_factor = 0.
↪1)
     WindSp(Scanline, Field_of_view): Wind Speed (m/s) (scale_factor = 0.
↪01
     WindDir(Scanline, Field_of_view: Wind Direction (scale_factor = 0.
↪01)
     WindU(Scanline, Field_of_view):U-direction Wind Speed (m/s)
                                    (scale_factor = 0.01)
     WindV(Scanline, Field_of_view: V-direction Wind Speed (m/s)
                                    (scale_factor = 0.01)
     Prob_SF(Scanline, Field_of_view): Probability of falling snow (%)
```

Additional info:

```
Variables (nscan, npix):  npix=90 pixels per scan; nscan: vary with orbit
chan-1 AT:  89 GHz              as ch16    anttenna temperature at V-pol
                                           FOV 16km at nadir
chan-2 AT:  150 (157) GHz       as ch17 the number in bracket is for MHS
                                        from metops at V-pol, 16km at nadir
chan-3 AT:  183.31 +-1 GHz          as ch18    at H-pol, 16km
chan-4 AT:  183.31 +-3 GHz          as ch19    at H-pol, 16km
chan-5 AT:  183.31 +-7 (190.3) GHz  as ch20    at V-pol, 16km
lat:     -90, 90     deg
lon:     -180, 180   deg
RR:  surface rainrate (mm/hr)
Snow: surafce snow cover
IWP:  ice water path
SWE:  snow water equvelent
Sfc_type:  surface type
Orbit_mode:   -1: ascending, 1: decending, 2: both
SFR:  snowfall rate (unit?)
LZ_angle:  local zinath angle (deg)
SZ_angle:  solar zinath angle (deg)
```

geoips.plugins.modules.readers.amsub_mirs.**call**(*fnames*, *metadata_only=False*, *chans=None*, *area_def=None*, *self_register=False*)

Read AMSU/MHS MIRS data products.

Parameters

- **fnames** (*list*) –
  - List of strings, full paths to files
- **metadata_only** (*bool, default=False*) –
  - Return before actually reading data if True
- **chans** (*list of str, default=None*) –
  - List of desired channels (skip unneeded variables as needed).
  - Include all channels if None.
- **area_def** (*pyresample.AreaDefinition, default=None*) –
  - Specify region to read
  - Read all data if None.
- **self_register** (*str or bool, default=False*) –
  - NOT YET IMPLEMENTED
  - register all data to the specified dataset id (as specified in the return dictionary keys).
  - Read multiple resolutions of data if False.

Returns

- dictionary of xarray.Dataset objects with required Variables and Attributes.
- Dictionary keys can be any descriptive dataset ids.

Return type

dict of xarray.Datasets

See also:

Xarray and NetCDF Metadata Standards

Additional information regarding required attributes and variables for GeoIPS-formatted xarray Datasets.

geoips.plugins.modules.readers.ascat_uhr_netcdf module

Read derived surface winds from BYU ASCAT UHR NetCDF data.

geoips.plugins.modules.readers.ascat_uhr_netcdf.**call**(*fnames*, *metadata_only=False*, *chans=None*, *area_def=None*, *self_register=False*)

Read ASCAT UHR derived winds or normalized radar cross section data.

Parameters

- **fnames** (*list*) –
  - List of strings, full paths to files
- **metadata_only** (*bool, default=False*) –
  - NOT YET IMPLEMENTED
  - Return before actually reading data if True
- **chans** (*list of str, default=None*) –
  - NOT YET IMPLEMENTED
  - List of desired channels (skip unneeded variables as needed).
  - Include all channels if None.
- **area_def** (*pyresample.AreaDefinition, default=None*) –
  - NOT YET IMPLEMENTED
  - Specify region to read
  - Read all data if None.
- **self_register** (*str or bool, default=False*) –
  - NOT YET IMPLEMENTED
  - register all data to the specified dataset id (as specified in the return dictionary keys).
  - Read multiple resolutions of data if False.

Returns

- dictionary of xarray.Dataset objects with required Variables and Attributes.
- Dictionary keys can be any descriptive dataset ids.

Return type
dict of xarray.Datasets

See also:

Xarray and NetCDF Metadata Standards
Additional information regarding required attributes and variables for GeoIPS-formatted xarray Datasets.

geoips.plugins.modules.readers.ascat_uhr_netcdf.**read_byu_data**(*wind_xarray*, *fname*)

Reformat ascat xarray object appropriately.

- variables: latitude, longitude, time, wind_speed_kts, wind_dir_deg_met
- attributes: source_name, platform_name, data_provider, interpolation_radius_of_influence

geoips.plugins.modules.readers.atms_hdf5 module

Reader to read a grannual NOAA ATMS SDR TBs in h5 format.

Output variables in xarray object for geoips processing system

V0: August 25, 2021

The date is generated by the NOAA community satellite processing package (CSPP), developed at CIMSS

Example of ATMS file names:

```
'SATMS_j01_d20210809_t0959306_e1000023_b19296_fnmoc_ops.h5'
    SDR TBs variables
'GATMO_j01_d20210809_t0959306_e1000023_b19296_fnmoc_ops.h5'
    SDR Geolocation variables
```

Dataset info:

```
TB[12,96,22]:  for each granuel

CHAN#  Center-Freq(GHz)  POL
1      23.8                V
2      31.4                V
3      50.3                H
4      51.76               H
5      52.8                H
6      53.596+-0.115       H
```

(continues on next page)

(continued from previous page)

```
7       54.4                H
8       54.94               H
9       55.5                H
10      57.290(f0)          H
11      f0 +-0.217          H
12      f0 +-0.322+-0.048   H
13      f0 +-0.322+-0.022   H
14      f0 +-0.322+-0.010   H
15      f0 +-0.322+-0.0045  H

16      88.2                V
17      165.5               H
18      183.1+-7            H
19      183.1+-4.5          H    (FNMOC used this chan for 183 GHz image)
20      183.1+-3.0          H
21      183.1+-1.8          H
22      183.1+-1.0          H

BeamTime[12,96]:  microsecond, i.e., 1*10^-6.  IET "IDPS Epoch Time" is␣
 ↪used
                  It is a signed 64-bit integer giving microseconds since
                  00:00:00.000000 Jan 1 1958.
BrightnessTemperatureFactors[2]: 1: scale (unitless); 2: offset (K)
BrightnessTemperature[12,96,22]: [scan,pix,chans]

SDR geolocation Info
Latitude/Longitude[12,96]:   for Chan 17 only (for initial product,
                                                it is used for all␣
 ↪channels)
BeamLatitude[12,96,5]:       for chan 1,2,3,16,17. They will be used for
                                 associated TBs at a later date.
BeamLongitude[12,96,5]:      for chan 1,2,3,16,17
SatelliteAzimuthAngle, SatelliteZenithAngle,
SolarAzimuthAngle, SolarZenithAngle[12,96]
```

Notes

Unix epoch time is defined as the number of seconds that have elapsed since January 1, 1970 (midnight UTC/GMT). Thus, there is a 12 years difference for the JPSS data when datetime.datetime.utcfromtimestamp(epoch) is used to convert the JPSS IDPS Epoch time to the humman-readable date.

This reader is developed to read one granual a time from ATMS npp and jpss-1(n20) data files.

The example files are:

- `SATMS_j01_d20210809_t0959306_e1000023_b19296_fnmoc_ops.h5`: for TBs. 'b': orbit#
- `GATMO_j01_d20210809_t0959306_e1000023_b19296_fnmoc_ops.h5`: for geolocations

`geoips.plugins.modules.readers.atms_hdf5.`**`call`**(*fnames*, *metadata_only=False*, *chans=None*, *area_def=None*, *self_register=False*)

Read ATMS hdf5 data products.

Parameters

- **fnames** (*list*) –
  - List of strings, full paths to files
- **metadata_only** (*bool, default=False*) –
  - NOT YET IMPLEMENTED
  - Return before actually reading data if True
- **chans** (*list of str, default=None*) –
  - NOT YET IMPLEMENTED
  - List of desired channels (skip unneeded variables as needed).
  - Include all channels if None.
- **area_def** (*pyresample.AreaDefinition, default=None*) –
  - NOT YET IMPLEMENTED
  - Specify region to read
  - Read all data if None.
- **self_register** (*str or bool, default=False*) –
  - NOT YET IMPLEMENTED

- register all data to the specified dataset id (as specified in the return dictionary keys).
- Read multiple resolutions of data if False.

Returns

- dictionary of xarray.Dataset objects with required Variables and Attributes.
- Dictionary keys can be any descriptive dataset ids.

Return type
dict of xarray.Datasets

See also:

Xarray and NetCDF Metadata Standards
Additional information regarding required attributes and variables for GeoIPS-formatted xarray Datasets.

geoips.plugins.modules.readers.atms_hdf5.**convert_epoch_to_datetime64**(*time_array*, *use_shape=None*)

Convert time to datetime object.

Parameters

- **time_array** (*array*) – Array of start time integers (multiplied by 1e-6 in function)
- **use_shape** (*tuple, optional*) – desired output shape of time array, by default None

Returns
array of converted datetime objects

Return type
array

geoips.plugins.modules.readers.atms_hdf5.**read_atms_file**(*fname*, *xarray_atms*)

Read ATCF data from file fname.

geoips.plugins.modules.readers.ewsg_netcdf module

Read EWS-G data.

This EWS-G(Electro-Optical Infrared Weather System - Geostationary) reader is designed for reading theEWS-G data files (EWS-G is renamed from GOES-13). The reader is only using the python functions and xarray variables. The reader is based on EWS-G data in netcdf4 format.

V1.0: NRL-Monterey, 02/25/2021

EWS-G file information:

```
Example of the gvar filename:   2020.1212.0012.goes-13.gvar.nc

Note that channel-3 is not available for EWS-G.
  gvar_Ch3(TIR=5.8-7.3um, ctr=6.48um,4km): unit=temp-deg(C), scale_
↪factor=0.01

variables:
  gvar_Ch1(VIS=0.55-0.75um, ctr=0.65um,1km): unit=albedo*100,  scale_
↪factor=0.01
  gvar_Ch2(MWIR=3.8-4.0um,  ctr=3.9um, 4km): unit=temp-deg(C), scale_
↪factor=0.01
  gvar_Ch4(TIR=10.2-11.2um, ctr=10.7um,4km): unit=temp-deg(C), scale_
↪factor=0.01
  gvar_Ch6(TIR=12.9-13.7um, ctr=13.3um 4km): unit=temp-deg(C), scale_
↪factor=0.01
  latitude: unit=degree
  longitude:unit=degree
  sat_zenith: unit=degree
  sun_zenith: unit=degree
  rel_azimuth:unit=degree

  variable array definition:  var(scan,pix); scan-->lines, pix-->samples

attributes: many
```

geoips.plugins.modules.readers.ewsg_netcdf.**call**(*fnames*, *metadata_only=False*, *chans=None*, *area_def=None*, *self_register=False*)

Read EWS-G data in netcdf4 format.

Parameters

- **fnames** (*list*) –
  - List of strings, full paths to files

- **metadata_only** (*bool, default=False*) –
  - NOT YET IMPLEMENTED
  - Return before actually reading data if True
- **chans** (*list of str, default=None*) –
  - NOT YET IMPLEMENTED
  - List of desired channels (skip unneeded variables as needed).
  - Include all channels if None.
- **area_def** (*pyresample.AreaDefinition, default=None*) –
  - NOT YET IMPLEMENTED
  - Specify region to read
  - Read all data if None.
- **self_register** (*str or bool, default=False*) –
  - NOT YET IMPLEMENTED
  - register all data to the specified dataset id (as specified in the return dictionary keys).
  - Read multiple resolutions of data if False.

Returns

- dictionary of xarray.Dataset objects with required Variables and Attributes.
- Dictionary keys can be any descriptive dataset ids.

Return type
dict of xarray.Datasets

See also:

Xarray and NetCDF Metadata Standards
Additional information regarding required attributes and variables for GeoIPS-formatted xarray Datasets.

geoips.plugins.modules.readers.geoips_netcdf module

Read pre-processed GeoIPS-formatted NetCDF data.

geoips.plugins.modules.readers.geoips_netcdf.**call**(*fnames*, *metadata_only=False*, *chans=None*, *area_def=None*, *self_register=False*)

Read preprocessed geoips netcdf output.

Parameters

- **fnames** (*list*) –
  - List of strings, full paths to files
- **metadata_only** (*bool, default=False*) –
  - NOT YET IMPLEMENTED
  - Return before actually reading data if True
- **chans** (*list of str, default=None*) –
  - NOT YET IMPLEMENTED
  - List of desired channels (skip unneeded variables as needed).
  - Include all channels if None.
- **area_def** (*pyresample.AreaDefinition, default=None*) –
  - NOT YET IMPLEMENTED
  - Specify region to read
  - Read all data if None.
- **self_register** (*str or bool, default=False*) –
  - NOT YET IMPLEMENTED
  - register all data to the specified dataset id (as specified in the return dictionary keys).
  - Read multiple resolutions of data if False.

Returns

- dictionary of xarray.Dataset objects with required Variables and Attributes.
- Dictionary keys can be any descriptive dataset ids.

Return type
dict of xarray.Datasets

See also:

Xarray and NetCDF Metadata Standards
Additional information regarding required attributes and variables for GeoIPS-formatted xarray Datasets.

geoips.plugins.modules.readers.geoips_netcdf.**read_xarray_netcdf**(*ncdf_fname*)

Read NetCDF file written out using the xarray python package.

geoips.plugins.modules.readers.gmi_hdf5 module

Read NASA GPM GMI hdf5 data files.

Read a grannual NASA GPM GMI TBs in h5 format (each grannual file is about 5 minutes GPM GMI data)

Output variables in xarray object for geoips processing system

V0: August 4, 2020

Dataset information:

```
variables in original TBs structure format
tb_info = { 'S1': {  'tb10v': 0,
                     'tb10h': 1,
                     'tb19v': 2,
                     'tb19h': 3,
                     'tb23v': 4,
                     'tb37v': 5,
                     'tb37h': 6,
                     'tb89v': 7,
                     'tb89h': 8,},
            'S2': { 'tb166v': 0,
                     'tb166h': 1,
                     'tb183_3v': 2,
                     'tb183_7v': 3}
          }
```

geoips.plugins.modules.readers.gmi_hdf5.**call**(*fnames*, *metadata_only=False*, *chans=None*, *area_def=None*, *self_register=False*)

Read GMI hdf5 data products.

Parameters

- **fnames** (*list*) –

- List of strings, full paths to files
- **metadata_only** (*bool, default=False*) –
  - NOT YET IMPLEMENTED
  - Return before actually reading data if True
- **chans** (*list of str, default=None*) –
  - NOT YET IMPLEMENTED
  - List of desired channels (skip unneeded variables as needed).
  - Include all channels if None.
- **area_def** (*pyresample.AreaDefinition, default=None*) –
  - NOT YET IMPLEMENTED
  - Specify region to read
  - Read all data if None.
- **self_register** (*str or bool, default=False*) –
  - NOT YET IMPLEMENTED
  - register all data to the specified dataset id (as specified in the return dictionary keys).
  - Read multiple resolutions of data if False.

Returns

- dictionary of xarray.Dataset objects with required Variables and Attributes.
- Dictionary keys can be any descriptive dataset ids.

Return type
dict of xarray.Datasets

See also:

Xarray and NetCDF Metadata Standards
Additional information regarding required attributes and variables for GeoIPS-formatted xarray Datasets.

geoips.plugins.modules.readers.gmi_hdf5.**read_gmi_file**(*fname*, *xarray_gmi*)

Read a single GMI file fname.

geoips.plugins.modules.readers.imerg_hdf5 module

Read IMERG rainfall data.

A reader is designed to import IMERG rainfall data for GeoIPS using only python libraries

Aug 17, 2020

for a IMERG 30min data file, the time is the start time of a 30min interval:
0000-0030-0100-0130-0200 ....

Dataset information:

```
Spatial resolution is 0.1 deg.
1st grid is at (-179.95, -89.95).
Last grid is at (175.95, 89.95)
variable array is (3600,1800)

 metadata['top']['dataprovider'] = 'NASA-GPM'

 dataset_info = { 'Grid': {'MWtime': 'HQobservationTime',
                           'MWid': 'HQprecipSource',
                           'MWrr': 'HQprecipitation',
                           'IRweight': 'IRkalmanFilterWeight',
                           'IRrr': 'IRprecipitation',
                           'rain': 'precipitationCal',
                           'rrQC': 'precipitationQualityIndex',
                           'rrUncal': 'precipitationUncal',
                           'rrProb': 'probabilityLiquidPrecipitation',
                           'rrErr': 'randomError',},
          }
```

geoips.plugins.modules.readers.imerg_hdf5.**call**(*fnames*, *metadata_only=False*, *chans=None*, *area_def=None*, *self_register=False*)

Read IMERG hdf5 rain rate data products.

Parameters

- **fnames** (*list*) –
  - List of strings, full paths to files
- **metadata_only** (*bool, default=False*) –
  - NOT YET IMPLEMENTED
  - Return before actually reading data if True
- **chans** (*list of str, default=None*) –

- NOT YET IMPLEMENTED
- List of desired channels (skip unneeded variables as needed).
- Include all channels if None.

- **area_def** (*pyresample.AreaDefinition, default=None*) –
  - NOT YET IMPLEMENTED
  - Specify region to read
  - Read all data if None.
- **self_register** (*str or bool, default=False*) –
  - NOT YET IMPLEMENTED
  - register all data to the specified dataset id (as specified in the return dictionary keys).
  - Read multiple resolutions of data if False.

Returns

- dictionary of xarray.Dataset objects with required Variables and Attributes.
- Dictionary keys can be any descriptive dataset ids.

Return type
dict of xarray.Datasets

See also:

Xarray and NetCDF Metadata Standards
Additional information regarding required attributes and variables for GeoIPS-formatted xarray Datasets.

geoips.plugins.modules.readers.mimic_netcdf module

MIMIC TPW NetCDF reader.

geoips.plugins.modules.readers.mimic_netcdf.**call**(*fnames*, *metadata_only=False*, *chans=None*, *area_def=None*, *self_register=False*)

Read TPW MIMIC data from a list of filenames.

Dataset information:

```
<xarray.Dataset>
Dimensions:              (lat: 721, lon: 1440)
Dimensions without coordinates: lat, lon
Data variables:
    lonArr               (lon) float32 ...
    latArr               (lat) float32 ...
    tpwGrid              (lat, lon) float32 ...
    tpwGridPrior         (lat, lon) float32 ...
    tpwGridSubseq        (lat, lon) float32 ...
    timeAwayGridPrior    (lat, lon) timedelta64[ns] ...
    timeAwayGridSubseq   (lat, lon) timedelta64[ns] ...
    footGridPrior        (lat, lon) float32 ...
    footGridSubseq       (lat, lon) float32 ...
    satGridPrior         (lat, lon) uint8 ...
    satGridSubseq        (lat, lon) uint8 ...
```

Parameters

- **fnames** (*list*) –
  - List of strings, full paths to files
- **metadata_only** (*bool, default=False*) –
  - Return before actually reading data if True
- **chans** (*list of str, default=None*) –
  - NOT YET IMPLEMENTED
  - List of desired channels (skip unneeded variables as needed).
  - Include all channels if None.
- **area_def** (*pyresample.AreaDefinition, default=None*) –
  - NOT YET IMPLEMENTED
  - Specify region to read
  - Read all data if None.
- **self_register** (*str or bool, default=False*) –
  - NOT YET IMPLEMENTED
  - register all data to the specified dataset id (as specified in the return dictionary keys).
  - Read multiple resolutions of data if False.

Returns

- dictionary of xarray.Dataset objects with required Variables and Attributes.
- Dictionary keys can be any descriptive dataset ids.

Return type
dict of xarray.Datasets

See also:

Xarray and NetCDF Metadata Standards
Additional information regarding required attributes and variables for GeoIPS-formatted xarray Datasets.

geoips.plugins.modules.readers.modis_hdf4 module

MODIS HDF4 reader.

This reader is designed for geoips for importing MODIS data files in hdf4 format Example files are:

```
AQUA:      MYD files
     MYD021KM.A2021004.2005.061.NRT.hdf
     MYD03.A2021004.2005.061.NRT.hdf
     MYD14.A2021004.2005.006.NRT.hdf
Terra:     MOD files
     MOD021KM.A2021004.2005.061.NRT.hdf
     MOD02HKM.A2021004.2005.061.NRT.hdf
     MOD02QKM.A2021004.2005.061.NRT.hdf
     MOD03.A2021004.2005.061.NRT.hdf
     MOD14.A2021004.2005.006.NRT.hdf
```

The MOD03 and MOD14 files have the geolocation (lat/lon) and sensor geoometry infomation, while other files have values at each channels.

geoips.plugins.modules.readers.modis_hdf4.**add_to_xarray**(*varname*, *nparr*, *xobj*, *cumulative_mask*, *data_type*)

Add variable to xarray Dataset.

geoips.plugins.modules.readers.modis_hdf4.**call**(*fnames*, *metadata_only=False*, *chans=None*, *area_def=None*, *self_register=False*)

Read MODIS Aqua and Terra hdf data files.

Parameters

- **fnames** (*list*) –

- List of strings, full paths to files
- **metadata_only** (*bool, default=False*) –
  - Return before actually reading data if True
- **chans** (*list of str, default=None*) –
  - List of desired channels (skip unneeded variables as needed).
  - Include all channels if None.
- **area_def** (*pyresample.AreaDefinition, default=None*) –
  - NOT YET IMPLEMENTED
  - Specify region to read
  - Read all data if None.
- **self_register** (*str or bool, default=False*) –
  - NOT YET IMPLEMENTED
  - register all data to the specified dataset id (as specified in the return dictionary keys).
  - Read multiple resolutions of data if False.

Returns

- dictionary of xarray.Dataset objects with required Variables and Attributes.
- Dictionary keys can be any descriptive dataset ids.

Return type
dict of xarray.Datasets

See also:

Xarray and NetCDF Metadata Standards
Additional information regarding required attributes and variables for GeoIPS-formatted xarray Datasets.

geoips.plugins.modules.readers.modis_hdf4.**parse_archive_metadata**(*metadata*, *metadatastr*)

Parse archive metadata.

geoips.plugins.modules.readers.modis_hdf4.**parse_core_metadata**(*metadata*, *metadatastr*)

Parse core metadata.

geoips.plugins.modules.readers.modis_hdf4.**parse_metadata**(*metadatadict*)

Parse MODIS metadata dictionary.

geoips.plugins.modules.readers.modis_hdf4.**parse_struct_metadata**(*metadata*, *metadatastr*)

Parse metadata struct.

geoips.plugins.modules.readers.saphir_hdf5 module

Read SAPHIR hdf files.

geoips.plugins.modules.readers.saphir_hdf5.**call**(*fnames*, *metadata_only=False*, *chans=None*, *area_def=None*, *self_register=False*)

Read SAPHIR hdf data products.

Parameters

- **fnames** (*list*) –
  - List of strings, full paths to files
- **metadata_only** (*bool, default=False*) –
  - NOT YET IMPLEMENTED
  - Return before actually reading data if True
- **chans** (*list of str, default=None*) –
  - NOT YET IMPLEMENTED
  - List of desired channels (skip unneeded variables as needed).
  - Include all channels if None.
- **area_def** (*pyresample.AreaDefinition, default=None*) –
  - NOT YET IMPLEMENTED
  - Specify region to read
  - Read all data if None.
- **self_register** (*str or bool, default=False*) –
  - NOT YET IMPLEMENTED
  - register all data to the specified dataset id (as specified in the return dictionary keys).
  - Read multiple resolutions of data if False.

Returns

- dictionary of xarray.Dataset objects with required Variables and Attributes.
- Dictionary keys can be any descriptive dataset ids.

Return type
dict of xarray.Datasets

See also:

Xarray and NetCDF Metadata Standards
Additional information regarding required attributes and variables for GeoIPS-formatted xarray Datasets.

geoips.plugins.modules.readers.sar_winds_netcdf module

Read derived surface winds from SAR netcdf data.

geoips.plugins.modules.readers.sar_winds_netcdf.**call**(*fnames*, *metadata_only=False*, *chans=None*, *area_def=None*, *self_register=False*)

Read SAR derived winds from netcdf data.

Parameters

- **fnames** (*list*) –
  - List of strings, full paths to files
- **metadata_only** (*bool, default=False*) –
  - NOT YET IMPLEMENTED
  - Return before actually reading data if True
- **chans** (*list of str, default=None*) –
  - NOT YET IMPLEMENTED
  - List of desired channels (skip unneeded variables as needed).
  - Include all channels if None.
- **area_def** (*pyresample.AreaDefinition, default=None*) –
  - NOT YET IMPLEMENTED
  - Specify region to read

- Read all data if None.

- **self_register** (*str or bool, default=False*) –
  - NOT YET IMPLEMENTED
  - register all data to the specified dataset id (as specified in the return dictionary keys).
  - Read multiple resolutions of data if False.

Returns

- dictionary of xarray.Dataset objects with required Variables and Attributes.
- Dictionary keys can be any descriptive dataset ids.

Return type
dict of xarray.Datasets

See also:

Xarray and NetCDF Metadata Standards
Additional information regarding required attributes and variables for GeoIPS-formatted xarray Datasets.

geoips.plugins.modules.readers.sar_winds_netcdf.**read_sar_data**(*wind_xarray*)

Reformat SAR xarray object appropriately.

- variables: latitude, longitude, time, wind_speed_kts
- attributes: source_name, platform_name, data_provider, interpolation_radius_of_influence

geoips.plugins.modules.readers.scat_knmi_winds_netcdf module

Read derived surface winds from KNMI scatterometer netcdf data.

geoips.plugins.modules.readers.scat_knmi_winds_netcdf.**call**(*fnames*, *metadata_only=False*, *chans=None*, *area_def=None*, *self_register=False*)

Read KNMI scatterometer derived winds from netcdf data.

Parameters

- **fnames** (*list*) –
  - List of strings, full paths to files

- **metadata_only** (*bool, default=False*) –
  - NOT YET IMPLEMENTED
  - Return before actually reading data if True
- **chans** (*list of str, default=None*) –
  - NOT YET IMPLEMENTED
  - List of desired channels (skip unneeded variables as needed).
  - Include all channels if None.
- **area_def** (*pyresample.AreaDefinition, default=None*) –
  - NOT YET IMPLEMENTED
  - Specify region to read
  - Read all data if None.
- **self_register** (*str or bool, default=False*) –
  - NOT YET IMPLEMENTED
  - register all data to the specified dataset id (as specified in the return dictionary keys).
  - Read multiple resolutions of data if False.

Returns

- dictionary of xarray.Dataset objects with required Variables and Attributes.
- Dictionary keys can be any descriptive dataset ids.

Return type
dict of xarray.Datasets

See also:

Xarray and NetCDF Metadata Standards
Additional information regarding required attributes and variables for GeoIPS-formatted xarray Datasets.

geoips.plugins.modules.readers.scat_knmi_winds_netcdf.**read_knmi_data**(*wind_xarray*)

Reformat ascat xarray object appropriately.

- variables: latitude, longitude, time, wind_speed_kts, wind_dir_deg_met
- attributes: source_name, platform_name, data_provider, interpolation_radius_of_influence

geoips.plugins.modules.readers.scat_noaa_winds_netcdf module

Read derived surface winds from KNMI scatterometer netcdf data.

geoips.plugins.modules.readers.scat_noaa_winds_netcdf.**call**(*fnames*, *metadata_only=False*, *chans=None*, *area_def=None*, *self_register=False*)

Read KNMI scatterometer derived winds from netcdf data.

Parameters

- **fnames** (*list*) –
  - List of strings, full paths to files
- **metadata_only** (*bool, default=False*) –
  - NOT YET IMPLEMENTED
  - Return before actually reading data if True
- **chans** (*list of str, default=None*) –
  - NOT YET IMPLEMENTED
  - List of desired channels (skip unneeded variables as needed).
  - Include all channels if None.
- **area_def** (*pyresample.AreaDefinition, default=None*) –
  - NOT YET IMPLEMENTED
  - Specify region to read
  - Read all data if None.
- **self_register** (*str or bool, default=False*) –
  - NOT YET IMPLEMENTED
  - register all data to the specified dataset id (as specified in the return dictionary keys).
  - Read multiple resolutions of data if False.

Returns

- dictionary of xarray.Dataset objects with required Variables and Attributes.
- Dictionary keys can be any descriptive dataset ids.

Return type
dict of xarray.Datasets

See also:

Xarray and NetCDF Metadata Standards
Additional information regarding required attributes and variables for GeoIPS-formatted xarray Datasets.

`geoips.plugins.modules.readers.scat_noaa_winds_netcdf.`**`read_noaa_data`**`(`*wind_xarray*`)`

Reformat ascat xarray object appropriately.

- variables: latitude, longitude, time, wind_speed_kts, wind_dir_deg_met
- attributes: source_name, platform_name, data_provider, interpolation_radius_of_influence

geoips.plugins.modules.readers.seviri_hrit module

Read SEVIRI hrit data.

Notes

1) At present, this reader does not work for High Resolution Visible data, which is ignored. Additionally, to ease generation of geolocation fields, datasets are assumed to be square and centered at their sub longitude.

20170330 MLS Try to only decompress what we need (VERY filename dependent),
make scifile and hrit channel names match (more filename dependence), don't try to decompress/open file for import_metadata (more filename dependence for time). satpy requires time to open file, and requires standard (decompressed) filenames, so built in filename dependence by using satpy.

`class geoips.plugins.modules.readers.seviri_hrit.`**`Chan`**`(`*name*`)`

Bases: `object`

Channel class.

`property band`

Band property.

`property band_num`

Band number property.

`property name`

Name property.

property type

Type property.

class geoips.plugins.modules.readers.seviri_hrit.**ChanList**(*chans*)

Bases: object

ChanList Class.

property bands

Bands property.

property chans

Chans property.

property names

Names property.

exception geoips.plugins.modules.readers.seviri_hrit.**XritError**(*msg*, *code=None*)

Bases: Exception

Raise exception on XritError.

geoips.plugins.modules.readers.seviri_hrit.**calculate_chebyshev_polynomial**(*coefs*, *start_dt*, *end_dt*, *dt*)

Calculate Chebyshev Polynomial.

geoips.plugins.modules.readers.seviri_hrit.**call**(*fnames*, *metadata_only=False*, *chans=None*, *area_def=None*, *self_register=False*)

Read SEVIRI hrit data products.

Parameters

- **fnames** (*list*) –
  - List of strings, full paths to files
- **metadata_only** (*bool, default=False*) –
  - Return before actually reading data if True
- **chans** (*list of str, default=None*) –
  - List of desired channels (skip unneeded variables as needed).
  - Include all channels if None.
- **area_def** (*pyresample.AreaDefinition, default=None*) –

- Specify region to read
- Read all data if None.

- **self_register** (*str or bool, default=False*) –
  - register all data to the specified dataset id (as specified in the return dictionary keys).
  - Read multiple resolutions of data if False.

Returns

- dictionary of xarray.Dataset objects with required Variables and Attributes.
- Dictionary keys can be any descriptive dataset ids.

Return type
dict of xarray.Datasets

See also:

Xarray and NetCDF Metadata Standards
Additional information regarding required attributes and variables for GeoIPS-formatted xarray Datasets.

geoips.plugins.modules.readers.seviri_hrit.**compare_dicts**(*d1*, *d2*, *skip=None*)

Compare the values in two dictionaries.

If they are equal, return True, otherwise False If skip is set and contains one of the keys, skip that key

geoips.plugins.modules.readers.seviri_hrit.**countsToRad**(*counts*, *slope*, *offset*)

Convert counts to rad.

geoips.plugins.modules.readers.seviri_hrit.**get_latitude_longitude**(*gmd*, *BADVALS*, *area_def*)

Generate full-disk latitudes and longitudes.

geoips.plugins.modules.readers.seviri_hrit.**get_top_level_metadata**(*fnames*, *sect*)

Get top level metadata.

geoips.plugins.modules.readers.seviri_hrit.**radToBT**(*rad*, *platform*, *band*)

Convert rad to BT.

geoips.plugins.modules.readers.seviri_hrit.**radToRef**(*rad*, *sun_zen*, *platform*, *band*)

Convert Rad to Ref.

geoips.plugins.modules.readers.sfc_winds_text module

Read derived surface winds from SAR, SMAP, SMOS, and AMSR text data.

geoips.plugins.modules.readers.sfc_winds_text.**call**(*fnames*, *metadata_only=False*, *chans=None*, *area_def=None*, *self_register=False*)

Read one of SAR, SMAP, SMOS, AMSR derived winds from text data.

Parameters

- **fnames** (*list*) –
  - List of strings, full paths to files
- **metadata_only** (*bool, default=False*) –
  - NOT YET IMPLEMENTED
  - Return before actually reading data if True
- **chans** (*list of str, default=None*) –
  - NOT YET IMPLEMENTED
  - List of desired channels (skip unneeded variables as needed).
  - Include all channels if None.
- **area_def** (*pyresample.AreaDefinition, default=None*) –
  - NOT YET IMPLEMENTED
  - Specify region to read
  - Read all data if None.
- **self_register** (*str or bool, default=False*) –
  - NOT YET IMPLEMENTED
  - register all data to the specified dataset id (as specified in the return dictionary keys).
  - Read multiple resolutions of data if False.

Returns

- dictionary of xarray.Dataset objects with required Variables and Attributes.
- Dictionary keys can be any descriptive dataset ids.

Return type
dict of xarray.Datasets

See also:

Xarray and NetCDF Metadata Standards
Additional information regarding required attributes and variables for GeoIPS-formatted xarray Datasets.

geoips.plugins.modules.readers.smap_remss_winds_netcdf module

Read derived surface winds from REMSS SMAP netcdf data.

geoips.plugins.modules.readers.smap_remss_winds_netcdf.**call**(*fnames*, *metadata_only=False*, *chans=None*, *area_def=None*, *self_register=False*)

Read one of SMAP derived winds from netcdf data.

Parameters

- **fnames** (*list*) –
  - List of strings, full paths to files
- **metadata_only** (*bool, default=False*) –
  - NOT YET IMPLEMENTED
  - Return before actually reading data if True
- **chans** (*list of str, default=None*) –
  - NOT YET IMPLEMENTED
  - List of desired channels (skip unneeded variables as needed).
  - Include all channels if None.
- **area_def** (*pyresample.AreaDefinition, default=None*) –
  - NOT YET IMPLEMENTED
  - Specify region to read
  - Read all data if None.
- **self_register** (*str or bool, default=False*) –
  - NOT YET IMPLEMENTED
  - register all data to the specified dataset id (as specified in the return dictionary keys).
  - Read multiple resolutions of data if False.

Returns

- dictionary of xarray.Dataset objects with required Variables and Attributes.
- Dictionary keys can be any descriptive dataset ids.

Return type
dict of xarray.Datasets

See also:

Xarray and NetCDF Metadata Standards
Additional information regarding required attributes and variables for GeoIPS-formatted xarray Datasets.

geoips.plugins.modules.readers.smos_winds_netcdf module

Read derived surface winds from SAR, SMAP, SMOS, and AMSR netcdf data.

`geoips.plugins.modules.readers.smos_winds_netcdf.call`(*fnames*, *metadata_only=False*, *chans=None*, *area_def=None*, *self_register=False*)

Read SMOS derived winds from netcdf data.

Parameters

- **fnames** (*list*) –
  - List of strings, full paths to files
- **metadata_only** (*bool, default=False*) –
  - NOT YET IMPLEMENTED
  - Return before actually reading data if True
- **chans** (*list of str, default=None*) –
  - NOT YET IMPLEMENTED
  - List of desired channels (skip unneeded variables as needed).
  - Include all channels if None.
- **area_def** (*pyresample.AreaDefinition, default=None*) –
  - NOT YET IMPLEMENTED
  - Specify region to read

- Read all data if None.

- **self_register** (*str or bool, default=False*) –
  - NOT YET IMPLEMENTED
  - register all data to the specified dataset id (as specified in the return dictionary keys).
  - Read multiple resolutions of data if False.

Returns

- dictionary of xarray.Dataset objects with required Variables and Attributes.
- Dictionary keys can be any descriptive dataset ids.

Return type
dict of xarray.Datasets

See also:

Xarray and NetCDF Metadata Standards
Additional information regarding required attributes and variables for GeoIPS-formatted xarray Datasets.

geoips.plugins.modules.readers.smos_winds_netcdf.**read_smos_data**(*wind_xarray*, *fname*)

Reformat SMOS xarray object appropriately.

- variables: latitude, longitude, time, wind_speed_kts
- attributes: source_name, platform_name, data_provider, interpolation_radius_of_influence

geoips.plugins.modules.readers.ssmi_binary module

SSMI binary reader.

This SSMI reader is desgined for importing SSMI sdr data files (such as ssmi_orbital_sdrmi_f15_d20200427_s104500_e123100_r05323_cfnoc.def). This reader is created to read in TBs at 19 (V,H),22V, 37(V,H) and 85 (V,H) GHz channels. There are A/B scans for 85 GHz. The combined A/B scans will be used for TC imagery products at 85 GHz.

This GEOIPS python code is based on a SSMI SDR reader in C.

Convert SSMI_HIRES_AB for 85GHz and SSMI_LORES for 19-37GHz into xarray for GEOIPS framework

V1.0: Initial version, NRL-MRY, May 19, 2020

SSMI input data info:

```
pixels per scan:
LORES=64 for 19, 22, and 37 GHz channels;
HIRES=128 for 85 GHz channels
    19V[LORES]                                          FOV:   69km x 43km
    19H[LORES]
    22V[LORES]                                                 50km x 40km
    37V[LORES]                                                 37km x 28km
    37H[LORES]
    85V[HIRES][2]    [][0]: A scans; [][1]: B scans            15km x 13km
    85H[HIRES][2]

       -----header info-----
int32
    cyr, cmon, cday,       /* file creation date */
    chr, cmin,             /* file creation time */
    scans,                 /* number of scans in file (from DataSeq) */
    scid,                  /* spacecraft ID */
    rev,                   /* nominal rev  */
    bjld, bhr, bmin, bsec, /* begin day of year (julian day), time=hr,min,
↪sec */
    ejld, ehr, emin, esec, /* ending day of year, time */
    ajld, ahr, amin, asec, /* ascending node DOY, time */
    lsat;                  /* logical satellite ID */

   -----scan data-----
int32 scann;                 /* scan number (from ScanHdr1) */
int32 bst;                   /* B-scan start time (sec) scaled by 10000 */
double xtime;                /* bst as seconds since 0z 1 Jan 1987 */
uint16
    v19[LORES],              /* TBs */
    h19[LORES],
    v22[LORES],
    v37[LORES],
    h37[LORES],
    v85[HIRES][2],
    h85[HIRES][2],
    lon[HIRES][2];           /* longitudes */
int16 lat[HIRES][2];         /* latitudes */
char sft[HIRES][2];          /* surface types */
```

geoips.plugins.modules.readers.ssmi_binary.**call**(*fnames*, *metadata_only=False*, *chans=False*, *area_def=None*, *self_register=False*)

Read SSMI FNMOC Binary Data.

Parameters

- **fnames** (*list*) –
  - List of strings, full paths to files
- **metadata_only** (*bool, default=False*) –
  - NOT YET IMPLEMENTED
  - Return before actually reading data if True
- **chans** (*list of str, default=None*) –
  - NOT YET IMPLEMENTED
  - List of desired channels (skip unneeded variables as needed).
  - Include all channels if None.
- **area_def** (*pyresample.AreaDefinition, default=None*) –
  - NOT YET IMPLEMENTED
  - Specify region to read
  - Read all data if None.
- **self_register** (*str or bool, default=False*) –
  - NOT YET IMPLEMENTED
  - register all data to the specified dataset id (as specified in the return dictionary keys).
  - Read multiple resolutions of data if False.

Returns

- dictionary of xarray.Dataset objects with required Variables and Attributes.
- Dictionary keys can be any descriptive dataset ids.

Return type

dict of xarray.Datasets

See also:

Xarray and NetCDF Metadata Standards

Additional information regarding required attributes and variables for GeoIPS-formatted xarray Datasets.

geoips.plugins.modules.readers.ssmis_binary module

SSMIS Binary reader.

This code is converted from geoips v1 into geoipd v2 framework. This new version of reader is indepent from the GEOIPS system whose environmental parametrs must be used in V1. Now, only python functipns are used with geoips framework and xarray is utilized to process datasets for product applications.

Version Histopry:
V1: initial code, July 24, 2020, NRL-MRY

Input File
SSMIS SDR data

Output Fields
XARRAY onjectives to hold variables

geoips.plugins.modules.readers.ssmis_binary.**append_xarray_dicts**(*xobjs_list*)
Append two dictionaries of xarray objects.

geoips.plugins.modules.readers.ssmis_binary.**call**(*fnames*, *metadata_only=False*, *chans=False*, *area_def=None*, *self_register=False*)
Read SSMIS binary data products.

Parameters

- **fnames** (*list*) –
  - List of strings, full paths to files
- **metadata_only** (*bool, default=False*) –
  - Return before actually reading data if True
- **chans** (*list of str, default=None*) –
  - NOT YET IMPLEMENTED
  - List of desired channels (skip unneeded variables as needed).
  - Include all channels if None.
- **area_def** (*pyresample.AreaDefinition, default=None*) –
  - NOT YET IMPLEMENTED
  - Specify region to read
  - Read all data if None.
- **self_register** (*str or bool, default=False*) –

- NOT YET IMPLEMENTED
- register all data to the specified dataset id (as specified in the return dictionary keys).
- Read multiple resolutions of data if False.

Returns

- dictionary of xarray.Dataset objects with required Variables and Attributes.
- Dictionary keys can be any descriptive dataset ids.

Return type
dict of xarray.Datasets

See also:

Xarray and NetCDF Metadata Standards
Additional information regarding required attributes and variables for GeoIPS-formatted xarray Datasets.

geoips.plugins.modules.readers.ssmis_binary.**read_ssmis_data_file**(*fname*, *metadata_only=False*)

Read a single SSMIS data file.

geoips.plugins.modules.readers.viirs_netcdf module

VIIRS NetCDF reader.

This VIIRS reader is designed for reading the NPP/JPSS VIIRS files geoips. The reader is only using the python functions and xarray variables. Although the file name indicates the data is in netcdf4 format.

Thus, the reader is based on the netcdf4 data format.

The orginal reader (viirs_aotcimss_ncdf4_reader.py) was developed for geoips1, which applied many geoips1 function.

V1.0: NRL-Monterey, 09/17/2020

VIIRS file infOrmation:

```
There are 6 files for each time of VIIRS data, i.e.,
For NASA NPP VIIRS
20200826.074800.npp.viirs.viirs_npp_nasaearthdata_x.x.VNP02DNB.sdr.x.x.nc
20200826.074800.npp.viirs.viirs_npp_nasaearthdata_x.x.VNP02IMG.sdr.x.x.nc
```

(continues on next page)

(continued from previous page)

```
20200826.074800.npp.viirs.viirs_npp_nasaearthdata_x.x.VNP02MOD.sdr.x.x.nc
20200826.074800.npp.viirs.viirs_npp_nasaearthdata_x.x.VNP03DNB.sdr.x.x.nc
20200826.074800.npp.viirs.viirs_npp_nasaearthdata_x.x.VNP03IMG.sdr.x.x.nc
20200826.074800.npp.viirs.viirs_npp_nasaearthdata_x.x.VNP03MOD.sdr.x.x.nc

For JPSS VIIRS
20200914.150000.npp.viirs.viirs_sips_jpss_uwssec_001.x.VJ102DNB.sdr.x.x.nc
20200914.150000.npp.viirs.viirs_sips_jpss_uwssec_001.x.VJ102IMG.sdr.x.x.nc
20200914.150000.npp.viirs.viirs_sips_jpss_uwssec_001.x.VJ102MOD.sdr.x.x.nc
20200914.150000.npp.viirs.viirs_sips_jpss_uwssec_001.x.VJ103DNB.sdr.x.x.nc
20200914.150000.npp.viirs.viirs_sips_jpss_uwssec_001.x.VJ103IMG.sdr.x.x.nc
20200914.150000.npp.viirs.viirs_sips_jpss_uwssec_001.x.VJ103MOD.sdr.x.x.nc

DNB: VIIRS day-night Band obs
MOD: VIIRS M-Band  obs
IMG: VIIRS I-Band  obs
```

The *.VNP02* files are for the data records, while the *.VNP03* files are for the geolocation data records.

The xarray of geoips reader need both the data and lat/lon info. Thus, this VIIRS reader is designed to read in the paired VNP02 and VNP03 files, depending on any one of DNB or IMG or MOD file. In order to minimize dupilcated excution of VIIRS files, additional adjust of excution of the VIIRS files will be needed (discussion with Mindy on how to do it).

geoips.plugins.modules.readers.viirs_netcdf.**add_to_xarray**(*varname*, *nparr*, *xobj*, *dataset_masks*, *data_type*, *nparr_mask*)

Add variable to xarray Dataset.

geoips.plugins.modules.readers.viirs_netcdf.**call**(*fnames*, *metadata_only=False*, *chans=None*, *area_def=None*, *self_register=False*)

Read VIIRS netcdf data products.

Parameters

- **fnames** (*list*) –
  - List of strings, full paths to files
- **metadata_only** (*bool, default=False*) –
  - Return before actually reading data if True
- **chans** (*list of str, default=None*) –
  - List of desired channels (skip unneeded variables as needed).

- Include all channels if None.

- **area_def** (*pyresample.AreaDefinition, default=None*) –
  - NOT YET IMPLEMENTED
  - Specify region to read
  - Read all data if None.
- **self_register** (*str or bool, default=False*) –
  - NOT YET IMPLEMENTED
  - register all data to the specified dataset id (as specified in the return dictionary keys).
  - Read multiple resolutions of data if False.

Returns

- dictionary of xarray.Dataset objects with required Variables and Attributes.
- Dictionary keys can be any descriptive dataset ids.

Return type
dict of xarray.Datasets

See also:

Xarray and NetCDF Metadata Standards
Additional information regarding required attributes and variables for GeoIPS-formatted xarray Datasets.

geoips.plugins.modules.readers.viirs_netcdf.**required_chan**(*chans*, *varnames*)
Return True if required channel.

geoips.plugins.modules.readers.viirs_netcdf.**required_geo**(*chans*, *data_type*)
Return True if required geolocation dataset.

geoips.plugins.modules.readers.viirs_netcdf.**required_geo_chan**(*xarrays*, *xvarname*)
Return True if required geolocation channel.

geoips.plugins.modules.readers.wfabba_ascii module

WFABBA ascii data reader.

WFABBA is a geostationary fire product produced by SSEC

geoips.plugins.modules.readers.wfabba_ascii.**call**(*fnames*, *metadata_only=False*, *chans=None*, *area_def=None*, *self_register=False*)

Read WFABBA ascii data from a list of filenames.

WFABBA ascii files contain list of fire detects with their latitude, longitude, and scan location

Parameters

- **fnames** (*list*) –
  - List of strings, full paths to files
- **metadata_only** (*bool, default=False*) –
  - Return before actually reading data if True
- **chans** (*list of str, default=None*) –
  - NOT YET IMPLEMENTED
  - List of desired channels (skip unneeded variables as needed).
  - Include all channels if None.
- **area_def** (*pyresample.AreaDefinition, default=None*) –
  - NOT YET IMPLEMENTED
  - Specify region to read
  - Read all data if None.
- **self_register** (*str or bool, default=False*) –
  - NOT YET IMPLEMENTED
  - register all data to the specified dataset id (as specified in the return dictionary keys).
  - Read multiple resolutions of data if False.

Returns

- dictionary of xarray.Dataset objects with required Variables and Attributes.
- Dictionary keys can be any descriptive dataset ids.

Return type
dict of xarray.Datasets

See also:

Xarray and NetCDF Metadata Standards
Additional information regarding required attributes and variables for GeoIPS-formatted xarray Datasets.

geoips.plugins.modules.readers.wfabba_ascii.**parse_header_line**(*line*)
Parse header line.

geoips.plugins.modules.readers.wfabba_ascii.**read_wfabba_header**(*wfabba_file*)
Read WFABBA header.

geoips.plugins.modules.readers.wfabba_ascii.**read_wfabba_text**(*wfabba_file*)
Read WFABBA text from file wfabba_file into xarray Dataset.

geoips.plugins.modules.readers.windsat_idr37_binary module

Windsat binary data reader.

This code is designed to read windsat sdr binary data (idr37) file for windsat 37 GHz products in GEOIPS environments. the input file name is something alike US058SORB-BINspp.wndmi_fws_d20191126_s134102_e153244_r87467_cfnmoc.idr37.

V1.0: initial version. Song Yang, NRL-MRY, 01/08/2020

errflag is the important parameter of the windsat edr dataset. It is a 32-bit integer which describes what is the current data point status. Here are the meaning of each bit:

```
0-7: Wilheit rain flag
  8: forward/aft scan (bit set to 1 for forward part of scan, 0 for aft␣
↪scan )
  9: ascending/descending pass flag (1 for ascending, 0 for descending)
 10: Warm load flag
 11: Warm load gains applied (1 = gains applied, 0 = gains not applied)
 12: Glare angle invalid because no 1 vector or LOS doesn't pierce earth
 13-18: Glare angle (0 to 30 represents angles of 0 to 60 degree in␣
↪increments
        of 2 deg; 31 represents angles .gt. 60 deg; 32 represents invalid
        glare angle)
 19: Cold load flag. If set to 1 the VH channel data had to be corrected␣
↪due
     to interference in the cold load signal, such as the moon or a
     geostationary satellite.
```

(continues on next page)

(continued from previous page)

```
20: Gain Saturation flag. Set to 1 when strong RFI causes the gain to␣
↪change.
    This is set if any TDR saturation flag is set at this frequency
23: used to hold the rfi flag so that it may be passed on to other␣
↪structures
    such as the resampling and intermediate structures. Finally RFI is␣
↪placed
    in the sdr structure.
other bits are spare
```

Here is the original sdr record in Fortran:

```
type IDRRecord_short
   real(double) :: JD2000           8 bytes
   real, dimension(4)::  stokes     16 bytes
   real :: latitude                 4 bytes
   real :: longitude                4 bytes
   real :: EIA                      4 bytes: earth incidence angle, the␣
↪angle
                                             on the ground between␣
↪vertical and
                                             the satellite look vector
   real :: PRA                      4 bytes: rotation of the polarization␣
↪plane
                                             from true
   real :: CAA                      4 bytes: compass azimuth angle on the␣
↪ground
   real :: tI45, tIcp, pra45        12 bytes
   integer :: errflag               4 bytes (32-bits integer):a set of bit
                                            flags for data quality and
                                            conditions (above explanation)
   integer :: Scan                  4 bytes: scan line number in the␣
↪orbit.
                                             WindSat scans every 1.9␣
↪seconds
   integer(int16) :: dcnum          2 bytes: pixel number along the scan,
                                             called 'downcount number',
                                             because the highest pixel␣
↪number
                                             is measured first.
   integer(int16) :: SurfaceType    2 bytes: legacy SSMI surface type
   integer(int16) :: scanAngle      2 bytes: angle on the ground between␣
↪the
```

(continues on next page)

(continued from previous page)

```
                                                flight direction and the look
                                                direction
   integer(int8) :: water2land ! copied from IDRL record   1 byte:
   integer(int8) :: land2water ! copied from IDRL record   1 byte:
end type IDRRecord_short
```

- Gain saturation is when a sudden, large signal causes the gain to change quickly and make averaged gain unreliable.
- Forward/Aft is for sensor view position
- The warm load flag indicates that calibration may be unreliable due to solar intrusion into the warm load.
- Cold load flags do not mean calibration is unreliable. It's a way for us to check the cold load correction algorithm.
- Sun glare is not something to worry about.
- The RFI flag is never set at 37 GHz.
- tI45, tIcp, pra45, scanAngleI, water2land, and land2water aren't commanly used.
  - The first four are for recreating the 6-element pre-Stokes polarization vector, and the last two measure coastal contamination

The actual idr37 data record (idr_record) in C:

```
typedef struct {
   double jd2000;
   float stokes[4];
   float plat;     lat of earth observation
   float plon;     lon of earth observation
   float eia;      radiance= ~53deg
   float pra;
   float caa;
   float slat;  latitude of satellite position?  not
   float slon;  longitude of satellite position? not
   float salt;  altitude of satellite (meter? km?) not
   int errflag;
   int scan;
   short dcnum;
   short surf;
   float spare;
idr_record;

Its total length of idr_record is 72 bytes
```

geoips.plugins.modules.readers.windsat_idr37_binary.**call**(*fnames*, *metadata_only=False*, *chans=None*, *area_def=None*, *self_register=False*)

Read Windsat binary data products.

Parameters

- **fnames** (*list*) –
  - List of strings, full paths to files
- **metadata_only** (*bool, default=False*) –
  - Return before actually reading data if True
- **chans** (*list of str, default=None*) –
  - NOT YET IMPLEMENTED
  - List of desired channels (skip unneeded variables as needed).
  - Include all channels if None.
- **area_def** (*pyresample.AreaDefinition, default=None*) –
  - NOT YET IMPLEMENTED
  - Specify region to read
  - Read all data if None.
- **self_register** (*str or bool, default=False*) –
  - NOT YET IMPLEMENTED
  - register all data to the specified dataset id (as specified in the return dictionary keys).
  - Read multiple resolutions of data if False.

Returns

- dictionary of xarray.Dataset objects with required Variables and Attributes.
- Dictionary keys can be any descriptive dataset ids.

Return type
dict of xarray.Datasets

See also:

Xarray and NetCDF Metadata Standards
Additional information regarding required attributes and variables for GeoIPS-formatted xarray Datasets.

geoips.plugins.modules.readers.windsat_remss_winds_netcdf module

Read derived surface winds from REMSS WINDSAT netcdf data.

geoips.plugins.modules.readers.windsat_remss_winds_netcdf.**call**(*fnames*, *metadata_only=False*, *chans=None*, *area_def=None*, *self_register=False*)

Read Remote Sensing Systems Windsat data.

Parameters

- **fnames** (*list*) –
  - List of strings, full paths to files
- **metadata_only** (*bool, default=False*) –
  - NOT YET IMPLEMENTED
  - Return before actually reading data if True
- **chans** (*list of str, default=None*) –
  - NOT YET IMPLEMENTED
  - List of desired channels (skip unneeded variables as needed).
  - Include all channels if None.
- **area_def** (*pyresample.AreaDefinition, default=None*) –
  - NOT YET IMPLEMENTED
  - Specify region to read
  - Read all data if None.
- **self_register** (*str or bool, default=False*) –
  - NOT YET IMPLEMENTED
  - register all data to the specified dataset id (as specified in the return dictionary keys).
  - Read multiple resolutions of data if False.

Returns

- dictionary of xarray.Dataset objects with required Variables and Attributes.
- Dictionary keys can be any descriptive dataset ids.

Return type
dict of xarray.Datasets

See also:

Xarray and NetCDF Metadata Standards
Additional information regarding required attributes and variables for GeoIPS-formatted xarray Datasets.

Module contents

Geoips readers init file.

geoips.plugins.modules.sector_metadata_generators package

Submodules

geoips.plugins.modules.sector_metadata_generators.bdeck_parser module

TC trackfile parser for B-Deck formatted TC deck files.

Each B-Deck file contains the full history of storm BEST tracks, on storm per location per line (split between 3 lines each in comments for readability):

```
AL, 20, 2020091318,   , BEST,   0, 126N,  374W,  30, 1006, TD,   0,    , ␣
↪  0,
    0,    0,    0, 1011,  240, 100,  40,   0,   L,   0,    ,   0,   0,
          TWENTY, M, 12, NEQ,   60,    0,    0,   60, genesis-num, 039,
AL, 20, 2020091400,   , BEST,   0, 130N,  386W,  30, 1006, TD,   0,    , ␣
↪  0,
        0,    0,    0, 1011,  240, 100,  40,   0,   L,   0,    ,   0,   0,
          TWENTY, M, 12, NEQ,   60,   60,    0,    0, genesis-num, 039,
AL, 20, 2020091406,   , BEST,   0, 130N,  404W,  35, 1004, TS,  34, NEQ, ␣
↪  0,
        0,   40,   40, 1011,  240,  40,   0,   0,   L,   0,    ,   0,   0,
          TEDDY, M,  0,    ,    0,    0,    0,    0, genesis-num, 039,
AL, 20, 2020091412,   , BEST,   0, 128N,  422W,  35, 1004, TS,  34, NEQ, ␣
↪ 50,
```

(continues on next page)

(continued from previous page)

```
       30,    0,   50, 1011,  240,  40,  45,   0,   L,   0,    ,   0,   0,
          TEDDY, M,  0,    ,    0,    0,    0,    0, genesis-num, 039,
AL, 20, 2020091418,    , BEST,   0, 129N,  434W,  40, 1003, TS,  34, NEQ, ␣
↪ 80,
       40,    0,   70, 1012,  210,  50,  55,   0,   L,   0,    ,   0,   0,
          TEDDY, M, 12, NEQ,   90,   30,    0,   30, genesis-num, 039,
AL, 20, 2020091500,    , BEST,   0, 130N,  445W,  45, 1002, TS,  34, NEQ, ␣
↪100,
       50,    0,   80, 1012,  210,  40,  55,   0,   L,   0,    ,   0,   0,
          TEDDY, M, 12, NEQ,   90,   30,    0,   30, genesis-num, 039,
AL, 20, 2020091506,    , BEST,   0, 134N,  455W,  50, 1001, TS,  34, NEQ, ␣
↪100,
       50,   20,   80, 1012,  210,  20,  60,   0,   L,   0,    ,   0,   0,
          TEDDY, M, 12, NEQ,  300,  210,   30,    0, genesis-num, 039,
AL, 20, 2020091506,    , BEST,   0, 134N,  455W,  50, 1001, TS,  50, NEQ, ␣
↪ 20,
        0,    0,    0, 1012,  210,  20,  60,   0,   L,   0,    ,   0,   0,
          TEDDY, M, 12, NEQ,  300,  210,   30,    0, genesis-num, 039,
AL, 20, 2020091512,    , BEST,   0, 138N,  466W,  55,  999, TS,  34, NEQ, ␣
↪140,
   60,   40,  160, 1011,  250,  20,  65,   0,   L,   0,    ,   0,   0,
      TEDDY, D, 12, NEQ,  300,  210,   30,   30, genesis-num, 039,
AL, 20, 2020091512,    , BEST,   0, 138N,  466W,  55,  999, TS,  50, NEQ, ␣
↪ 30,
    0,    0,   30, 1011,  250,  20,  65,   0,   L,   0,    ,   0,   0,
          TEDDY, D, 12, NEQ,  300,  210,   30,   30, genesis-num, 039,
AL, 20, 2020091518,    , BEST,   0, 142N,  475W,  55,  997, TS,  34, NEQ, ␣
↪140,
   80,   40,  160, 1011,  250,  20,  65,   0,   L,   0,    ,   0,   0,
          TEDDY, D, 12, NEQ,  300,  240,   60,   60, genesis-num, 039,
AL, 20, 2020091518,    , BEST,   0, 142N,  475W,  55,  997, TS,  50, NEQ, ␣
↪ 30,
    0,    0,   30, 1011,  250,  20,  65,   0,   L,   0,    ,   0,   0,
          TEDDY, D, 12, NEQ,  300,  240,   60,   60, genesis-num, 039,
AL, 20, 2020091600,    , BEST,   0, 147N,  480W,  65,  987, HU,  34, NEQ, ␣
↪140,
   80,   40,  150, 1010,  180,  20,  75,   0,   L,   0,    ,   0,   0,
          TEDDY, D,  0,    ,    0,    0,    0,    0, genesis-num, 039,
AL, 20, 2020091600,    , BEST,   0, 147N,  480W,  65,  987, HU,  50, NEQ, ␣
↪ 40,
   30,    0,   30, 1010,  180,  20,  75,   0,   L,   0,    ,   0,   0,
          TEDDY, D,  0,    ,    0,    0,    0,    0, genesis-num, 039,
AL, 20, 2020091600,    , BEST,   0, 147N,  480W,  65,  987, HU,  64, NEQ, ␣
```

(continues on next page)

(continued from previous page)

```
↪ 20,
    0,    0,    0, 1010,  180,  20,  75,   0,   L,   0,    ,   0,   0,
          TEDDY, D,  0,    ,    0,    0,    0,    0, genesis-num, 039,
AL, 20, 2020091606,    , BEST,   0, 154N,  486W,  80,  978, HU,  34, NEQ, ␣
↪140,
   80,   40,  150, 1010,  180,  20, 100,   0,   L,   0,    ,   0,   0,
          TEDDY, D, 12, NEQ,  300,  240,  120,  150, genesis-num, 039,
AL, 20, 2020091606,    , BEST,   0, 154N,  486W,  80,  978, HU,  50, NEQ, ␣
↪ 40,
   30,   20,   30, 1010,  180,  20, 100,   0,   L,   0,    ,   0,   0,
          TEDDY, D, 12, NEQ,  300,  240,  120,  150, genesis-num, 039,
AL, 20, 2020091606,    , BEST,   0, 154N,  486W,  80,  978, HU,  64, NEQ, ␣
↪ 20,
   10,   10,   20, 1010,  180,  20, 100,   0,   L,   0,    ,   0,   0,
          TEDDY, D, 12, NEQ,  300,  240,  120,  150, genesis-num, 039,
AL, 20, 2020091612,    , BEST,   0, 161N,  493W,  85,  973, HU,  34, NEQ, ␣
↪170,
  170,   40,  150, 1010,  180,  20, 100,   0,   L,   0,    ,   0,   0,
          TEDDY, D, 12, NEQ,  270,  270,  240,  270, genesis-num, 039,
AL, 20, 2020091612,    , BEST,   0, 161N,  493W,  85,  973, HU,  50, NEQ, ␣
↪ 50,
   50,   20,   30, 1010,  180,  20, 100,   0,   L,   0,    ,   0,   0,
          TEDDY, D, 12, NEQ,  270,  270,  240,  270, genesis-num, 039,
AL, 20, 2020091612,    , BEST,   0, 161N,  493W,  85,  973, HU,  64, NEQ, ␣
↪ 25,
   25,   10,   20, 1010,  180,  20, 100,   0,   L,   0,    ,   0,   0,
          TEDDY, D, 12, NEQ,  270,  270,  240,  270, genesis-num, 039,
AL, 20, 2020091618,    , BEST,   0, 168N,  502W,  85,  973, HU,  34, NEQ, ␣
↪190,
  100,   70,  170, 1010,  180,  20, 105,   0,   L,   0,    ,   0,   0,
          TEDDY, D, 12, NEQ,  300,  300,  240,  300, genesis-num, 039,
AL, 20, 2020091618,    , BEST,   0, 168N,  502W,  85,  973, HU,  50, NEQ, ␣
↪ 80,
   50,   30,   90, 1010,  180,  20, 105,   0,   L,   0,    ,   0,   0,
          TEDDY, D, 12, NEQ,  300,  300,  240,  300, genesis-num, 039,
AL, 20, 2020091618,    , BEST,   0, 168N,  502W,  85,  973, HU,  64, NEQ, ␣
↪ 30,
   25,    0,   30, 1010,  180,  20, 105,   0,   L,   0,    ,   0,   0,
          TEDDY, D, 12, NEQ,  300,  300,  240,  300, genesis-num, 039,
AL, 20, 2020091700,    , BEST,   0, 174N,  511W,  85,  973, HU,  34, NEQ, ␣
↪220,
  100,   80,  170, 1009,  210,  20, 100,   0,   L,   0,    ,   0,   0,
          TEDDY, D, 12, NEQ,  330,  300,  270,  300, genesis-num, 039,
```

(continues on next page)

(continued from previous page)

```
AL, 20, 2020091700,    , BEST,   0, 174N,  511W,  85,  973, HU,  50, NEQ, ␣
↪ 60,
   50,   50,   70, 1009,  210,  20, 100,   0,   L,   0,    ,   0,   0,
          TEDDY, D, 12, NEQ,  330,  300,  270,  300, genesis-num, 039,
AL, 20, 2020091700,    , BEST,   0, 174N,  511W,  85,  973, HU,  64, NEQ, ␣
↪ 30,
   25,   20,   30, 1009,  210,  20, 100,   0,   L,   0,    ,   0,   0,
          TEDDY, D, 12, NEQ,  330,  300,  270,  300, genesis-num, 039,
AL, 20, 2020091706,    , BEST,   0, 180N,  520W,  85,  973, HU,  34, NEQ, ␣
↪220,
  100,   80,  170, 1009,  210,  20, 105,   0,   L,   0,    ,   0,   0,
          TEDDY, D, 12, NEQ,  330,  360,  300,  300, genesis-num, 039,
AL, 20, 2020091706,    , BEST,   0, 180N,  520W,  85,  973, HU,  50, NEQ, ␣
↪ 60,
   50,   50,   70, 1009,  210,  20, 105,   0,   L,   0,    ,   0,   0,
          TEDDY, D, 12, NEQ,  330,  360,  300,  300, genesis-num, 039,
AL, 20, 2020091706,    , BEST,   0, 180N,  520W,  85,  973, HU,  64, NEQ, ␣
↪ 30,
   25,   20,   30, 1009,  210,  20, 105,   0,   L,   0,    ,   0,   0,
          TEDDY, D, 12, NEQ,  330,  360,  300,  300, genesis-num, 039,
```

geoips.plugins.modules.sector_metadata_generators.bdeck_parser.**call**(*trackfile_name*)

TC deckfile parser for B-Deck files.

Each B-Deck file contains the full history of storm BEST tracks, one storm location per line. Example b-deck files are available in the GeoIPS repo.

Parameters

trackfile_name (*str*) – Path to bdeck file, with full 6 hourly storm track history, formatted as follows:

Returns

List of Dictionaries of storm metadata fields from each storm location

Return type

list

See also:

API Reference

Valid fields can be found in geoips.sector_utils.utils.SECTOR_INFO_ATTRS

geoips.plugins.modules.sector_metadata_generators.bdeck_parser.**get_final_storm_name_bd**

Get final storm name from full bdeck file.

geoips.plugins.modules.sector_metadata_generators.bdeck_parser.**get_invest_number_bdeck**

Get invest number from full bdeck file.

geoips.plugins.modules.sector_metadata_generators.bdeck_parser.**get_storm_start_datetim**

Get storm start datetime from full bdeck file.

geoips.plugins.modules.sector_metadata_generators.bdeck_parser.**get_storm_start_datetim**

Get storm start datetime from bdeck file name.

geoips.plugins.modules.sector_metadata_generators.bdeck_parser.**get_stormyear_from_bdec**

Get the storm year from the B-deck filename.

Parameters

bdeck_filename (*str*) –

- Path to deck file to search for storm year
- Must be of format: xxxxxYYYY.dat - pulls YYYY from filename based on location

Returns

Storm year

Return type

int

geoips.plugins.modules.sector_metadata_generators.bdeck_parser.**lat_to_dec**(*lat_str*)

Return decimal latitude based on N/S specified string.

geoips.plugins.modules.sector_metadata_generators.bdeck_parser.**lon_to_dec**(*lon_str*)

Return decimal longitude based on E/W specified string.

geoips.plugins.modules.sector_metadata_generators.bdeck_parser.**parse_bdeck_line**(*line*, *source_*
storm_y
fi-
nal_stor
in-
vest_nu
storm_s
orig-
i-
nal_stor
*parser_*

Retrieve the storm information from the current line from the deck file.

Parameters

line (*str*) – Current line from the deck file including all storm information

- AL, 20, 2020091618, , BEST, 0, 168N, 502W, 85, 973, HU, 64, NEQ, 30, 25, 0, 30, 1010, 180, 20, 105, 0, L, 0, , 0, 0, TEDDY, D, 12, NEQ, 300, 300, 240, 300, genesis-num, 039,
- AL, 20, 2020091700, , BEST, 0, 174N, 511W, 85, 973, HU, 34, NEQ, 220, 100, 80, 170, 1009, 210, 20, 100, 0, L, 0, , 0, 0, TEDDY, D, 12, NEQ, 330, 300, 270, 300, genesis-num, 039,
- AL, 20, 2020091700, , BEST, 0, 174N, 511W, 85, 973, HU, 50, NEQ, 60, 50, 50, 70, 1009, 210, 20, 100, 0, L, 0, , 0, 0, TEDDY, D, 12, NEQ, 330, 300, 270, 300, genesis-num, 039,

Returns
Dictionary of the fields from the current storm location from the deck file

Return type
dict

See also:

API Reference
Valid fields can be found in geoips.sector_utils.utils.SECTOR_INFO_ATTRS

geoips.plugins.modules.sector_metadata_generators.tc_sector_file_parser module

TC trackfile parser for flat text sectorfiles containing current active storms.

These files contain no storm history, only the currently active storm locations. Potentially useful for real-time processing.

10S JOSHUA 210120 1200 21.8S 78.1E SHEM 20 1007 12S ELOISE 210120 1800 15.6S 44.9E SHEM 35 1001 92S INVEST 210120 1800 14.9S 120.8E SHEM 30 1002 93S INVEST 210120 1800 12.6S 98.5E SHEM 30 1003

geoips.plugins.modules.sector_metadata_generators.tc_sector_file_parser.**NSEW_to_float**(

Convert lat/lon values with NSEW identifiers to positive or negative floats.

Parameters
lat_lon_val (*str*) – Latitude or longitude value as a string, with hemisphere specified by NSEW identifiers

Returns
Latitude or Longitude value as a float.

Return type
float

geoips.plugins.modules.sector_metadata_generators.tc_sector_file_parser.**call**(*trackfile_na*

TC trackfile parser for flat text sectorfiles containing current active storms.

These files contain no storm history, only the currently active storm locations. Potentially useful for real-time processing.

Parameters

trackfile_name (*str*) – Flat text sector file name containing all currently active storm locations, formatted as follows: * 10S JOSHUA 210120 1200 21.8S 78.1E SHEM 20 1007 * 12S ELOISE 210120 1800 15.6S 44.9E SHEM 35 1001 * 92S INVEST 210120 1800 14.9S 120.8E SHEM 30 1002 * 93S INVEST 210120 1800 12.6S 98.5E SHEM 30 1003

Returns

List of Dictionaries of storm metadata fields from each storm location in the flat text sector file

Return type

list

See also:

API Reference

Valid fields can be found in geoips.sector_utils.utils.SECTOR_INFO_ATTRS

geoips.plugins.modules.sector_metadata_generators.tc_sector_file_parser.**get_storm_year**

Ensure correct storm_year is applied.

For Southern Hemisphere storms that initiate late in the year, the storm year identifier is for the following year.

Parameters

- **storm_basin** (*str*) – basin of current storm, one of SH, AL, EP, CP, WP, IO
- **current_month** (*int*) – Current month of storm location
- **current_year** (*int*) – Current year of storm location

Returns

Storm year identifier. current year, unless SH storm later than June, then current year + 1

Return type

int

geoips.plugins.modules.sector_metadata_generators.tc_sector_file_parser.**parse_flat_sec**

Retrieve the storm information from the current line from the deck file.

Parameters
line (*str*) – Current line from the deck file including all storm information
* 10S JOSHUA 210120 1200 21.8S 78.1E SHEM 20 1007

Returns
Dictionary of the fields from the current storm location from the deck file

Return type
dict

See also:

API Reference
Valid fields can be found in geoips.sector_utils.utils.SECTOR_INFO_ATTRS

Module contents

GeoIPS trackfile parsers init file.

geoips.plugins.modules.sector_spec_generators package

Submodules

geoips.plugins.modules.sector_spec_generators.center_coordinates module

Generate standard pyresample area definitions given center coordinates.

Given desired center lat/lon, projection, resolution, and shape, return a valid pyresample area definition object.

geoips.plugins.modules.sector_spec_generators.center_coordinates.**call**(*area_id*, *long_description*, *clat*, *clon*, *projection*, *pixel_width*, *pixel_height*, *num_samples*, *num_lines*)

Create area definition using clat, clon, resolution, and shape.

geoips.plugins.modules.sector_spec_generators.center_coordinates.**set_clat_clon_proj_in**

Create standard proj4 dictionary from passed projection information.

Module contents

Geoips sector_spec_generators init file.

geoips.plugins.modules.title_formatters package

Submodules

geoips.plugins.modules.title_formatters.static_standard module

Standard GeoIPS static title production.

geoips.plugins.modules.title_formatters.static_standard.**call**(*area_def*, *xarray_obj*, *product_name_title*, *product_datatype_title=None*, *bg_xarray=None*, *bg_product_name_title=None*, *bg_datatype_title=None*, *title_copyright=None*)

Generate standard GeoIPS formatted title.

geoips.plugins.modules.title_formatters.tc_copyright module

Standard GeoIPS formatted titles for TC products, with copyright info.

geoips.plugins.modules.title_formatters.tc_copyright.**call**(*area_def*, *xarray_obj*, *product_name_title*, *product_datatype_title=None*, *bg_xarray=None*, *bg_product_name_title=None*, *bg_datatype_title=None*, *title_copyright=None*)

Create GeoIPS formatted title for TC products, with copyright info.

geoips.plugins.modules.title_formatters.tc_standard module

Standard GeoIPS formatted titles for tropical cyclone products.

geoips.plugins.modules.title_formatters.tc_standard.**call**(*area_def*, *xarray_obj*, *product_name_title*, *product_datatype_title=None*, *bg_xarray=None*, *bg_product_name_title=None*, *bg_datatype_title=None*, *title_copyright=None*)

Create standard GeoIPS formatted title for tropical cyclone products.

Module contents

GeoIPS title formatters init file.

Module contents

geoips.plugins.modules init file.

Module contents

Plugins init file.

geoips.sector_utils package

Submodules

geoips.sector_utils.estimate_area_extent module

Utility for estimating the area extent, used in pyresample area definitions.

geoips.sector_utils.estimate_area_extent.**center_longitude**(*min_longitude*, *max_longitude*)

Determine the center longitude based off longitude in either degW or degE.

Parameters

- **min_longitude** (*float*) – Min and Max Longitude (degrees West or East)
- **max_longitude** (*float*) – Min and Max Longitude (degrees West or East)

Returns
Center longitude in degrees West or East

Return type
float

geoips.sector_utils.estimate_area_extent.**convert_west2east**(*longitude*)

Convert Longitude from degrees West to degrees East, if applicable.

Parameters
longitude (*float*) – Longitude (degrees West or East)

Returns
Longitude in degrees East

Return type
float

geoips.sector_utils.estimate_area_extent.**esitmate_area_from_center**(*lat_0*, *lon_0*, *height*, *width*, *resolution*)

Estimate the area extent for use in the YAML area definition.

Parameters

- **lat_0** (*float*) – Center lat/lon values in degrees
- **lon_0** (*float*) – Center lat/lon values in degrees
- **height** (*int*) – Pixel dimensions
- **width** (*int*) – Pixel dimensions
- **resolution** (*int*) – Resolution in meters

Returns

Dictionary holding:

- lower_left_xy - list of projection x/y coordinates of lower left corner of lower left pixel
- upper_right_xy - list of projection x/y coordinates of upper right corner of upper right pixel
- height - number of grid rows
- width - number of grid columns
- lat_0 - Center latitude in degrees
- lon_0 - Center longitude in degrees

Return type
dict

geoips.sector_utils.estimate_area_extent.**estimate_area_extent**(*min_lat*, *min_lon*, *max_lat*, *max_lon*, *resolution*)

Estimate the area extent for use in the YAML area definition.

Parameters

- **min_lat** (*float*) – Min/Max lat/lon values in degrees
- **min_lon** (*float*) – Min/Max lat/lon values in degrees
- **max_lat** (*float*) – Min/Max lat/lon values in degrees
- **max_lon** (*float*) – Min/Max lat/lon values in degrees
- **resolution** (*float*) – Resolution in meters

Returns

Dictionary holding:

- lower_left_xy - list of projection x/y coordinates of lower left corner of lower left pixel
- upper_right_xy - list of projection x/y coordinates of upper right corner of upper right pixel
- height - number of grid rows
- width - number of grid columns
- lat_0 - Center latitude in degrees
- lon_0 - Center longitude in degrees

Return type
dict

geoips.sector_utils.estimate_area_extent.**generateMinMaxLatLong**(*lat_0*, *lon_0*, *height*, *width*, *resolution*)

Generate minimum and maximum latitude longitude pairs.

Min/max lat/lon based off the resolution and height/width provided.

Parameters

- **lat_0** (*float*) – Pair of latitude and longitude coordinates in degrees
- **lon_0** (*float*) – Pair of latitude and longitude coordinates in degrees
- **height** (*int*) – Represents pixel dimensions and resolution of image in meters
- **width** (*int*) – Represents pixel dimensions and resolution of image in meters
- **resolution** (*int*) – Represents pixel dimensions and resolution of image in meters

Returns
min_lat, max_lat, min_lon, max_lon

Return type
list of floats

geoips.sector_utils.estimate_area_extent.**haversine_distance**(*lat1*, *lon1*, *lat2*, *lon2*)

Calculate the distance between two latitude and longitude points.

Uses the haversine formula.

Parameters

- **lat1** (*float*) – Pair of latitude and longitude coordinates in degrees
- **lon1** (*float*) – Pair of latitude and longitude coordinates in degrees
- **lat2** (*float*) – Pair of latitude and longitude coordinates in degrees
- **lon2** (*float*) – Pair of latitude and longitude coordinates in degrees

Returns
Distance in meters between two coordinates

Return type
float

geoips.sector_utils.overpass_predictor module

Overpass predictor, based on Two Line Element files.

geoips.sector_utils.overpass_predictor.**calculate_overpass**(*tle*, *observer_lat*, *observer_lon*, *date*, *satellite_name*)

Calculate next overpass for a satellite at an observer location and time.

Parameters

- **tle** (*ephem.EarthSatellite*) – tle for satellite
- **observer_lat** (*float*) – observer latitude
- **observer_lon** (*float*) – observer longitude
- **date** (*datetime.datetime*) – start time for next overpass
- **satellite_name** (*str*) – name of satellite

Returns
next overpass information

Return type
dict

geoips.sector_utils.overpass_predictor.**check_tle_name_to_passed_names**(*tle_name*, *satellite_names_list*)

Check if the satellite name in the TLE files is in the satellite names list.

Satellite names in the TLE files may be longer than the names passed to the overpass predictor For example, a user might request 'GCOM-W1', and the name in the TLE file is 'GCOM-W1 (SHIZUKU)'.

Parameters

- **tle_name** (*str*) – satellite name read from the TLE file
- **satellite_names_list** (*list of str*) – list of user specified satellites to read from TLE file

Returns
True if tle_name is in the passed satellite names list

Return type
bool

geoips.sector_utils.overpass_predictor.**floor_minute**(*datetime_obj*)

Remove seconds and microseconds from datetime object.

Parameters
datetime_obj (*datetime.datetime*) – datetime

Returns
datetime with no seconds/microseconds

Return type
datetime.datetime

geoips.sector_utils.overpass_predictor.**predict_overpass_area_def**(*tlefile*, *area_definition*, *satellite_list*, *start_datetime*, *check_midpoints=False*)

Predict satellite overpass for an area_definition.

Parameters

- **tlefile** (*str*) – file path of TLE
- **area_definition** (*pyresample AreaDefinition*) – pyresample area definition
- **satellite_list** (*list*) – list of satellites to predict the overpass times
- **start_datetime** (*datetime.datetime*) – start time to find the next available overpass
- **check_midpoints** (*bool*) – check mid points of area definition for additional overpassses

Returns
dictionary holding next satellite overpass estimates (sorted by satellite -> overpass info)

Return type
dict

geoips.sector_utils.overpass_predictor.**predict_overpass_yaml**(*tlefile*, *sectorfile*, *sector_list*, *satellite_list*, *start_datetime*, *check_midpoints=False*)

Predict satellite overpass for sectors from a given yaml sector file.

Parameters

- **tlefile** (*str*) – file path of TLE
- **sectorfile** (*str*) – file path of sectorfile
- **sector_list** (*list*) – list of sectors held within the sectorfile
- **satellite_list** (*list*) – list of satellites to predict the overpass times
- **start_datetime** (*datetime.datetime*) – start time to find the next available overpass
- **check_midpoints** (*bool*) – check mid points of area definition for additional overpassses

Returns
dictionary holding next satellite overpass estimates for sectors (sorted by sector -> satellite -> overpass info)

Return type
dict

geoips.sector_utils.overpass_predictor.**predict_satellite_overpass**(*tlefile*, *satellite_name*, *satellite_tle*, *area_def*, *start_datetime*, *check_midpoints=False*)

Estimate next satellite overpass information with ephem.

Parameters

- **tlefile** (*str*) – file path of TLE
- **satellite_name** (*str*) – name of satellite
- **satellite_tle** (*dict*) – dictionary holding satellite tle line1 and line2 data

- **area_def** (*pyresample AreaDefinition*) – area definition
- **start_datetime** (*datetime.datetime*) – start time to find the next available overpass
- **check_midpoints** (*bool*) – check mid points of area definition for additional overpassses

Returns
dictionary holding next overpass information

Return type
dict

geoips.sector_utils.overpass_predictor.**read_satellite_tle**(*tlefile*, *satellite_list*)

Open and extract satellite infromation from TLE file.

Parameters

- **tlefile** (*str*) – file path of TLE
- **satellite_list** (*list of str*) – list of satellites to read from TLE

Returns
satellite TLE data

Return type
dict

geoips.sector_utils.projections module

Projection information for setting up pyresample area definitions.

geoips.sector_utils.projections.**get_projection**(*name*)

Get a dictionary of projection names containing the specified keys.

Dictionary keys:

- name: the basemap projection short name
- p4name: the Proj4 projection name
- longname: a long name describing the projection
- type: an integer indicating how the projection must be set up

The type field tells the program which arguments will be useful to a given projection

- Can use corner lats and lons or center lats and lons with width and height
- Ignores corner lats and lons and width/height arguments. Uses center lat/lon

- Can use corner lats and lons or corner coordinates in the local projection space, but ignores all other location parameters.

geoips.sector_utils.tc_tracks module

Modules to access TC tracks, based on locations found in the deck files.

geoips.sector_utils.tc_tracks.**create_tc_sector_info_dict**(*clat*, *clon*, *synoptic_time*, *storm_year*, *storm_basin*, *storm_num*, *aid_type=None*, *storm_name=None*, *final_storm_name=None*, *deck_line=None*, *source_sector_file=None*, *pressure=None*, *vmax=None*)

Create storm info dictionary from items.

Parameters

- **clat** (*float*) – center latitude of storm
- **clon** (*float*) – center longitude of storm
- **synoptic_time** (*datetime.datetime*) – time of storm location
- **storm_year** (*int*) – 4 digit year of storm
- **storm_basin** (*str*) – 2 digit basin identifier
- **storm_num** (*int*) – 2 digit storm number
- **aid_type** (*str, default=None*) – type of TC aid (BEST, MBAM, etc)
- **storm_name** (*str, default=None*) – Common name of storm
- **final_storm_name** (*str, default=None*) – Final name found throughout entire track file (ie, if reprocessing, will ensure early storm locations are identified with final storm name)
- **deck_line** (*str, default=None*) – source deck line for storm information
- **pressure** (*float, default=None*) – minimum pressure
- **vmax** (*float, default=None*) – maximum wind speed

Returns
fields – Dictionary of sector information, as passed into function.

Return type
dict

geoips.sector_utils.tc_tracks.**get_tc_area_id**(*fields*, *finalstormname*, *tcyear*)

Get TC area_id from fields, to be used as pyresample AreaDefinition area_id.

Will be of form: * tcYYYYBBNNname (ie, tc2016io01one)

geoips.sector_utils.tc_tracks.**get_tc_long_description**(*area_id*, *fields*)

Return long_description of TC sector.

This is commonly used as the long name/description on the pyresample AreaDefinition.

geoips.sector_utils.tc_tracks.**interpolate_storm_location**(*interp_dt*, *longitudes*, *latitudes*, *synoptic_times*)

Interpolate the storm location at a specific time.

Based on a list of known locations and times

geoips.sector_utils.tc_tracks.**set_tc_area_def**(*fields*, *tcyear=None*, *finalstormname=None*, *source_sector_file=None*, *clat=None*, *clon=None*, *tc_spec_template='tc_web'*, *aid_type=None*)

Set the TC area definition, using specified arguments.

Parameters

- **fields** (*dict*) – Dictionary of TC sector_info fields (clat, clon, storm name, etc) Valid fields can be found in geoips.sector_utils.utils.SECTOR_INFO_ATTRS
- **tcyear** (*int, default=None*) – Passed tcyear - since current year may not match tcyear for SHEM storms
- **finalstormname** (*str, default=None*) – finalstormname allows re-processed storms to go in final storm directory
- **source_sector_file** (*str, default=None*) – attach source_sector_file to area_definition if known
- **clat** (*float, default=None*) – specify clat/clon separately from that found in 'fields'
- **clon** (*float, default=None*) – specify clat/clon separately from that found in 'fields'

- **tc_spec_template** (*str, default="tc_web"*) – Path to template YAML file to use when setting up area definition.
- **aid_type** (*str, default=None*) – type of TC aid (BEST, MBAM, etc)

Returns
pyresample AreaDefinition object with specified parameters.

Return type
pyresample.AreaDefinition

geoips.sector_utils.tc_tracks.**trackfile_to_area_defs**(*trackfile_name*, *trackfile_parser='bdeck_parser'*, *tc_spec_template=None*)

Get TC area definitions for the specified text trackfile.

Limit to optionally specified trackfile_sectorlist

Parameters

- **trackfile** (*str*) – Full path to trackfile, convert each line into a separate area_def
- **trackfile_parser** (*str*) – Parser to use from plugins.modules.sector_metadata_generators on trackfiles

Returns
List of pyresample AreaDefinition objects

Return type
list

geoips.sector_utils.tc_tracks_database module

Utilities for creating a database of tropical cyclone tracks.

geoips.sector_utils.tc_tracks_database.**check_db**(*filenames=None*, *process=False*)

Check TC database for passed filenames.

filenames is a list of filenames and directories. if a list element is a string directory name, it expands to list of files in dir.

geoips.sector_utils.tc_tracks_database.**get_all_storms_from_db**(*start_datetime*, *end_datetime*, *tc_spec_template=None*, *trackfile_parser=None*, *include_track_files=False*)

Get all entries from all storms within a specific range of time from the TC DB.

Parameters

- **start_datetime** (*datetime.datetime*) – Start time of desired range
- **end_datetime** (*datetime.datetime*) – End time of desired range

Returns

List of pyresample Area Definitions, each storm location that falls within the desired time range.

Return type

list of pyresample Area Definitions

Examples

```
>>> startdt = datetime.strptime('20200216', '%Y%m%d')
>>> enddt = datetime.strptime('20200217', '%Y%m%d')
>>> get_storm_from_db(startdt, enddt)
```

geoips.sector_utils.tc_tracks_database.**open_tc_db**(*dbname='/users/surratt/geoips/outdirs/longtern*

Open the TC Decks Database, create it if it doesn't exist.

geoips.sector_utils.tc_tracks_database.**reprocess_storm**(*tc_trackfilename*)

Reprocess storm tc_trackfilename, using info in TC tracks database.

geoips.sector_utils.tc_tracks_database.**update_fields**(*tc_trackfilename*, *cc*, *conn*, *process=False*)

Update fields in TC track database with passed tc_trackfilename.

geoips.sector_utils.utils module

Utilities for working with dynamic sector specifications.

geoips.sector_utils.utils.**check_center_coverage**(*xarray_obj*, *area_def*, *varlist*, *covg_varname=None*, *covg_varlist=None*, *width_degrees=8*, *height_degrees=8*, *verbose=False*, *hours_before_sector_time=18*, *hours_after_sector_time=6*)

Check if there is any data covering the center of the sector.

Do not provide any longitude padding for coverage check sectoring - we want to see if there is any data within the exact center box, not within +- 3 degrees of the center box.

geoips.sector_utils.utils.**copy_sector_info**(*src_area_def*, *dest_area_def*)

Copy sector info from src_area_def to dest_area_def.

geoips.sector_utils.utils.**create_areadefinition_from_yaml**(*yamlfile*, *sector*)

Take a YAML with misc metadata and create a pyresample areadefinition.

Misc. metadata will be parsed from the YAML file and manually added to the areadefinition

Parameters

- **yamlfile** (*str*) – full path to YAML area definition file
- **sector** (*str*) – name of sector

Returns

pyresample AreaDefinition based on YAML specification.

Return type

pyresample.AreaDefinition

geoips.sector_utils.utils.**filter_area_defs_actual_time**(*area_defs*, *actual_datetime*)

Filter list of area_defs to only include the passed actual_datetime.

geoips.sector_utils.utils.**get_lat_center**(*lats*)

Return the center longitude point from lats array.

geoips.sector_utils.utils.**get_lon_center**(*lons*)

Return the center longitude point from lons array.

geoips.sector_utils.utils.**get_max_lat**(*lats*)

Get maximum latitude from array of latitudes.

Parameters

lats (*numpy.ndarray*) – numpy MaskedArray of latitudes

Returns

Maximum latitude, between -90 and 90

Return type

float

geoips.sector_utils.utils.**get_max_lon**(*lons*)

Get maximum longitude from array of longitudes, handling date line.

Parameters

lons (*numpy.ndarray*) – numpy MaskedArray of longitudes

Returns

Maximum longitude, between -180 and 180

Return type
float

geoips.sector_utils.utils.**get_min_lat**(*lats*)

Get minimum latitude from array of latitudes.

Parameters
lats (*numpy.ndarray*) – numpy MaskedArray of latitudes

Returns
Minimum latitude, between -90 and 90

Return type
float

geoips.sector_utils.utils.**get_min_lon**(*lons*)

Get minimum longitude from array of longitudes, handling date line.

Parameters
lons (*numpy.ndarray*) – numpy MaskedArray of longitudes

Returns
Minimum longitude, between -180 and 180

Return type
float

geoips.sector_utils.utils.**get_sectors_from_yamls**(*sector_list*)

Get AreaDefinition objects with custom "sector_info" dictionary.

Based on YAML area definition contained in "sectorfnames" files.

Parameters
sector_list (*list of str*) – list of strings of desired sector names to retrieve from YAML files

Returns
List of pyresample AreaDefinition objects, with arbitrary additional YAML entries added as attributes to each area def (this is to allow specifying "sector_info" metadata dictionary within the YAML file)

Return type
list

geoips.sector_utils.utils.**get_static_area_defs_for_xarray**(*xarray_obj*, *sectorlist*)

Get all STATIC area definitions for the current xarray object.

Filter based on requested sectors.

Parameters

- **xarray_obj** (*xarray.Dataset*) – xarray Dataset to which we are assigning area_defs
- **sectorlist** (*list of str*) – list of sector names

Returns
List of pyresample AreaDefinition objects

Return type
list of pyresample.AreaDefinition

geoips.sector_utils.utils.**get_tc_area_defs_for_xarray**(*xarray_obj*, *tcdb_sector_list=None*, *tc_spec_template=None*, *trackfile_parser=None*, *hours_before_sector_time=18*, *hours_after_sector_time=6*, *aid_type=None*)

Get all TC area definitions for the current xarray object, and requested sectors.

Parameters

- **xarray_obj** (*xarray.Dataset*) – xarray Dataset to which we are assigning area_defs
- **tcdb_sector_list** (*list of str, default=None*) –
  - list of sector names to process, of format: tc2020io01amphan.
  - If None, or 'all' contained in list, process all matching TC sectors.
- **actual_datetime** (*datetime.datetime, default=None*) – Optional datetime to match for dynamic sectors
- **var_for_coverage** (*str*) – Default None, optional variable to sector to check exact time
- **hours_before_sector_time** (*float, default=18*) – hours to look before sector time
- **hours_after_sector_time** (*float, default=6*) – hours to look after sector time
- **aid_type** (*str, default=None*) – string to look for in "aid_type" TC deck file field for inclusion

Returns
List of pyresample AreaDefinition objects required for passed xarray

Return type
list of pyresample AreaDefinition

geoips.sector_utils.utils.**get_trackfile_area_defs**(*trackfiles*, *trackfile_parser*, *trackfile_sectorlist=None*, *tc_spec_template=None*, *aid_type=None*, *start_datetime=None*, *end_datetime=None*)

Get all TC area definitions for the current xarray object, and requested sectors.

Parameters

- **trackfiles** (*list*) – List of trackfiles to convert into area_defs
- **trackfile_parser** (*str*) – Parser to use from plugins.modules.sector_metadata_generators on trackfiles
- **str** (*trackfile_sectorlist list of*) –
  - list of sector names to process, of format: tc2020io01amphan.
  - If None, or 'all' contained in list, process all matching TC sectors.
- **default=None** –
  - list of sector names to process, of format: tc2020io01amphan.
  - If None, or 'all' contained in list, process all matching TC sectors.
- **aid_type** (*str, default=None*) – If specified, string to look for in "aid_type" TC deck file field for inclusion

Returns

List of pyresample AreaDefinition objects

Return type

list of pyresample.AreaDefinition

geoips.sector_utils.utils.**is_dynamic_sector**(*area_def*)

Determine if the AreaDefinition object is a dynamic region of interest.

Parameters

area_def (*pyresample.AreaDefinition*) – pyresample AreaDefinition object specifying region of interest

Returns

- True if area_def.sector_start_datetime exists and is not None,
- False otherwise

Return type

bool

geoips.sector_utils.utils.**is_requested_aid_type**(*area_def*, *aid_type=None*)

Return True if passed area_def is of requested aid_type.

geoips.sector_utils.utils.**is_sector_type**(*area_def*, *sector_type_str*)

Determine if the type of area_def sector is as specified in passed sector_type.

Parameters

- **area_def** (*pyresample.AreaDefinition*) – pyresample AreaDefinition object specifying region of interest
- **sector_type_str** (*str*) –
  - String specifying the type of sector, must match 'sector_type' attribute on AreaDefinition object
  - currently one of 'tc', 'pyrocb', 'volcano', 'atmosriver' 'static'

Returns

True if area_def.sector_type == 'sector_type', False otherwise

Return type

bool

geoips.sector_utils.utils.**remove_duplicate_storm_positions**(*area_defs*, *aid_type=None*)

Remove duplicate storm positions from passed list of area_defs.

Uses "is_requested_aid_type" and "storm_locations_match" utilities.

geoips.sector_utils.utils.**set_tc_coverage_check_area_def**(*area_def*, *width_degrees=8*, *height_degrees=8*)

Set the area definition for checking coverage for TC overpasses.

Take a small box around the center of the storm to evaluate coverage, rather than the entire image.

Parameters

area_def (*pyresample.AreaDefinition*) – original area definition

Returns

pyresample AreaDefinition pertaining to the region for plotting

Return type

pyresample.AreaDefinition

geoips.sector_utils.utils.**set_text_area_def**(*xarray_obj*, *area_def*)

Set the area definition for text files.

This uses raw sectored data, not interpolated.

Parameters

- **xarray_obj** (*xarray.Dataset*) – xarray dataset
- **area_def** (*pyresample.AreaDefinition*) – original area definition

Returns
pyresample AreaDefinition pertaining to the region for generating text file

Return type
pyresample.AreaDefinition

geoips.sector_utils.utils.**storm_locations_match**(*area_def*, *other_area_def*)

Return True if passed pyresample AreaDefinitions are the same location.

Match if center lat, center lon, storm year, storm basin, and synoptic time all match.

geoips.sector_utils.yaml_utils module

Utilities for working with YAML sector specifications.

geoips.sector_utils.yaml_utils.**add_description_to_yamldict**(*yaml_dict*, *sectorname*, *sector_type*, *sector_start_datetime=None*, *info_dict=None*)

Add passed sector description information to passed YAML dictionary.

geoips.sector_utils.yaml_utils.**add_dynamic_datetime_to_yamldict**(*yaml_dict*, *sectorname*, *sector_start_datetime*, *sector_end_datetime*)

Add passed dynamic datetime info to passed YAML dictionary.

geoips.sector_utils.yaml_utils.**add_projection_to_yamldict**(*yaml_dict*, *sectorname*, *center_lat*, *center_lon*, *center_x=0*, *center_y=0*, *template_yaml=None*)

Add projection information to YAML dictionary.

geoips.sector_utils.yaml_utils.**add_sectorinfo_to_yamldict**(*yaml_dict*, *sectorname*, *sector_info_dict*)

Add sector_info dictionary to YAML dictionary.

geoips.sector_utils.yaml_utils.**area_def_to_yamldict**(*area_def*)

Convert passed pyresample AreaDefinition to a valid YAML dictionary.

geoips.sector_utils.yaml_utils.**area_def_to_yamlfile**(*area_def*, *out_fname*)

Write pyresample AreaDefinition out as a valid YAML dictionary.

geoips.sector_utils.yaml_utils.**write_yamldict**(*yamldict*, *out_fname*, *force=False*)

Write yamldict to out_fname.

Parameters

- **yamldict** (*dict*) – Dictionary to write out to YAML file
- **out_fname** (*str*) – Output filename to write YAML dict to
- **force** (*bool, default=False*) – If True, overwrite existing file.

Returns

Path to output file if successfully produced

Return type

str

Module contents

Geoips sector utils init file.

geoips.utils package

Submodules

geoips.utils.decorators module

GeoIPS decorators module.

class geoips.utils.decorators.**deprecated**(*replacement=None*)

Bases: object

A decorator that deprecates a function.

When applied to a function, will cause that function to raise a DeprecationWarning when called.

geoips.utils.decorators.**developmental**(*func*)

Mark an interfaces as developmental.

When applied to a function, will prepend a "developmental" message to the beginning of that function's docstring.

geoips.utils.memusg module

Utilities for tracking and monitoring memory and resource usage.

geoips.utils.memusg.**print_mem_usage**(*logstr=''*, *verbose=False*)

Print memory usage to LOG.info.

- By default include psutil output.
- If verbose is True, include output from both psutil and resource packages.

geoips.utils.memusg.**print_resource_usage**(*logstr=''*)

Print verbose resource usage, using "resource" package.

Module contents

Geoips utilities init file.

geoips.xarray_utils package

Submodules

geoips.xarray_utils.data module

Utilities for manipulating xarray Datasets and DataArrays.

geoips.xarray_utils.data.**get_lat_lon_points**(*checklat*, *checklon*, *diff*, *sect_xarray*, *varname*, *drop=False*)

Pull values from xarray Datasets in specified geographic location.

Return points a given distance around a specified lat/lon location, from xarray Datasets.

Parameters

- **checklat** (*float*) – latitude of interest
- **checklon** (*float*) – longitude of interest
- **diff** (*float*) – check +- diff of latitude and longitude
- **sect_xarray** (*Dataset*) – xarray dataset containing 'latitude' 'longitude' and varname variables
- **varname** (*str*) – variable name of data array to use for returning data values

Returns

- min value in range
- max value in range
- and number of points in range

Return type
float, float, int

geoips.xarray_utils.data.**get_lat_lon_points_numpy**(*checklat*, *checklon*, *diff*, *lat_array*, *lon_array*, *data_array*)

Pull values from numpy arrays in specified geographic location.

Return points a given distance around a specified lat/lon location, from numpy arrays.

Parameters

- **checklat** (*float*) – latitude of interest
- **checklon** (*float*) – longitude of interest
- **diff** (*float*) – check +- diff of latitude and longitude
- **lat_array** (*ndarray*) – numpy ndarray of latitude locations - same shape as lon_array and data_array
- **lon_array** (*ndarray*) – numpy ndarray of longitude locations - same shape as lat_array and data_array
- **data_array** (*ndarray*) – numpy ndarray data values - same shape as lat_array and lon_array

Returns

- min value in range
- max value in range
- and number of points in range

Return type
float, float, int

geoips.xarray_utils.data.**get_sectored_xarrays**(*xobjs*, *area_def*, *varlist*, *get_bg_xarrays=False*, *check_center=True*, *drop=False*)

Get all xarray objects sectored to area_def.

Return primary dataset, as well as VIS/IR overlay datasets.

geoips.xarray_utils.data.**get_vis_ir_bg**(*sect_xarray*)

Find matching vis/ir background for data in sect_xarray.

geoips.xarray_utils.data.**sector_xarray_dataset**(*full_xarray*, *area_def*, *varnames*, *lon_pad=3*, *lat_pad=0*, *verbose=False*, *hours_before_sector_time=18*, *hours_after_sector_time=6*, *drop=False*)

Use the xarray to appropriately sector out data by lat/lon and time.

geoips.xarray_utils.data.**sector_xarray_spatial**(*full_xarray*, *extent_lonlat*, *varnames*, *lon_pad=3*, *lat_pad=0*, *verbose=False*, *drop=False*)

Sector an xarray object spatially. If full_xarray is None, return None.

Parameters

- **full_xarray** (*xarray.Dataset*) – xarray object to sector spatially
- **extent_lonlat** (*list of float*) – Area to sector: [MINLON, MINLAT, MAXLON, MAXLAT]
- **varnames** (*list of str*) – list of variable names that should be sectored based on 'time'
- **drop** (*bool*) – Specify whether to remove points with no coverage (rather than masking)

Returns

- if full_xarray is None, return None,
- else return resulting xarray Dataset.

Return type
xarray.Dataset

geoips.xarray_utils.data.**sector_xarray_temporal**(*full_xarray*, *mindt*, *maxdt*, *varnames*, *verbose=False*, *drop=False*)

Sector an xarray object temporally. If full_xarray is None, return None.

Parameters

- **full_xarray** (*xarray.Dataset*) – xarray object to sector temporally
- **mindt** (*datetime.datetime*) – minimum datetime of desired data
- **maxdt** (*datetime.datetime*) – maximum datetime of desired data
- **varnames** (*list of str*) – list of variable names that should be sectored based on 'time', mindt, maxdt

Returns

- if full_xarray is None, return None
- return full original xarray object if 'time' is not included in varnames list
- else, return sectored xarray object with only the desired times, specified by mindt and maxdt

Return type
xarray Dataset, or None

geoips.xarray_utils.data.**sector_xarrays**(*xobjs*, *area_def*, *varlist*, *verbose=False*, *hours_before_sector_time=18*, *hours_after_sector_time=6*, *check_center=True*, *drop=False*, *lon_pad=3*, *lat_pad=0*)

Return list of sectored xarray objects.

geoips.xarray_utils.time module

Utils to handle time stamp information within xarray objects.

geoips.xarray_utils.time.**get_datetime_from_datetime64**(*dt64*)

Get a python datetime object from a numpy datetime64 object.

Parameters
dt64 (*numpy.datetime64'*) – numpy.datetime64 object

Returns
Python datetime object

Return type
datetime.datetime

Notes

Backwards compatible with numpy versions

geoips.xarray_utils.time.**get_max_from_xarray_time**(*xarray_obj*, *varname*)

Get the maximum time as a datetime object from xarray object.

Parameters

- **xarray_obj** (*xarray.Dataset or xarray.DataArray*) – xarray object from which to extract the maximum time
- **varname** (*str*) – Timestamp variable name from which to extract the maximum time

Returns
Python datetime.datetime object representing maximum time of the Dataset or DataArray

Return type
datetime.datetime

geoips.xarray_utils.time.**get_min_from_xarray_time**(*xarray_obj*, *varname*)

Get the minimum time as a datetime object from xarray object.

Parameters

- **xarray_obj** (*xarray.Dataset or xarray.DataArray*) – xarray object from which to extract the minimum time
- **varname** (*str*) – Timestamp variable name from which to extract the minimum time

Returns
Python datetime.datetime object representing minimum time of the Dataset or DataArray

Return type
datetime.datetime

geoips.xarray_utils.time.**get_posix_from_datetime**(*dt*)

Return the POSIX timestamp in seconds.

Parameters
dt (*datetime.datetime*) – datetime object to convert to posix timestamp

Returns
representing seconds since 1 January 1970 at 00Z (epoch seconds)

Return type
long

Module contents

Geoips xarray utils init file.

6.1.2 Submodules

6.1.3 geoips.cli module

GeoIPS

The Geolocated Information Processing System (GeoIPS) is a generalized processing system, providing a collection of algorithm and product implementations facilitating consistent and reliable application of specific products across a variety of sensors and data types.

GeoIPS acts as a toolbox for internal GeoIPS-based product development - all modules are expected to have simple inputs and outputs (Python numpy or dask arrays or xarrays, dictionaries, strings, lists), to enable portability and simplified interfacing between modules.

class geoips.cli.**RawDescriptionArgumentDefaultsHelpFormatter**(*prog*, *indent_increment=2*, *max_help_position=24*, *width=None*)

Bases: ArgumentDefaultsHelpFormatter, RawDescriptionHelpFormatter

Compound formatter class for user-readable help.

- preserves the raw description formatting
- adds defaults to helps.

geoips.cli.**add_list_interface_parser**(*subparsers*, *name*, *aliases=None*)

Add list interface parser.

geoips.cli.**formclass**

alias of *RawDescriptionArgumentDefaultsHelpFormatter*

geoips.cli.**get_interface**(*name*)

Get interface.

geoips.cli.**list_dev_interfaces**()

Return a list of all developmental interfaces.

geoips.cli.**list_interface_plugins**(*interface_name*)

List interface plugins.

geoips.cli.**list_interfaces**(*dev=False*)

List interfaces.

geoips.cli.**main**()

Command line interface main function.

geoips.cli.**print_table**(*title*, *headings*, *rows*)

Print a column formatted table.

Parameters

- **title** (*str*) – A title for the table
- **headings** (*list of str*) – A list of strings to use as column headings
- **rows** (*list of tupl of str*) – A list of equal-length tuples

6.1.4 geoips.compare_outputs module

Test script for representative product comparisons.

geoips.compare_outputs.**compare_outputs**(*compare_path*, *output_products*, *test_product_func=None*)

Compare the "correct" imagery found the list of current output_products.

Compares files produced in the current processing run with the list of "correct" files contained in "compare_path".

Parameters

- **compare_path** (*str*) – Path to directory of "correct" products - filenames must match output_products
- **output_products** (*list of str*) – List of strings of current output products, to compare with products in compare_path
- **test_product_func** (*function, default=None*) – Alternative function to be used for testing output product
  - Call signature must be:
    - output_product, compare_product, goodcomps, badcomps, compare_strings
  - Return must be:
    - goodcomps, badcomps, compare_strings
  - If None, use geoips.compare_outputs.test_product)

Returns
Binary code: 0 if all comparisons were completed successfully.

Return type
int

geoips.compare_outputs.**geoips_netcdf_match**(*output_product*, *compare_product*)

Check if two geoips formatted netcdf files match.

Parameters

- **output_product** (*str*) – Full path to current output product
- **compare_product** (*str*) – Full path to comparison product

Returns
Return True if products match, False if they differ

Return type
bool

geoips.compare_outputs.**geotiffs_match**(*output_product*, *compare_product*)

Use diff system command to compare currently produced image to correct image.

Parameters

- **output_product** (*str*) – Full path to current output product
- **compare_product** (*str*) – Full path to comparison product

Returns
Return True if images match, False if they differ

Return type
bool

geoips.compare_outputs.**get_out_diff_fname**(*compare_product*, *output_product*, *ext=None*, *flag=None*)

Obtain the filename for output and comparison product diff.

Parameters

- **compare_product** (*str*) – Full path to product filename in the comparison directory
- **output_product** (*str*) – Full path to product filename in the current output directory
- **ext** (*str, default=None*) – Extension to use as an alternative to the original file extension
- **flag** (*str, default=None*) – Additional identifying string to include in output diff filename

Returns
out_diff_fname – Full path to output diff file.

Return type
str

geoips.compare_outputs.**gunzip_product**(*fname*)

Gunzip file fname.

Parameters
fname (*str*) – File to gunzip.

Returns
Filename after gunzipping

Return type
str

geoips.compare_outputs.**gzip_product**(*fname*)

Gzip file fname.

Parameters
fname (*str*) – File to gzip.

Returns
Filename after gzipping

Return type
str

geoips.compare_outputs.**images_match**(*output_product*, *compare_product*, *fuzz='5%'*)

Use imagemagick compare system command to compare two images.

Parameters

- **output_product** (*str*) – Current output product
- **compare_product** (*str*) – Path to comparison product
- **fuzz** (*str, optional*) – "fuzz" argument to pass to compare - larger "fuzz" factor to make comparison less strict, by default 5%.

Returns
Return True if images match, False if they differ

Return type
bool

geoips.compare_outputs.**is_geoips_netcdf**(*fname*)

Check if fname is a geoips formatted netcdf file.

Parameters
fname (*str*) – Name of file to check.

Returns
True if it is a geoips netcdf file, False otherwise.

Return type
bool

geoips.compare_outputs.**is_geotiff**(*fname*)

Determine if fname is a geotiff file.

Parameters
fname (*str*) – Name of file to check.

Returns
True if it is a geotiff file, False otherwise.

Return type
bool

geoips.compare_outputs.**is_gz**(*fname*)

Check if fname is a gzip file.

Parameters
fname (*str*) – Name of file to check.

Returns
True if it is a gz file, False otherwise.

Return type
bool

geoips.compare_outputs.**is_image**(*fname*)

Determine if fname is an image file.

Parameters
fname (*str*) – Name of file to check.

Returns
True if it is an image file, False otherwise.

Return type
bool

geoips.compare_outputs.**is_text**(*fname*)

Check if fname is a text file.

Parameters
fname (*str*) – Name of file to check.

Returns
True if it is a text file, False otherwise.

Return type
bool

geoips.compare_outputs.**print_gunzip_to_file**(*fobj*, *gunzip_fname*)

Write the command to gunzip the passed "gunzip_fname" to file.

Writes to the currently open file object, if required.

geoips.compare_outputs.**print_gzip_to_file**(*fobj*, *gzip_fname*)

Write the command to gzip the passed "gzip_fname" to file.

Writes to the currently open file object, if required.

geoips.compare_outputs.**test_product**(*output_product*, *compare_product*, *goodcomps*, *badcomps*, *compare_strings*)

Test output_product against "good" product stored in "compare_path".

Parameters

- **output_product** (*str*) –
  - Full path to current output product
- **compare_product** (*str*) –
  - Full path to "good" comparison product
- **goodcomps** (*list of str*) –
  - List of full paths to all "good" successful comparisons (output and compare images match)
  - Each str is prepended with a "compare_string" tag to identify which comparison type was performed.
- **badcomps** (*list of str*) –
  - List of full paths to all "bad" unsuccessful comparisons (output and compare images differ)
  - Each str is prepended with a "compare_string" tag to identify which comparison type was performed.
- **compare_strings** (*list of str*) –
  - List of all comparison "tags" included in goodcomps and badcomps lists.
  - This list is used to remove the comparison tags from goodcomps and badcomps to retrieve only the file path.

Returns

- **goodcomps** (*list of str*) – All current good comparisons appended to the list passed in.
- **badcomps** (*list of str*) – All current bad comparisons appended to the list passed in.
- **compare_strings** (*list of str*) – All current comparison "tags" added to the list passed in.

Raises

TypeError – Raised when current output product does not have an associated comparison test defined.

geoips.compare_outputs.**text_match**(*output_product*, *compare_product*)

Check if two text files match.

Parameters

- **output_product** (*str*) – Full path to current output product
- **compare_product** (*str*) – Full path to "good" comparison product

Returns

Return True if products match, False if they differ

Return type

bool

6.1.5 geoips.errors module

GeoIPS error module.

exception geoips.errors.**CoverageError**

Bases: Exception

Raise exception on data coverage error.

exception geoips.errors.**EntryPointError**

Bases: Exception

Exception to be raised when an entry-point cannot be found.

exception geoips.errors.**PluginError**

Bases: Exception

Exception to be raised when there is an error in a plugin module.

6.1.6 geoips.geoips_utils module

General high level utilities for geoips processing.

geoips.geoips_utils.**copy_standard_metadata**(*orig_xarray*, *dest_xarray*, *extra_attrs=None*, *force=True*)

Copy standard metadata from orig_xarray to dest_xarray.

Parameters

- **orig_xarray** (*xarray.Dataset*) – Original xarray to copy attributes from
- **dest_xarray** (*xarray.Dataset*) – Destination xarray to copy attributes to

- **extra_attrs** (*list of str, optional*) – Additional attributes to copy, beyond the standard metadata, by default None
- **force** (*bool, optional*) – If force is True, overwrite existing attributes, by default True

Returns
dest_xarray with standard metadata copied in place from orig_xarray.

Return type
xarray.Dataset

geoips.geoips_utils.**deprecation**(*message*)

Print a deprecation warning during runtime.

geoips.geoips_utils.**find_all_txt_plugins**(*subdir=''*)

Find all txt plugins in registered plugin packages.

Search the `plugins` directory of each registered plugin package for files ending in `.txt`. Return list of files

geoips.geoips_utils.**find_ascii_palette**(*name*)

Find ASCII palette named "name".

Search the plugins/txt/ascii_palettes directory for ASCII palettes to use as colormaps.

geoips.geoips_utils.**find_config**(*subpackage_name*, *config_basename*, *txt_suffix='.yaml'*)

Find matching config file within GEOIPS packages.

Given 'subpackage_name', 'config_basename', and txt_suffix, find matching text file within GEOIPS packages.

Parameters

- **subpackage_name** (*str*) – subdirectory under GEOIPS package to look for text file ie text_fname = geoips/<subpackage_name>/<config_basename><txt_suffix>
- **config_basename** (*str*) – text basename to look for, ie text_fname = geoips/<subpackage_name>/<config_basename><txt_suffix>
- **txt_suffix** (*str*) – suffix to look for on config file, defaults to ".yaml" ie text_fname = geoips/<subpackage_name>/<config_basename><txt_suffix>

Returns
text_fname – Full path to text filename

Return type
str

geoips.geoips_utils.**find_entry_point**(*namespace*, *name*, *default=None*)

Find object matching 'name' using GEOIPS entry point namespace 'namespace'.

Automatically add 'geoips' prefix to namespace for disambiguation.

Parameters

- **namespace** (*str*) – Entry point namespace (e.g. 'readers')
- **name** (*str*) – Entry point name (e.g. 'amsr2_netcdf')
- **default** (*entry point, optional*) – Default value if no match is found. If this is not set (i.e. None), then no match will result in an exception

geoips.geoips_utils.**get_all_entry_points**(*namespace*)

Return all entry points in GEOIPS entry point namespace 'namespace'.

Automatically add 'geoips' prefix to namespace for disambiguation.

Parameters

namespace (*str*) – Entry point namespace (e.g. 'readers')

geoips.geoips_utils.**get_entry_point_group**(*group*)

Get entry point group.

geoips.geoips_utils.**get_required_geoips_xarray_attrs**()

Interface deprecated v2.0.

geoips.geoips_utils.**list_entry_points**(*namespace*)

List names of objects in GEOIPS entry point namespace 'namespace'.

Automatically add 'geoips' prefix to namespace for disambiguation.

Parameters

namespace (*str*) – Entry point namespace (e.g. 'readers')

geoips.geoips_utils.**list_product_source_dict_yamls**()

List all YAML files containing product source specifications.

Search in all geoips packages.

Returns

List of all product source dict YAMLs in all geoips packages

Return type

list

geoips.geoips_utils.**list_product_specs_dict_yamls**()

List all YAML files containing product params in all geoips packages.

Returns

List of all product params dict YAMLs in all geoips packages

Return type
list

geoips.geoips_utils.**load_all_yaml_plugins**()

Find all YAML plugins in registered plugin packages.

Search the `plugins` directory of each registered plugin package for files ending in `.yaml`. Read each plugin file

geoips.geoips_utils.**merge_nested_dicts**(*dest*, *src*, *in_place=True*)

Perform an in-place merge of src into dest.

Performs an in-place merge of src into dest while preserving any values that already exist in dest.

geoips.geoips_utils.**output_process_times**(*process_datetimes*, *num_jobs=None*, *job_str='GeoIPS 2'*)

Calculate and print the process times from the process_datetimes dictionary.

Parameters
process_datetimes (*dict*) – dictionary formatted as follows:

- `process_datetimes['overall_start']` - overall start datetime of the entire script
- `process_datetimes['overall_end']` - overall end datetime of the entire script
- `process_datetimes[process_name]['start']` - start time of an individual process
- `process_datetimes[process_name]['end']` - end time of an individual process

geoips.geoips_utils.**replace_geoips_paths**(*fname*, *replace_paths=None*, *base_paths=None*)

Replace standard environment variables with their non-expanded equivalents.

Ie, replace

- `$HOME/geoproc/geoips_packages with $GEOIPS_PACKAGES_DIR`
- `$HOME/geoproc/geoips_outdirs with $GEOIPS_OUTDIRS`
- `$HOME/geoproc with $GEOIPS_BASEDIR`

This allows generating output YAML fields / NetCDF attributes that can match between different instantiations.

Parameters

- **fname** (*str*) – Full path to a filename on disk

- **replace_paths** (*list, default=None*) –
  - Explicit list of standard variable names you would like replaced.
  - If None, replace ['GEOIPS_OUTDIRS', 'GEOIPS_PACKAGES_DIR', 'GEOIPS_TESTDATA_DIR', 'GEOIPS_DEPENDENCIES_DIR', 'GEOIPS_BASEDIR']
- **base_paths** (*list, default=None*) –
  - List of PATHS dictionaries in which to find the "replace_paths" variables
  - If None, use geoips.filenames.base_paths

Returns

fname – Path to file on disk, with explicit path replaced with environment variable name and/or full URL.

Return type

str

Notes

Note it replaces ALL standard variables that have a corresponding <key>_URL variable.

Additionally, it replaces variables specified in "replace_paths" list with the unexpanded environment variable name.

6.1.7 Module contents

The Geolocated Information Processing System (GeoIPS).

GeoIPS ® Base Package

The GeoIPS Base Package provides a Python 3 based architecture supporting a wide variety of satellite and weather data processing. The modular nature of the GeoIPS base infrastructure also allows plug-and-play capability for user-specified custom functionality.

Homepage: https://github.com/NRLMMD-GEOIPS/geoips

Distribution Statement A. Approved for public release. Distribution unlimited.
# # #
Author:
Naval Research Laboratory, Marine Meteorology Division
# # #
This program is free software: you can redistribute it and/or modify it under

the terms of the NRLMMD License included with this program. This program is
distributed WITHOUT ANY WARRANTY; without even the implied warranty of
MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the included license
for more details. If you did not receive the license, for more information see:
https://github.com/U-S-NRL-Marine-Meteorology-Division/

CHAPTER

SEVEN

CONTACT

contact geoips@nrlmry.navy.mil

Distribution Statement A. Approved for public release. Distribution unlimited.
# # #
Author:
Naval Research Laboratory, Marine Meteorology Division
# # #
This program is free software: you can redistribute it and/or modify it under
the terms of the NRLMMD License included with this program. This program is
distributed WITHOUT ANY WARRANTY; without even the implied warranty of
MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the included license
for more details. If you did not receive the license, for more information see:
https://github.com/U-S-NRL-Marine-Meteorology-Division/

7.1 About Us

contact geoips@nrlmry.navy.mil

PYTHON MODULE INDEX

g

geoips, 286
geoips.cli, 276
geoips.commandline, 58
geoips.commandline.args, 55
geoips.commandline.list_available_modules, 56
geoips.commandline.log_setup, 57
geoips.commandline.run_procflow, 57
geoips.commandline.test_interfaces, 57
geoips.commandline.update_tc_tracks_database, 57
geoips.compare_outputs, 277
geoips.data_manipulations, 68
geoips.data_manipulations.conversions, 58
geoips.data_manipulations.corrections, 58
geoips.data_manipulations.info, 65
geoips.data_manipulations.merge, 65
geoips.dev, 75
geoips.dev.output_config, 68
geoips.dev.product, 71
geoips.errors, 282
geoips.filenames, 76
geoips.filenames.base_paths, 75
geoips.filenames.duplicate_files, 75
geoips.geoips_utils, 282
geoips.image_utils, 90
geoips.image_utils.colormap_utils, 76
geoips.image_utils.maps, 80
geoips.image_utils.mpl_utils, 84
geoips.interfaces, 108
geoips.interfaces.base, 98
geoips.interfaces.module_based, 96
geoips.interfaces.module_based.algorithms, 90
geoips.interfaces.module_based.colormappers, 91
geoips.interfaces.module_based.coverage_checkers, 91
geoips.interfaces.module_based.filename_formatte 92
geoips.interfaces.module_based.interpolators, 92
geoips.interfaces.module_based.output_formatters 93
geoips.interfaces.module_based.procflows, 94
geoips.interfaces.module_based.readers, 94
geoips.interfaces.module_based.sector_adjusters, 94
geoips.interfaces.module_based.sector_metadata_g 95
geoips.interfaces.module_based.sector_spec_gener 95
geoips.interfaces.module_based.title_formatters, 95
geoips.interfaces.yaml_based, 98
geoips.interfaces.yaml_based.feature_annotators, 96
geoips.interfaces.yaml_based.gridline_annotators 96
geoips.interfaces.yaml_based.product_defaults, 97
geoips.interfaces.yaml_based.products, 97
geoips.interfaces.yaml_based.sectors,

98
geoips.plugins, 253
geoips.plugins.modules, 252
geoips.plugins.modules.algorithms, 120
geoips.plugins.modules.algorithms.pmw_tb, 112
geoips.plugins.modules.algorithms.pmw_tb.pmw_37pct, 109
geoips.plugins.modules.algorithms.pmw_tb.pmw_89pct, 110
geoips.plugins.modules.algorithms.pmw_tb.pmw_color37, 110
geoips.plugins.modules.algorithms.pmw_tb.pmw_color89, 111
geoips.plugins.modules.algorithms.sfc_winds, 114
geoips.plugins.modules.algorithms.sfc_winds.windbarbs, 112
geoips.plugins.modules.algorithms.single_channel, 118
geoips.plugins.modules.algorithms.visir, 117
geoips.plugins.modules.algorithms.visir.Night_Vis, 114
geoips.plugins.modules.algorithms.visir.Night_Vis_GeoIPS1, 115
geoips.plugins.modules.algorithms.visir.Night_Vis_IR, 116
geoips.plugins.modules.algorithms.visir.Night_Vis_IR_GeoIPS1, 117
geoips.plugins.modules.colormappers, 134
geoips.plugins.modules.colormappers.cmap_rgb, 131
geoips.plugins.modules.colormappers.matplotlib_linear_norm, 132
geoips.plugins.modules.colormappers.pmw_tb, 127
geoips.plugins.modules.colormappers.pmw_tb.cmap_150H, 120
geoips.plugins.modules.colormappers.pmw_tb.cmap_37H, 121
geoips.plugins.modules.colormappers.pmw_tb.cmap_37H_Legacy, 122
geoips.plugins.modules.colormappers.pmw_tb.cmap_89H, 122
geoips.plugins.modules.colormappers.pmw_tb.cmap_ASCAT, 123
geoips.plugins.modules.colormappers.pmw_tb.cmap_AVAPS, 123
geoips.plugins.modules.colormappers.pmw_tb.cmap_GMI, 124
geoips.plugins.modules.colormappers.pmw_tb.cmap_SMAP, 124
geoips.plugins.modules.colormappers.pmw_tb.cmap_SMOS, 125
geoips.plugins.modules.colormappers.pmw_tb.cmap_WSF, 126
geoips.plugins.modules.colormappers.tpw, 128
geoips.plugins.modules.colormappers.tpw.tpw_cims, 127
geoips.plugins.modules.colormappers.tpw.tpw_purp, 127
geoips.plugins.modules.colormappers.tpw.tpw_pwat, 128
geoips.plugins.modules.colormappers.visir, 130
geoips.plugins.modules.colormappers.visir.Infrared_Gray, 129
geoips.plugins.modules.colormappers.visir.IR_BD, 128
geoips.plugins.modules.colormappers.visir.WV, 130
geoips.plugins.modules.colormappers.winds, 131
geoips.plugins.modules.colormappers.winds.wind_radii, 130
geoips.plugins.modules.coverage_checkers, 138
geoips.plugins.modules.coverage_checkers.center_radius_150H, 134
geoips.plugins.modules.coverage_checkers.center_radius_37H, 135
geoips.plugins.modules.coverage_checkers.masked_pct_37H_Legacy, 136
geoips.plugins.modules.coverage_checkers.numpy_a

136 geoips.plugins.modules.coverage_checkers.rgba,
137 geoips.plugins.modules.coverage_checkers.windbarbs,
137 geoips.plugins.modules.filename_formatters,
154 geoips.plugins.modules.filename_formatters.basic_fname,
140 geoips.plugins.modules.filename_formatters.geoips_fname,
140 geoips.plugins.modules.filename_formatters.geoips_netcdf_fname,
144 geoips.plugins.modules.filename_formatters.geotiff_fname,
145 geoips.plugins.modules.filename_formatters.metadata_default_fname,
146 geoips.plugins.modules.filename_formatters.tc_clean_fname,
146 geoips.plugins.modules.filename_formatters.tc_fname,
148 geoips.plugins.modules.filename_formatters.text_winds_day_fname,
150 geoips.plugins.modules.filename_formatters.text_winds_full_fname,
151 geoips.plugins.modules.filename_formatters.text_winds_tc_fname,
152 geoips.plugins.modules.filename_formatters.utils,
139 geoips.plugins.modules.filename_formatters.utils.tc_file_naming,
138 geoips.plugins.modules.interpolators,
164 geoips.plugins.modules.interpolators.pyresample_wrappers,
156 geoips.plugins.modules.interpolators.pyresample_wrappers.interp_gaussian,
155 geoips.plugins.modules.interpolators.pyresample_wrappers.interp_nearest,
155 geoips.plugins.modules.interpolators.scipy_wrappers,
156 geoips.plugins.modules.interpolators.scipy_wrappers.interp_grid,
156 geoips.plugins.modules.interpolators.utils,
164 geoips.plugins.modules.interpolators.utils.boxdef,
156 geoips.plugins.modules.interpolators.utils.interp_grid,
161 geoips.plugins.modules.interpolators.utils.interp_nearest,
162 geoips.plugins.modules.output_formatters,
155 geoips.plugins.modules.output_formatters.full_disk_image,
164 geoips.plugins.modules.output_formatters.geotiff,
165 geoips.plugins.modules.output_formatters.imagery_annotated,
165 geoips.plugins.modules.output_formatters.imagery_clean,
166 geoips.plugins.modules.output_formatters.imagery_gtiff,
167 geoips.plugins.modules.output_formatters.imagery_web_mercator,
169 geoips.plugins.modules.output_formatters.metadata,
170 geoips.plugins.modules.output_formatters.netcdf_cf,
172 geoips.plugins.modules.output_formatters.netcdf_geoips,
173 geoips.plugins.modules.output_formatters.netcdf_xarray,
174 geoips.plugins.modules.output_formatters.text_winds,
174 geoips.plugins.modules.output_formatters.unprojected_image,
175 geoips.plugins.modules.procflows, 185
175 geoips.plugins.modules.procflows.config_based,
175 geoips.plugins.modules.procflows.single_source,
181 geoips.plugins.modules.readers, 343
189 geoips.plugins.modules.readers.abi_l2_netcdf,
190 geoips.plugins.modules.readers.abi_netcdf,
191 geoips.plugins.modules.readers.ahi_hsd,
191 geoips.plugins.modules.readers.amsr2_netcdf,

194
geoips.plugins.modules.readers.amsr2_remss_winds_netcdf,
195
geoips.plugins.modules.readers.amsub_hdf,
196
geoips.plugins.modules.readers.amsub_mirs,
199
geoips.plugins.modules.readers.ascat_uhr_netcdf,
204
geoips.plugins.modules.readers.atms_hdf5,
205
geoips.plugins.modules.readers.ewsg_netcdf,
209
geoips.plugins.modules.readers.geoips_netcdf,
211
geoips.plugins.modules.readers.gmi_hdf5,
212
geoips.plugins.modules.readers.imerg_hdf5,
214
geoips.plugins.modules.readers.mimic_netcdf,
215
geoips.plugins.modules.readers.modis_hdf4,
217
geoips.plugins.modules.readers.saphir_hdf5,
219
geoips.plugins.modules.readers.sar_winds_netcdf,
220
geoips.plugins.modules.readers.scat_knmi_winds_netcdf,
221
geoips.plugins.modules.readers.scat_noaa_winds_netcdf,
223
geoips.plugins.modules.readers.seviri_hrit,
224
geoips.plugins.modules.readers.sfc_winds_netcdf,
227
geoips.plugins.modules.readers.smap_remss_winds_netcdf,
228
geoips.plugins.modules.readers.smos_winds_netcdf,
229
geoips.plugins.modules.readers.ssmi_binary,
230
geoips.plugins.modules.readers.ssmis_binary,
233
geoips.plugins.modules.readers.utils,
189
geoips.plugins.modules.readers.utils.geostationa
185
geoips.plugins.modules.readers.utils.hrit_reader
187
geoips.plugins.modules.readers.utils.remss_reade
188
geoips.plugins.modules.readers.viirs_netcdf,
234
geoips.plugins.modules.readers.wfabba_ascii,
237
geoips.plugins.modules.readers.windsat_idr37_bin
238
geoips.plugins.modules.readers.windsat_remss_win
242
geoips.plugins.modules.sector_metadata_generator
250
geoips.plugins.modules.sector_metadata_generator
243
geoips.plugins.modules.sector_metadata_generator
248
geoips.plugins.modules.sector_spec_generators,
251
geoips.plugins.modules.sector_spec_generators.ce
250
geoips.plugins.modules.title_formatters,
252
geoips.plugins.modules.title_formatters.static_s
251
geoips.plugins.modules.title_formatters.tc_copyr
252
geoips.plugins.modules.title_formatters.tc_stand
252
geoips.sector_utils, 270
geoips.sector_utils.estimate_area_extent,
253
geoips.sector_utils.overpass_predictor,
256
geoips.sector_utils.projections, 259
geoips.sector_utils.tc_tracks, 260
geoips.sector_utils.tc_tracks_database,
262
geoips.sector_utils.utils, 263
geoips.sector_utils.yaml_utils, 269
geoips.utils, 271
geoips.utils.decorators, 270

geoips.utils.memusg, 271
geoips.xarray_utils, 275
geoips.xarray_utils.data, 271
geoips.xarray_utils.time, 274

INDEX

A

add_args() (*in module geoips.commandline.args*), 55
add_description_to_yamldict() (*in module geoips.sector_utils.yaml_utils*), 269
add_dynamic_datetime_to_yamldict() (*in module geoips.sector_utils.yaml_utils*), 269
add_filename_extra_field() (*in module geoips.plugins.modules.procflows.single_source*), 181
add_list_interface_parser() (*in module geoips.cli*), 276
add_projection_to_yamldict() (*in module geoips.sector_utils.yaml_utils*), 269
add_sectorinfo_to_yamldict() (*in module geoips.sector_utils.yaml_utils*), 269
add_to_xarray() (*in module geoips.plugins.modules.readers.modis_hdf4*), 217
add_to_xarray() (*in module geoips.plugins.modules.readers.viirs_netcdf*), 235
AlgorithmsInterface (*class in geoips.interfaces.module_based.algorithms*), 90
allowable_kwargs (*geoips.interfaces.module_based.colormappers.ColormappersInterface attribute*), 91
allowable_kwargs (*geoips.interfaces.module_based.coverage_checkers.CoverageCheckersInterface attribute*), 91
alpha_from_masked_arrays() (*in module geoips.image_utils.mpl_utils*), 84
annotation_metadata (*geoips.plugins.modules.readers.utils.hrit_reader.HritI property*), 187
append_xarray_dicts() (*in module geoips.plugins.modules.readers.ssmis_binary*), 233
apply_data_range() (*in module geoips.data_manipulations.corrections*), 58
apply_gamma() (*in module geoips.data_manipulations.corrections*), 60
apply_maximum_value() (*in module geoips.data_manipulations.corrections*), 60
apply_minimum_value() (*in module geoips.data_manipulations.corrections*), 60
apply_offset() (*in module geoips.data_manipulations.corrections*), 61
apply_scale_factor() (*in module geoips.data_manipulations.corrections*), 61
apply_solar_zenith_correction() (*in module geoips.data_manipulations.corrections*), 62
area_def_to_yamldict() (*in module geoips.sector_utils.yaml_utils*), 269
area_def_to_yamlfile() (*in module geoips.sector_utils.yaml_utils*), 270
assemble_geoips_fname() (*in module geoips.plugins.modules.filename_formatters.geoips_fn* 140

assemble_geoips_netcdf_fname() (*in module geoips.plugins.modules.filename_formatters.geoips_netcdf_fname*), 144
assemble_tc_fname() (*in module geoips.plugins.modules.filename_formatters.tc_fname*), 148
assemble_windspeeds_text_full_fname() (*in module geoips.plugins.modules.filename_formatters.text_windspeeds_full_fname*), 151
assemble_windspeeds_text_tc_fname() (*in module geoips.plugins.modules.filename_formatters.text_windspeeds_tc_fname*), 152
AutoGenError, 185, 191

B

band (*geoips.plugins.modules.readers.seviri_hrit.Chan property*), 224
band (*geoips.plugins.modules.readers.utils.hrit_reader.HritFile property*), 187
band_num (*geoips.plugins.modules.readers.seviri_hrit.Chan property*), 224
bands (*geoips.plugins.modules.readers.seviri_hrit.ChanList property*), 225
BaseInterface (*class in geoips.interfaces.base*), 98
BaseModuleInterface (*class in geoips.interfaces.base*), 99
BaseModulePlugin (*class in geoips.interfaces.base*), 100
basename (*geoips.plugins.modules.readers.utils.hrit_reader.HritFile property*), 187
BaseYamlInterface (*class in geoips.interfaces.base*), 100
BaseYamlPlugin (*class in geoips.interfaces.base*), 101
block_info (*geoips.plugins.modules.readers.utils.hrit_reader.HritFile property*), 187
block_map (*geoips.plugins.modules.readers.utils.hrit_reader.HritFile property*), 187

C

calculate_abi_geolocation() (*in module geoips.plugins.modules.readers.abi_l2_netcdf*), 189
calculate_cdf_fsnhe_polynomial() (*in module geoips.plugins.modules.readers.seviri_hrit*), 225
calculate_overpass() (*in module geoips.sector_utils.overpass_predictor*), 256
calculate_sol_r_angles() (*in module geoips.plugins.modules.readers.utils.geostationary_geo*), 185
call() (*in module geoips.plugins.modules.algorithms.pmw_tb.pmw_37pc*), 109
call() (*in module geoips.plugins.modules.algorithms.pmw_tb.pmw_89pc*), 110
call() (*in module geoips.plugins.modules.algorithms.pmw_tb.pmw_color*), 110
call() (*in module geoips.plugins.modules.algorithms.pmw_tb.pmw_color*), 111
call() (*in module geoips.plugins.modules.algorithms.sfc_winds.windbarb*), 112
call() (*in module geoips.plugins.modules.algorithms.single_channel*), 118
call() (*in module geoips.plugins.modules.algorithms.visir.Night_Vis*), 114
call() (*in module geoips.plugins.modules.algorithms.visir.Night_Vis_Ge*), 115
call() (*in module geoips.plugins.modules.algorithms.visir.Night_Vis_IR*), 116
call() (*in module geoips.plugins.modules.algorithms.visir.Night_Vis_IR_*), 117
call() (*in module geoips.plugins.modules.colormappers.cmap_rgb*), 131

call() (in module geoips.plugins.modules.colormappers.matplotlib_grayscale_plugins), 132
call() (in module geoips.plugins.modules.colormappers.pmw_tb.cmap_ip37H), 120
call() (in module geoips.plugins.modules.colormappers.pmw_tb.cmap_ip37pct), 121
call() (in module geoips.plugins.modules.colormappers.pmw_tb.cmap_ip37pct_Legacy), 122
call() (in module geoips.plugins.modules.colormappers.pmw_tb.cmap_ip37pct_Physical), 122
call() (in module geoips.plugins.modules.colormappers.pmw_tb.cmap_ip37pct), 123
call() (in module geoips.plugins.modules.colormappers.pmw_tb.cmap_ip89H), 123
call() (in module geoips.plugins.modules.colormappers.pmw_tb.cmap_ip89pct_Legacy), 124
call() (in module geoips.plugins.modules.colormappers.pmw_tb.cmap_ip89pct_Physical), 125
call() (in module geoips.plugins.modules.colormappers.pmw_tb.cmap_ip89pctHWB), 124
call() (in module geoips.plugins.modules.colormappers.pmw_tb.cmap_ip89pct), 126
call() (in module geoips.plugins.modules.colormappers.pmw_tb.cmap_ipBasin), 126
call() (in module geoips.plugins.modules.colormappers.tpw.tpw_cimss), 127
call() (in module geoips.plugins.modules.colormappers.tpw.tpw_purple), 127
call() (in module geoips.plugins.modules.colormappers.tpw.tpw_pwats), 128
call() (in module geoips.plugins.modules.colormappers.visir.Infrared), 129
call() (in module geoips.plugins.modules.colormappers.visir.IR_BD), 128
call() (in module geoips.plugins.modules.colormappers.visir.WV), 130
call() (in module geoips.plugins.modules.colormappers.winds.wind_rad 130
call() (in module geoips.plugins.modules.coverage_checkers.center_radi 134
call() (in module geoips.plugins.modules.coverage_checkers.center_radi 135
call() (in module geoips.plugins.modules.coverage_checkers.masked_ar 136
call() (in module geoips.plugins.modules.coverage_checkers.numpy_arr 136
call() (in module geoips.plugins.modules.coverage_checkers.rgba), 137
call() (in module geoips.plugins.modules.coverage_checkers.windbarbs) 137
call() (in module geoips.plugins.modules.filename_formatters.basic_fna 140
call() (in module geoips.plugins.modules.filename_formatters.geoips_fn 142
call() (in module geoips.plugins.modules.filename_formatters.geoips_ne 144
call() (in module geoips.plugins.modules.filename_formatters.geotiff_fn 145
call() (in module geoips.plugins.modules.filename_formatters.metadata_ 146

call() (in module geoips.plugins.modules.filename_formatters.tc_clean_name), 146
call() (in module geoips.plugins.modules.filename_formatters.tc_fname), 149
call() (in module geoips.plugins.modules.filename_formatters.text_winds_plugin_name), 150
call() (in module geoips.plugins.modules.filename_formatters.text_winds_plugin_name), 152
call() (in module geoips.plugins.modules.filename_formatters.text_winds_plugin_name), 154
call() (in module geoips.plugins.modules.interpolators.pyresample_gaussian_interp), 155
call() (in module geoips.plugins.modules.interpolators.pyresample_nearest_neighbor), 155
call() (in module geoips.plugins.modules.interpolators.scipy_wrapper_interp), 156
call() (in module geoips.plugins.modules.output_formatters.full_disk_image), 164
call() (in module geoips.plugins.modules.output_formatters.geotiff_standard), 165
call() (in module geoips.plugins.modules.output_formatters.imagery_annotated), 165
call() (in module geoips.plugins.modules.output_formatters.imagery_clean), 166
call() (in module geoips.plugins.modules.output_formatters.imagery_windbarbs), 167
call() (in module geoips.plugins.modules.output_formatters.imagery_windbarbs_clean), 169
call() (in module geoips.plugins.modules.output_formatters.metadata_default), 170
call() (in module geoips.plugins.modules.output_formatters.metadata_tc), 172
call() (in module geoips.plugins.modules.output_formatters.netcdf_geoips), 173
call() (in module geoips.plugins.modules.output_formatters.netcdf_xarray), 174
call() (in module geoips.plugins.modules.output_formatters.text_winds), 174
call() (in module geoips.plugins.modules.output_formatters.unprojected_imagery), 175
call() (in module geoips.plugins.procflows.config_based), 175
call() (in module geoips.plugins.procflows.single_source), 181
call() (in module geoips.plugins.modules.readers.abi_l2_netcdf), 189
call() (in module geoips.plugins.modules.readers.abi_netcdf), 190
call() (in module geoips.plugins.modules.readers.ahi_hsd), 191
call() (in module geoips.plugins.modules.readers.amsr2_netcdf), 194
call() (in module geoips.plugins.modules.readers.amsr2_remss_winds_netcdf), 195
call() (in module geoips.plugins.modules.readers.amsub_hdf), 197
call() (in module geoips.plugins.modules.readers.amsub_mirs), 202
call() (in module geoips.plugins.modules.readers.ascat_uhr_netcdf), 204

call() (*in module geoips.plugins.modules.readers.atms_hdf5*), 207
call() (*in module geoips.plugins.modules.readers.ewsg_netcdf*), 209
call() (*in module geoips.plugins.modules.readers.geoips_netcdf*), 211
call() (*in module geoips.plugins.modules.readers.gmi_hdf5*), 212
call() (*in module geoips.plugins.modules.readers.imerg_hdf5*), 214
call() (*in module geoips.plugins.modules.readers.mimic_netcdf*), 215
call() (*in module geoips.plugins.modules.readers.modis_hdf4*), 217
call() (*in module geoips.plugins.modules.readers.saphir_hdf5*), 219
call() (*in module geoips.plugins.modules.readers.sar_winds_netcdf*), 220
call() (*in module geoips.plugins.modules.readers.scat_knmi_winds_geotiff*), 221
call() (*in module geoips.plugins.modules.readers.scat_noaa_winds_geotiff*), 223
call() (*in module geoips.plugins.modules.readers.seviri_hrit*), 225
call() (*in module geoips.plugins.modules.readers.sfc_winds_text*), 227
call() (*in module geoips.plugins.modules.readers.smap_remss_winds_netcdf*), 228
call() (*in module geoips.plugins.modules.readers.smos_winds_netcdf*), 229
call() (*in module geoips.plugins.modules.readers.ssmi_binary*), 231
call() (*in module geoips.plugins.modules.readers.ssmis_binary*), 233
call() (*in module geoips.plugins.modules.readers.viirs_netcdf*), 235
call() (*in module geoips.plugins.modules.readers.wfabba_ascii*), 237
call() (*in module geoips.plugins.modules.readers.windsat_idr37_binary*), 240
call() (*in module geoips.plugins.modules.readers.windsat_remss_winds_*), 242
call() (*in module geoips.plugins.modules.sector_metadata_generators.ba*), 246
call() (*in module geoips.plugins.modules.sector_metadata_generators.tc*), 248
call() (*in module geoips.plugins.modules.sector_spec_generators.center_*), 250
call() (*in module geoips.plugins.modules.title_formatters.static_standard*), 251
call() (*in module geoips.plugins.modules.title_formatters.tc_copyright*), 252
call() (*in module geoips.plugins.modules.title_formatters.tc_standard*), 252
cartesian_coords (*geoips.plugins.modules.interpolators.utils.boxdefinitio attribute*), 158
center_longitude() (*in module geoips.sector_utils.estimate_area_extent*), 253
Chan (*class in geoips.plugins.modules.readers.seviri_hrit*), 224

ChanList (*class in geoips.plugins.modules.readers.seviri_hrit*), 225
chans (*geoips.plugins.modules.readers.seviri_hrit.ChanList property*), 225
check_center_coverage() (*in module geoips.sector_utils.utils*), 263
check_command_line_args() (*in module geoips.commandline.args*), 55
check_db() (*in module geoips.sector_utils.tc_tracks_database*), 262
check_feature_annotator() (*in module geoips.image_utils.maps*), 80
check_gridline_annotator() (*in module geoips.image_utils.maps*), 80
check_tle_name_to_passed_names() (*in module geoips.sector_utils.overpass_predictor*), 256
ColormappersInterface (*class in geoips.interfaces.module_based.colormappers*), 91
combine_filename_extra_fields() (*in module geoips.plugins.modules.procflows.single_source*), 182
compare_dicts() (*in module geoips.plugins.modules.readers.seviri_hrit*), 226
compare_outputs() (*in module geoips.compare_outputs*), 277
compressed (*geoips.plugins.modules.readers.utils.hrit_reader.HritFile property*), 187
compute_lat_auto_spacing() (*in module geoips.image_utils.maps*), 80
compute_lon_auto_spacing() (*in module geoips.image_utils.maps*), 80
convert_epoch_to_datetime64() (*in module geoips.plugins.modules.readers.atms_hdf5*), 208
convert_west2east() (*in module geoips.sector_utils.estimate_area_extent*), 253
copy_sector_info() (*in module geoips.sector_utils.utils*), 264
copy_standard_metadata() (*in module geoips.geoips_utils*), 282
corners (*geoips.plugins.modules.interpolators.utils.boxdefinit property*), 158
corners (*geoips.plugins.modules.interpolators.utils.boxdefinit property*), 159
countsToRad() (*in module geoips.plugins.modules.readers.seviri_hrit*), 226
CoverageCheckersInterface (*class in geoips.interfaces.module_based.coverage_checkers*), 91
CoverageError, 282
create_areadefinition_from_yaml() (*in module geoips.sector_utils.utils*), 264
create_colorbar() (*in module geoips.image_utils.mpl_utils*), 84
create_figure_and_main_ax_and_mapobj() (*in module geoips.image_utils.mpl_utils*), 85
create_linear_segmented_colormap() (*in module geoips.image_utils.colormap_utils*), 76
create_radius() (*in module geoips.plugins.modules.coverage_checkers.center_radi* 134
create_tc_sector_info_dict() (*in module geoips.sector_utils.tc_tracks*), 260

D

daterange() (*in module geoips.data_manipulations.merge*), 65
decompress() (*geoips.plugins.modules.readers.utils.hrit_read method*), 187
deprecated (*class in geoips.utils.decorators*), 270
deprecation() (*in module geoips.geoips_utils*), 283
developmental() (*in module geoips.utils.decorators*), 270

dirname (*geoips.plugins.modules.readers.utils.hrit_reader.HritFile method*), 92
dirname (*geoips.plugins.modules.readers.utils.hrit_reader.HritFile property*), 187
draw_features() (*in module geoips.image_utils.maps*), 80
draw_gridlines() (*in module geoips.image_utils.maps*), 81

E

ellps2axis() (*in module geoips.image_utils.maps*), 81
end (*geoips.plugins.modules.interpolators.utils.boxdefinition.BoxDefinition attribute*), 157
EntryPointError, 282
epilogue (*geoips.plugins.modules.readers.utils.hrit_reader.HritFile property*), 188
esitmate_area_from_center() (*in module geoips.sector_utils.estimate_area_extent*), 253
estimate_area_extent() (*in module geoips.sector_utils.estimate_area_extent*), 254
extend_with_default() (*in module geoips.interfaces.base*), 105

F

FeatureAnnotatorsInterface (*class in geoips.interfaces.yaml_based.feature_annotators*), 96
file_type (*geoips.plugins.modules.readers.utils.hrit_reader.HritFile property*), 188
FilenameFormattersInterface (*class in geoips.interfaces.module_based.filename_formatters*), 92
filter_area_defs_actual_time() (*in module geoips.sector_utils.utils*), 264
find_all_txt_plugins() (*in module geoips.geoips_utils*), 283
find_ascii_palette() (*in module geoips.geoips_utils*), 283
find_config() (*in module geoips.geoips_utils*), 283
find_datafiles_in_range() (*in module geoips.data_manipulations.merge*), 65
find_duplicates() (*geoips.interfaces.module_based.filename_formatters.FilenameFormattersInterface method*), 92
find_entry_point() (*in module geoips.geoips_utils*), 283
findDiff() (*in module geoips.plugins.modules.readers.ahi_hsd*), 192
floor_minute() (*in module geoips.sector_utils.overpass_predictor*), 257
format_windbarb_data() (*in module geoips.plugins.modules.output_formatters.imagery_windbarbs*), 168
formclass (*in module geoips.cli*), 276
from_ascii() (*geoips.plugins.modules.readers.utils.hrit_reader.HritFile method*), 92
from_file() (*in module geoips.image_utils.colormap_utils*), 77

G

generateMinMaxLatLong() (*in module geoips.sector_utils.estimate_area_extent*), 255
geoips
    module, 286
geoips.cli
    module, 276
geoips.commandline
    module, 58
geoips.commandline.args
    module, 55
geoips.commandline.list_available_modules
    module, 56
geoips.commandline.log_setup
    module, 57
geoips.commandline.run_procflow
    module, 57
geoips.commandline.test_interfaces
    module, 57
geoips.commandline.update_tc_tracks_database
    module, 57
geoips.compare_outputs
    module, 277
geoips.data_manipulations
    module, 68
geoips.data_manipulations.conversions
    module, 58

geoips.data_manipulations.corrections
module, 58
geoips.data_manipulations.info
module, 65
geoips.data_manipulations.merge
module, 65
geoips.dev
module, 75
geoips.dev.output_config
module, 68
geoips.dev.product
module, 71
geoips.errors
module, 282
geoips.filenames
module, 76
geoips.filenames.base_paths
module, 75
geoips.filenames.duplicate_files
module, 75
geoips.geoips_utils
module, 282
geoips.image_utils
module, 90
geoips.image_utils.colormap_utils
module, 76
geoips.image_utils.maps
module, 80
geoips.image_utils.mpl_utils
module, 84
geoips.interfaces
module, 108
geoips.interfaces.base
module, 98
geoips.interfaces.module_based
module, 96
geoips.interfaces.module_based.algorithms
module, 90
geoips.interfaces.module_based.colormappers
module, 91
geoips.interfaces.module_based.coverage_checkers
module, 91
geoips.interfaces.module_based.filename_formatters
module, 92
geoips.interfaces.module_based.interpolators
module, 92
geoips.interfaces.module_based.output_formatters
module, 93
geoips.interfaces.module_based.procflows
module, 94
geoips.interfaces.module_based.readers
module, 94
geoips.interfaces.module_based.sector_adjusters
module, 94
geoips.interfaces.module_based.sector_metadata_g
module, 95
geoips.interfaces.module_based.sector_spec_gener
module, 95
geoips.interfaces.module_based.title_formatters
module, 95
geoips.interfaces.yaml_based
module, 98
geoips.interfaces.yaml_based.feature_annotators
module, 96
geoips.interfaces.yaml_based.gridline_annotators
module, 96
geoips.interfaces.yaml_based.product_defaults
module, 97
geoips.interfaces.yaml_based.products
module, 97
geoips.interfaces.yaml_based.sectors
module, 98
geoips.plugins
module, 253
geoips.plugins.modules
module, 252
geoips.plugins.modules.algorithms
module, 120
geoips.plugins.modules.algorithms.pmw_tb
module, 112
geoips.plugins.modules.algorithms.pmw_tb.pmw_37p
module, 109
geoips.plugins.modules.algorithms.pmw_tb.pmw_89p
module, 110
geoips.plugins.modules.algorithms.pmw_tb.pmw_col
module, 110
geoips.plugins.modules.algorithms.pmw_tb.pmw_col
module, 111
geoips.plugins.modules.algorithms.sfc_winds
module, 114

geoips.plugins.modules.algorithms.sfc_winds.windbarbs
module, 112
geoips.plugins.modules.algorithms.single_channel
module, 118
geoips.plugins.modules.algorithms.visir
module, 117
geoips.plugins.modules.algorithms.visir.Night_Vis
module, 114
geoips.plugins.modules.algorithms.visir.Night_Vis_GeoIPS1
module, 115
geoips.plugins.modules.algorithms.visir.Night_Vis_IR
module, 116
geoips.plugins.modules.algorithms.visir.Night_Vis_IR_GeoIPS1
module, 117
geoips.plugins.modules.colormappers
module, 134
geoips.plugins.modules.colormappers.cmap_rgb
module, 131
geoips.plugins.modules.colormappers.matplotlib_linear_norm
module, 132
geoips.plugins.modules.colormappers.pmw_tb
module, 127
geoips.plugins.modules.colormappers.pmw_tb.cmap_150H
module, 120
geoips.plugins.modules.colormappers.pmw_tb.cmap_37H
module, 121
geoips.plugins.modules.colormappers.pmw_tb.cmap_37H_Legacy
module, 122
geoips.plugins.modules.colormappers.pmw_tb.cmap_37H_Physical
module, 122
geoips.plugins.modules.colormappers.pmw_tb.cmap_37pct
module, 123
geoips.plugins.modules.colormappers.pmw_tb.cmap_89H
module, 123
geoips.plugins.modules.colormappers.pmw_tb.cmap_89H_Legacy
module, 124
geoips.plugins.modules.colormappers.pmw_tb.cmap_89H_Physical
module, 125
geoips.plugins.modules.colormappers.pmw_tb.cmap_89HW
module, 124
geoips.plugins.modules.colormappers.pmw_tb.cmap_89pct
module, 126
geoips.plugins.modules.colormappers.pmw_tb.cmap_Rain
module, 126
geoips.plugins.modules.colormappers.tpw

module, 128
geoips.plugins.modules.colormappers.tpw.tpw_cims
module, 127
geoips.plugins.modules.colormappers.tpw.tpw_purp
module, 127
geoips.plugins.modules.colormappers.tpw.tpw_pwat
module, 128
geoips.plugins.modules.colormappers.visir
module, 130
geoips.plugins.modules.colormappers.visir.Infrar
module, 129
geoips.plugins.modules.colormappers.visir.IR_BD
module, 128
geoips.plugins.modules.colormappers.visir.WV
module, 130
geoips.plugins.modules.colormappers.winds
module, 131
geoips.plugins.modules.colormappers.winds.wind_r
module, 130
geoips.plugins.modules.coverage_checkers
module, 138
geoips.plugins.modules.coverage_checkers.center_
module, 134
geoips.plugins.modules.coverage_checkers.center_
module, 135
geoips.plugins.modules.coverage_checkers.masked_
module, 136
geoips.plugins.modules.coverage_checkers.numpy_a
module, 136
geoips.plugins.modules.coverage_checkers.rgba
module, 137
geoips.plugins.modules.coverage_checkers.windbar
module, 137
geoips.plugins.modules.filename_formatters
module, 154
geoips.plugins.modules.filename_formatters.basic
module, 140
geoips.plugins.modules.filename_formatters.geoip
module, 140
geoips.plugins.modules.filename_formatters.geoip
module, 144
geoips.plugins.modules.filename_formatters.geoti
module, 145
geoips.plugins.modules.filename_formatters.metad
module, 146

geoips.plugins.modules.filename_formatters.tc_clean_fname
module, 146
geoips.plugins.modules.filename_formatters.tc_fname
module, 148
geoips.plugins.modules.filename_formatters.text_winds_day_fname
module, 150
geoips.plugins.modules.filename_formatters.text_winds_full_fname
module, 151
geoips.plugins.modules.filename_formatters.text_winds_tc_fname
module, 152
geoips.plugins.modules.filename_formatters.utils
module, 139
geoips.plugins.modules.filename_formatters.utils.tc_file_naming
module, 138
geoips.plugins.modules.interpolators
module, 164
geoips.plugins.modules.interpolators.pyresample_wrappers
module, 156
geoips.plugins.modules.interpolators.pyresample_wrappers.interp_gauss
module, 155
geoips.plugins.modules.interpolators.pyresample_wrappers.interp_nearest
module, 155
geoips.plugins.modules.interpolators.scipy_wrappers
module, 156
geoips.plugins.modules.interpolators.scipy_wrappers.interp_grid
module, 156
geoips.plugins.modules.interpolators.utils
module, 164
geoips.plugins.modules.interpolators.utils.boxdefinitions
module, 156
geoips.plugins.modules.interpolators.utils.interp_pyresample
module, 161
geoips.plugins.modules.interpolators.utils.interp_scipy
module, 162
geoips.plugins.modules.output_formatters
module, 175
geoips.plugins.modules.output_formatters.full_disk_image
module, 164
geoips.plugins.modules.output_formatters.geotiff_standard
module, 165
geoips.plugins.modules.output_formatters.imagery_annotated
module, 165
geoips.plugins.modules.output_formatters.imagery_clean
module, 166
geoips.plugins.modules.output_formatters.imagery_windbarbs
module, 167
geoips.plugins.modules.output_formatters.imagery
module, 169
geoips.plugins.modules.output_formatters.metadat
module, 170
geoips.plugins.modules.output_formatters.metadat
module, 172
geoips.plugins.modules.output_formatters.netcdf_
module, 173
geoips.plugins.modules.output_formatters.netcdf_
module, 174
geoips.plugins.modules.output_formatters.text_wi
module, 174
geoips.plugins.modules.output_formatters.unproje
module, 175
geoips.plugins.modules.procflows
module, 185
geoips.plugins.modules.procflows.config_based
module, 175
geoips.plugins.modules.procflows.single_source
module, 181
geoips.plugins.modules.readers
module, 243
geoips.plugins.modules.readers.abi_l2_netcdf
module, 189
geoips.plugins.modules.readers.abi_netcdf
module, 190
geoips.plugins.modules.readers.ahi_hsd
module, 191
geoips.plugins.modules.readers.amsr2_netcdf
module, 194
geoips.plugins.modules.readers.amsr2_remss_winds
module, 195
geoips.plugins.modules.readers.amsub_hdf
module, 196
geoips.plugins.modules.readers.amsub_mirs
module, 199
geoips.plugins.modules.readers.ascat_uhr_netcdf
module, 204
geoips.plugins.modules.readers.atms_hdf5
module, 205
geoips.plugins.modules.readers.ewsg_netcdf
module, 209
geoips.plugins.modules.readers.geoips_netcdf
module, 211

geoips.plugins.modules.readers.gmi_hdf5
    module, 212
geoips.plugins.modules.readers.imerg_hdf5
    module, 214
geoips.plugins.modules.readers.mimic_netcdf
    module, 215
geoips.plugins.modules.readers.modis_hdf4
    module, 217
geoips.plugins.modules.readers.saphir_hdf5
    module, 219
geoips.plugins.modules.readers.sar_winds_netcdf
    module, 220
geoips.plugins.modules.readers.scat_knmi_winds_netcdf
    module, 221
geoips.plugins.modules.readers.scat_noaa_winds_netcdf
    module, 223
geoips.plugins.modules.readers.seviri_hrit
    module, 224
geoips.plugins.modules.readers.sfc_winds_text
    module, 227
geoips.plugins.modules.readers.smap_remss_winds_netcdf
    module, 228
geoips.plugins.modules.readers.smos_winds_netcdf
    module, 229
geoips.plugins.modules.readers.ssmi_binary
    module, 230
geoips.plugins.modules.readers.ssmis_binary
    module, 233
geoips.plugins.modules.readers.utils
    module, 189
geoips.plugins.modules.readers.utils.geostationary_geolocation
    module, 185
geoips.plugins.modules.readers.utils.hrit_reader
    module, 187
geoips.plugins.modules.readers.utils.remss_reader
    module, 188
geoips.plugins.modules.readers.viirs_netcdf
    module, 234
geoips.plugins.modules.readers.wfabba_ascii
    module, 237
geoips.plugins.modules.readers.windsat_idr_binary
    module, 238
geoips.plugins.modules.readers.windsat_remss_netcdf
    module, 242
geoips.plugins.modules.sector_metadata_generators
    module, 250
geoips.plugins.modules.sector_metadata_generators.config_based_sector_metadata_generator
    module, 243
geoips.plugins.modules.sector_metadata_generators.tc_sector_metadata_generator
    module, 248
geoips.plugins.modules.sector_spec_generators
    module, 251
geoips.plugins.modules.sector_spec_generators.center_radius_sector_spec_generator
    module, 250
geoips.plugins.modules.title_formatters
    module, 252
geoips.plugins.modules.title_formatters.static_string_title_formatter
    module, 251
geoips.plugins.modules.title_formatters.tc_copyright_formatter
    module, 252
geoips.plugins.modules.title_formatters.tc_standard_formatter
    module, 252
geoips.sector_utils
    module, 270
geoips.sector_utils.estimate_area_extent
    module, 258
geoips.sector_utils.overpass_predictor
    module, 256
geoips.sector_utils.projections
    module, 259
geoips.sector_utils.tc_tracks
    module, 260
geoips.sector_utils.tc_tracks_database
    module, 262
geoips.sector_utils.utils
    module, 263
geoips.sector_utils.yaml_utils
    module, 269
geoips.utils
    module, 271
geoips.utils.decorators
    module, 270
geoips.utils.memusg
    module, 271
geoips.xarray_utils
    module, 275
geoips.xarray_utils.data
    module, 271
geoips.xarray_utils.time
    module, 274

geoips_fname_remove_duplicates() (*in module geoips.plugins.modules.filename_formatters.geoips_fname*), 143

geoips_netcdf_match() (*in module geoips.compare_outputs*), 277

geolocation_metadata (*geoips.plugins.modules.readers.utils.hrit_reader.HritFile property*), 188

geotiffs_match() (*in module geoips.compare_outputs*), 278

get_2d_false_corners() (*in module geoips.plugins.modules.interpolators.utils.boxdefinitions*), 160

get_alg_xarray() (*in module geoips.plugins.modules.procflows.single_source*), 182

get_all_entry_points() (*in module geoips.geoips_utils*), 284

get_all_storms_from_db() (*in module geoips.sector_utils.tc_tracks_database*), 262

get_area_def_list_from_dict() (*in module geoips.plugins.modules.procflows.config_based*), 176

get_area_defs_from_available_sectors() (*in module geoips.plugins.modules.procflows.config_based*), 176

get_area_defs_from_command_line_args() (*in module geoips.plugins.modules.procflows.single_source*), 182

get_band_metadata() (*in module geoips.plugins.modules.readers.abi_netcdf*), 191

get_band_metadata() (*in module geoips.plugins.modules.readers.ahi_hsd*), 192

get_bg_xarray() (*in module geoips.plugins.modules.procflows.config_based*), 177

get_bounding_box_lonlats() (*geoips.plugins.modules.interpolators.utils.boxdefinitions.MaskedCornersSwathDefinition method*), 158

get_bounding_box_lonlats() (*geoips.plugins.modules.interpolators.utils.boxdefinitio method*), 159

get_cmap_from_product() (*in module geoips.dev.product*), 71

get_command_line_args() (*in module geoips.commandline.args*), 56

get_config_dict() (*in module geoips.plugins.modules.procflows.config_based*), 177

get_covg_args_from_product() (*in module geoips.dev.product*), 72

get_covg_from_product() (*in module geoips.dev.product*), 72

get_data() (*in module geoips.plugins.modules.readers.abi_netcdf*), 191

get_data() (*in module geoips.plugins.modules.readers.ahi_hsd*), 193

get_data_box_definition() (*in module geoips.plugins.modules.interpolators.utils.interp_pyres*), 161

get_data_range() (*in module geoips.dev.product*), 73

get_datetime_from_datetime64() (*in module geoips.xarray_utils.time*), 274

get_entry_point_group() (*in module geoips.geoips_utils*), 284

get_filename() (*in module geoips.plugins.modules.procflows.single_source*), 182

get_filename_formatter_kwargs() (*in module geoips.dev.output_config*), 68

get_filename_formatters() (*in module geoips.dev.output_config*), 68

get_final_roi() (*in module geoips.plugins.modules.interpolators.pyresample_wrap*), 155

get_final_storm_name_bdeck() (*in module geoips.plugins.modules.sector_metadata_generators.b*), 246

get_geolocation() (*in module geoips.plugins.modules.readers.utils.geostationary_ge*

185
get_geolocation_cache_filename() (*in module geoips.plugins.modules.readers.utils.geostationary_geolocation*), 186
get_indexes() (*in module geoips.plugins.modules.readers.utils.geostationary_geolocation*), 186
get_interface() (*in module geoips.cli*), 276
get_invest_number_bdeck() (*in module geoips.plugins.modules.sector_metadata_generators.tcvitals_to_bdeck*), 246
get_lat_center() (*in module geoips.sector_utils.utils*), 264
get_lat_lon_points() (*in module geoips.xarray_utils.data*), 271
get_lat_lon_points_numpy() (*in module geoips.xarray_utils.data*), 272
get_latitude_longitude() (*in module geoips.plugins.modules.readers.abi_netcdf*), 191
get_latitude_longitude() (*in module geoips.plugins.modules.readers.ahi_hsd*), 193
get_latitude_longitude() (*in module geoips.plugins.modules.readers.seviri_hrit*), 226
get_lon_center() (*in module geoips.sector_utils.utils*), 264
get_matching_files() (*in module geoips.data_manipulations.merge*), 66
get_max_from_xarray_time() (*in module geoips.xarray_utils.time*), 274
get_max_lat() (*in module geoips.sector_utils.utils*), 264
get_max_lon() (*in module geoips.sector_utils.utils*), 264
get_metadata() (*in module geoips.plugins.modules.readers.abi_l2_netcdf*), 190
get_metadata_filename_formatter() (*in module geoips.dev.output_config*), 69
get_metadata_filename_formatter_kwargs() (*in module geoips.dev.output_config*), 69
get_metadata_output_formatter() (*in module geoips.dev.output_config*), 69
get_metadata_output_formatter_kwargs() (*in module geoips.dev.output_config*), 69
get_min_from_xarray_time() (*in module geoips.xarray_utils.time*), 275
get_min_lat() (*in module geoips.sector_utils.utils*), 265
get_min_lon() (*in module geoips.sector_utils.utils*), 265
get_minimum_coverage() (*in module geoips.dev.output_config*), 69
get_out_diff_fname() (*in module geoips.compare_outputs*), 278
get_output_config_type() (*in module geoips.dev.output_config*), 69
get_output_filenames() (*in module geoips.plugins.modules.procflows.single_source*), 183
get_output_formatter() (*in module geoips.dev.output_config*), 69
get_output_formatter_kwargs() (*in module geoips.dev.output_config*), 69
get_plugin() (*geoips.interfaces.base.BaseModuleInterface method*), 99
get_plugin() (*geoips.interfaces.base.BaseYamlInterface method*), 101
get_plugin() (*geoips.interfaces.yaml_based.products.Produc method*), 97
get_plugin_for_product() (*geoips.interfaces.module_based.coverage_checkers.C method*), 91
get_plugins() (*geoips.interfaces.base.BaseModuleInterface method*), 99
get_plugins() (*geoips.interfaces.base.BaseYamlInterface method*), 101
get_plugins() (*geoips.interfaces.yaml_based.products.ProductsInterf method*), 97
get_posix_from_datetime() (*in module geoips.xarray_utils.time*), 275

get_product_display_name() (*in module geoips.dev.product*), 73
get_projection() (*in module geoips.sector_utils.projections*), 259
get_rasterio_cmap_dict() (*in module geoips.plugins.modules.output_formatters.geotiff_standard*), 165
get_requested_datasets_for_variables() (*in module geoips.dev.product*), 74
get_required_geoips_xarray_attrs() (*in module geoips.geoips_utils*), 284
get_required_outputs() (*in module geoips.plugins.modules.procflows.config_based*), 177
get_required_variables() (*in module geoips.dev.product*), 74
get_resampled_read() (*in module geoips.plugins.modules.procflows.config_based*), 177
get_satellite_angles() (*in module geoips.plugins.modules.readers.utils.geostationary_geometry*), 186
get_schemas() (*in module geoips.interfaces.base*), 105
get_sectored_read() (*in module geoips.plugins.modules.procflows.config_based*), 178
get_sectored_xarrays() (*in module geoips.xarray_utils.data*), 272
get_sectors_from_yamls() (*in module geoips.sector_utils.utils*), 265
get_static_area_defs_for_xarray() (*in module geoips.sector_utils.utils*), 265
get_storm_start_datetime_from_bdeck_entry() (*in module geoips.plugins.modules.sector_metadata_generators.bdeck_parser*), 247
get_storm_start_datetime_from_bdeck_filename() (*in module geoips.plugins.modules.sector_metadata_generators.bdeck_parser*), 247
get_storm_subdir() (*in module geoips.plugins.modules.filename_formatters.utils.tc_file_naming*), 138
get_storm_year() (*in module geoips.plugins.modules.sector_metadata_generators.tc*), 249
get_stormyear_from_bdeck_filename() (*in module geoips.plugins.modules.sector_metadata_generators.bd*), 247
get_tc_area_defs_for_xarray() (*in module geoips.sector_utils.utils*), 266
get_tc_area_id() (*in module geoips.sector_utils.tc_tracks*), 261
get_tc_long_description() (*in module geoips.sector_utils.tc_tracks*), 261
get_title_string_from_objects() (*in module geoips.image_utils.mpl_utils*), 86
get_top_level_metadata() (*in module geoips.plugins.modules.readers.seviri_hrit*), 226
get_trackfile_area_defs() (*in module geoips.sector_utils.utils*), 266
get_validator() (*in module geoips.interfaces.base*), 105
get_variables_from_available_outputs_dict() (*in module geoips.plugins.modules.procflows.config_based*), 178
get_vis_ir_bg() (*in module geoips.xarray_utils.data*), 272
GridlineAnnotatorsInterface (*class in geoips.interfaces.yaml_based.gridline_annotators*), 96
gunzip_product() (*in module geoips.compare_outputs*), 278
gzip_product() (*in module geoips.compare_outputs*), 279

H

haversine_distance() (*in module geoips.sector_utils.estimate_area_extent*), 255
hourrange() (*in module geoips.data_manipulations.merge*), 68
HritDtype (*class in geoips.plugins.modules.readers.utils.hrit_reader*),

187
HritError, 187
HritFile (*class in geoips.plugins.modules.readers.utils.hrit_reader*), 187

I

images_match() (*in module geoips.compare_outputs*), 279
initialize_final_products() (*in module geoips.plugins.modules.procflows.config_based*), 178
interp_gaussian_kde() (*in module geoips.plugins.modules.interpolators.utils.interp_scipy*), 162
interp_griddata() (*in module geoips.plugins.modules.interpolators.utils.interp_scipy*), 162
interp_kd_tree() (*in module geoips.plugins.modules.interpolators.utils.interp_scipy*), 161
interpolate_storm_location() (*in module geoips.sector_utils.tc_tracks*), 261
InterpolatorsInterface (*class in geoips.interfaces.module_based.interpolators*), 92
intersection() (*geoips.plugins.modules.interpolators.utils.boxdefinitions.Box method*), 157
intersection() (*geoips.plugins.modules.interpolators.utils.boxdefinitions.MaskedCornersSwathDefinition method*), 158
intersection() (*geoips.plugins.modules.interpolators.utils.boxdefinitions.PolygonDefinition method*), 160
intersects() (*geoips.plugins.modules.interpolators.utils.boxdefinitions.Box method*), 157
invert_data_range() (*in module geoips.data_manipulations.corrections*), 62
is_crs() (*in module geoips.image_utils.maps*), 82
is_dynamic_sector() (*in module geoips.sector_utils.utils*), 267
is_geoips_netcdf() (*in module geoips.compare_outputs*), 279
is_geotiff() (*in module geoips.compare_outputs*), 279
is_hdf() (*in module geoips.compare_outputs*), 280
is_image() (*in module geoips.compare_outputs*), 280
is_requested_aid_type() (*in module geoips.sector_utils.utils*), 267
is_required_sector_type() (*in module geoips.plugins.modules.procflows.config_based*), 179
is_sector_type() (*in module geoips.sector_utils.utils*), 268
is_text() (*in module geoips.compare_outputs*), 280
is_valid_output_config() (*in module geoips.dev.output_config*), 70

L

lat_to_dec() (*in module geoips.plugins.modules.sector_metadata_generators.b*
247
lats (*geoips.plugins.modules.interpolators.utils.boxdefinitions attribute*), 158
Line (*class in geoips.plugins.modules.interpolators.utils.boxdefinitio*
157
list_dev_interfaces() (*in module geoips.cli*), 276
list_entry_points() (*in module geoips.geoips_utils*), 284
list_interface_plugins() (*in module geoips.cli*), 276
list_interfaces() (*in module geoips.cli*), 276
list_product_source_dict_yamls() (*in module geoips.geoips_utils*), 284
list_product_specs_dict_yamls() (*in module geoips.geoips_utils*), 284
load_all_yaml_plugins() (*in module geoips.geoips_utils*), 285
lon_to_dec() (*in module geoips.plugins.modules.sector_metadata_generators.b*
247

lons (*geoips.plugins.modules.interpolators.utils.boxdefs.MaskedCornersSwathDefinition attribute*), 158

M

main() (*in module geoips.cli*), 276
main() (*in module geoips.commandline.list_available_modules*), 56
main() (*in module geoips.commandline.run_procflow*), 57
main() (*in module geoips.commandline.test_interfaces*), 57
main() (*in module geoips.commandline.update_tc_tracks_database*), 57
make_dirs() (*in module geoips.filenames.base_paths*), 75
mask_day() (*in module geoips.data_manipulations.corrections*), 62
mask_night() (*in module geoips.data_manipulations.corrections*), 63
MaskedCornersSwathDefinition (*class in geoips.plugins.modules.interpolators.utils.boxdefs*), 157
merge_nested_dicts() (*in module geoips.geoips_utils*), 285
meridians() (*in module geoips.image_utils.maps*), 82
metadata (*geoips.plugins.modules.readers.utils.hrit_readers.HritFile property*), 188
metadata_to_datetime() (*in module geoips.plugins.modules.readers.abi_netcdf*), 191
metadata_to_datetime() (*in module geoips.plugins.modules.readers.ahi_hsd*), 193
minrange() (*in module geoips.data_manipulations.merge*), 68
module
    geoips, 286
    geoips.cli, 276
    geoips.commandline, 58
    geoips.commandline.args, 55
    geoips.commandline.list_available_modules, 56
    geoips.commandline.log_setup, 57
    geoips.commandline.run_procflow, 57
    geoips.commandline.test_interfaces, 57
    geoips.commandline.update_tc_tracks_database, 57
    geoips.compare_outputs, 277
    geoips.data_manipulations, 68
    geoips.data_manipulations.conversions, 58
    geoips.data_manipulations.corrections, 58
    geoips.data_manipulations.info, 65
    geoips.data_manipulations.merge, 65
    geoips.dev, 75
    geoips.dev.output_config, 68
    geoips.dev.product, 71
    geoips.errors, 282
    geoips.filenames, 76
    geoips.filenames.base_paths, 75
    geoips.filenames.duplicate_files, 75
    geoips.geoips_utils, 282
    geoips.image_utils, 90
    geoips.image_utils.colormap_utils, 76
    geoips.image_utils.maps, 80
    geoips.image_utils.mpl_utils, 84
    geoips.interfaces, 108
    geoips.interfaces.base, 98
    geoips.interfaces.module_based, 96
    geoips.interfaces.module_based.algorithms, 90
    geoips.interfaces.module_based.colormappers, 91
    geoips.interfaces.module_based.coverage_check 91
    geoips.interfaces.module_based.filename_forma 92

geoips.interfaces.module_based.interpolators, 92
geoips.interfaces.module_based.output_formatters, 93
geoips.interfaces.module_based.procflows, 94
geoips.interfaces.module_based.readers, 94
geoips.interfaces.module_based.sector_adjusters, 94
geoips.interfaces.module_based.sector_metadata_generators, 95
geoips.interfaces.module_based.sector_specifications, 95
geoips.interfaces.module_based.title_formatters, 95
geoips.interfaces.yaml_based, 98
geoips.interfaces.yaml_based.feature_annotators, 96
geoips.interfaces.yaml_based.gridline_annotators, 96
geoips.interfaces.yaml_based.product_defaults, 97
geoips.interfaces.yaml_based.products, 97
geoips.interfaces.yaml_based.sectors, 98
geoips.plugins, 253
geoips.plugins.modules, 252
geoips.plugins.modules.algorithms, 120
geoips.plugins.modules.algorithms.pmw_tb, 112
geoips.plugins.modules.algorithms.pmw_tb.pmw_37pct, 109
geoips.plugins.modules.algorithms.pmw_tb.pmw_89pct, 110
geoips.plugins.modules.algorithms.pmw_tb.pmw_color37, 110
geoips.plugins.modules.algorithms.pmw_tb.pmw_color89, 111
geoips.plugins.modules.algorithms.sfc_winds, 114
geoips.plugins.modules.algorithms.sfc_winds_windbarbs, 112
geoips.plugins.modules.algorithms.single_chan 118
geoips.plugins.modules.algorithms.visir, 117
geoips.plugins.modules.algorithms.visir.Night 114
geoips.plugins.modules.algorithms.visir.Night 115
geoips.plugins.modules.algorithms.visir.Night 116
geoips.plugins.modules.algorithms.visir.Night 117
geoips.plugins.modules.colormappers, 134
geoips.plugins.modules.colormappers.cmap_rgb, 131
geoips.plugins.modules.colormappers.matplotli
geoips.plugins.modules.colormappers.pmw_tb, 127
geoips.plugins.modules.colormappers.pmw_tb.cm
geoips.plugins.modules.colormappers.pmw_tb.cm 121
geoips.plugins.modules.colormappers.pmw_tb.cm 122
geoips.plugins.modules.colormappers.pmw_tb.cm 122
geoips.plugins.modules.colormappers.pmw_tb.cm 123
geoips.plugins.modules.colormappers.pmw_tb.cm
geoips.plugins.modules.colormappers.pmw_tb.cm 124
geoips.plugins.modules.colormappers.pmw_tb.cm 125
geoips.plugins.modules.colormappers.pmw_tb.cm
geoips.plugins.modules.colormappers.pmw_tb.cm 126
geoips.plugins.modules.colormappers.pmw_tb.cm
geoips.plugins.modules.colormappers.tpw, 128
geoips.plugins.modules.colormappers.tpw.tpw_c

127 geoips.plugins.modules.colormappers.tpw.tpw_purple,
127 geoips.plugins.modules.colormappers.tpw.tpw_pwat,
128 geoips.plugins.modules.colormappers.visir,
130 geoips.plugins.modules.colormappers.visir.infrared,
129 geoips.plugins.modules.colormappers.visir.IR_BD,
128 geoips.plugins.modules.colormappers.visir.WV,
130 geoips.plugins.modules.colormappers.winds,
131 geoips.plugins.modules.colormappers.winds.wind_radii_transitions,
130 geoips.plugins.modules.coverage_checkers,
138 geoips.plugins.modules.coverage_checkers.center_radius,
134 geoips.plugins.modules.coverage_checkers.center_radius_rgba,
135 geoips.plugins.modules.coverage_checkers.masked_arrays,
136 geoips.plugins.modules.coverage_checkers.numpy_arrays_nan,
136 geoips.plugins.modules.coverage_checkers.rgba,
137 geoips.plugins.modules.coverage_checkers.windbarbs,
137 geoips.plugins.modules.filename_formatters,
154 geoips.plugins.modules.filename_formatters.basic_fname,
140 geoips.plugins.modules.filename_formatters.geoips_fname,
140 geoips.plugins.modules.filename_formatters.geoips_netcdf_fname,
144 geoips.plugins.modules.filename_formatters.geotiff_fname,
145 geoips.plugins.modules.filename_formatters.metadata_default_fname,
146 geoips.plugins.modules.filename_formatters.tc_clean_fname,
146 geoips.plugins.modules.filename_formatters.tc
148 geoips.plugins.modules.filename_formatters.te
150 geoips.plugins.modules.filename_formatters.te
151 geoips.plugins.modules.filename_formatters.te
152 geoips.plugins.modules.filename_formatters.ut
153 geoips.plugins.modules.filename_formatters.ut
158 geoips.plugins.modules.interpolators,
164 geoips.plugins.modules.interpolators.pyresamp
156 geoips.plugins.modules.interpolators.pyresamp
155 geoips.plugins.modules.interpolators.pyresamp
155 geoips.plugins.modules.interpolators.scipy_wr
156 geoips.plugins.modules.interpolators.scipy_wr
158 geoips.plugins.modules.interpolators.utils,
164 geoips.plugins.modules.interpolators.utils.bo
156 geoips.plugins.modules.interpolators.utils.in
161 geoips.plugins.modules.interpolators.utils.in
162 geoips.plugins.modules.output_formatters,
175 geoips.plugins.modules.output_formatters.full
164 geoips.plugins.modules.output_formatters.geot
165 geoips.plugins.modules.output_formatters.imag
165 geoips.plugins.modules.output_formatters.imag
166 geoips.plugins.modules.output_formatters.imag
167 geoips.plugins.modules.output_formatters.imag

169
geoips.plugins.modules.output_formatters.metadata_default, 170
geoips.plugins.modules.output_formatters.metadata_tc, 172
geoips.plugins.modules.output_formatters.netcdf_geoips, 173
geoips.plugins.modules.output_formatters.netcdf_xarray, 174
geoips.plugins.modules.output_formatters.text_winds, 174
geoips.plugins.modules.output_formatters.unprojected_image, 175
geoips.plugins.modules.procflows, 185
geoips.plugins.modules.procflows.config_based, 175
geoips.plugins.modules.procflows.single_source, 181
geoips.plugins.modules.readers, 243
geoips.plugins.modules.readers.abi_l2_netcdf, 189
geoips.plugins.modules.readers.abi_netcdf, 190
geoips.plugins.modules.readers.ahi_hsd, 191
geoips.plugins.modules.readers.amsr2_netcdf, 194
geoips.plugins.modules.readers.amsr2_remss_winds_netcdf, 195
geoips.plugins.modules.readers.amsub_hdf, 196
geoips.plugins.modules.readers.amsub_mirs, 199
geoips.plugins.modules.readers.ascat_uhr_netcdf, 204
geoips.plugins.modules.readers.atms_hdf5, 205
geoips.plugins.modules.readers.ewsg_netcdf, 209
geoips.plugins.modules.readers.geoips_netcdf, 211
geoips.plugins.modules.readers.gmi_hdf5, 212
geoips.plugins.modules.readers.imerg_hdf5, 214
geoips.plugins.modules.readers.mimic_netcdf, 215
geoips.plugins.modules.readers.modis_hdf4, 217
geoips.plugins.modules.readers.saphir_hdf5, 219
geoips.plugins.modules.readers.sar_winds_netc 220
geoips.plugins.modules.readers.scat_knmi_wind 221
geoips.plugins.modules.readers.scat_noaa_wind 223
geoips.plugins.modules.readers.seviri_hrit, 224
geoips.plugins.modules.readers.sfc_winds_text 227
geoips.plugins.modules.readers.smap_remss_win 228
geoips.plugins.modules.readers.smos_winds_net 229
geoips.plugins.modules.readers.ssmi_binary, 230
geoips.plugins.modules.readers.ssmis_binary, 233
geoips.plugins.modules.readers.utils, 189
geoips.plugins.modules.readers.utils.geostati 185
geoips.plugins.modules.readers.utils.hrit_rea 187
geoips.plugins.modules.readers.utils.remss_re 188
geoips.plugins.modules.readers.viirs_netcdf, 234
geoips.plugins.modules.readers.wfabba_ascii, 237
geoips.plugins.modules.readers.windsat_idr37_ 238
geoips.plugins.modules.readers.windsat_remss_ 242
geoips.plugins.modules.sector_metadata_genera 250
geoips.plugins.modules.sector_metadata_genera

243
geoips.plugins.modules.sector_metadata_generators, 248
geoips.plugins.modules.sector_spec_generators, 251
geoips.plugins.modules.sector_spec_generators.center, 250
geoips.plugins.modules.title_formatters, 252
geoips.plugins.modules.title_formatters.standard, 251
geoips.plugins.modules.title_formatters.tc_copyright, 252
geoips.plugins.modules.title_formatters.tc_standard, 252
geoips.sector_utils, 270
geoips.sector_utils.estimate_area_extent, 253
geoips.sector_utils.overpass_predictor, 256
geoips.sector_utils.projections, 259
geoips.sector_utils.tc_tracks, 260
geoips.sector_utils.tc_tracks_database, 262
geoips.sector_utils.utils, 263
geoips.sector_utils.yaml_utils, 269
geoips.utils, 271
geoips.utils.decorators, 270
geoips.utils.memusg, 271
geoips.xarray_utils, 275
geoips.xarray_utils.data, 271
geoips.xarray_utils.time, 274

N

name (*geoips.interfaces.module_based.algorithms.AlgorithmsInterface attribute*), 90
name (*geoips.interfaces.module_based.colormappers.ColormappersInterface attribute*), 91
name (*geoips.interfaces.module_based.coverage_checkers.CoverageCheckersInterface attribute*), 91
name (*geoips.interfaces.module_based.filename_formatters.FilenameFormattersInterface attribute*), 92
name (*geoips.interfaces.module_based.interpolators.InterpolatorsInterface attribute*), 92
name (*geoips.interfaces.module_based.output_formatters.OutputFormattersInterface attribute*), 93
name (*geoips.interfaces.module_based.procflows.ProcflowsInterface attribute*), 94
name (*geoips.interfaces.module_based.readers.ReadersInterface attribute*), 94
name (*geoips.interfaces.module_based.sector_adjusters.SectorAdjustersInterface attribute*), 94
name (*geoips.interfaces.module_based.sector_metadata_generators.SectorMetadataGeneratorsInterface attribute*), 95
name (*geoips.interfaces.module_based.sector_spec_generators.SectorSpecGeneratorsInterface attribute*), 95
name (*geoips.interfaces.module_based.title_formatters.TitleFormattersInterface attribute*), 95
name (*geoips.interfaces.yaml_based.feature_annotators.FeatureAnnotatorsInterface attribute*), 96
name (*geoips.interfaces.yaml_based.gridline_annotators.GridlineAnnotatorsInterface attribute*), 96
name (*geoips.interfaces.yaml_based.product_defaults.ProductDefaultsInterface attribute*), 97
name (*geoips.interfaces.yaml_based.products.ProductsInterface attribute*), 97
name (*geoips.interfaces.yaml_based.sectors.SectorsInterface attribute*), 98
name (*geoips.plugins.modules.readers.seviri_hrit.Chan property*), 224
name (*geoips.plugins.modules.readers.utils.hrit_reader.HritFile property*), 188
names (*geoips.plugins.modules.readers.seviri_hrit.ChanList property*), 225
ndims (*geoips.plugins.modules.interpolators.utils.boxdefinition attribute*), 157
normalize() (*in module geoips.data_manipulations.corrections*), 63
NSEW_to_float() (*in module geoips.plugins.modules.sector_metadata_generators.tc_sector_file_parser*), 248

O

open_tc_db() (*in module geoips.sector_utils.tc_tracks_database*), 263

output_all_metadata() (*in module geoips.plugins.modules.procflows.single_source*), 183
output_clean_windbarbs() (*in module geoips.plugins.modules.output_formatters.imagery_windbarbs*), 168
output_metadata_yaml() (*in module geoips.plugins.modules.output_formatters.metadata_default*), 170
output_process_times() (*in module geoips.geoips_utils*), 285
output_tc_metadata_yaml() (*in module geoips.plugins.modules.output_formatters.metadata_tcdf*), 172
OutputFormattersInterface (*class in geoips.interfaces.module_based.output_formatters*), 93
overlaps() (*geoips.plugins.modules.interpolators.utils.boxdefinitions.PlanarPolygonDefinition method*), 160
overlaps_minmaxlatlon() (*geoips.plugins.modules.interpolators.utils.boxdefinitions.PlanarPolygonDefinition method*), 159
overlaps_minmaxlatlon() (*geoips.plugins.modules.interpolators.utils.boxdefinitions.SwathDefinition method*), 160

P

pad_area_definition() (*in module geoips.plugins.modules.procflows.single_source*), 183
parallels() (*in module geoips.image_utils.maps*), 82
parse_archive_metadata() (*in module geoips.plugins.modules.readers.modis_hdf4*), 218
parse_bdeck_line() (*in module geoips.plugins.modules.sector_metadata_generators.tc_parser*), 247
parse_core_metadata() (*in module geoips.plugins.modules.readers.modis_hdf4*), 218
parse_flat_sectorfile_line() (*in module geoips.plugins.modules.sector_metadata_generators.tc_parser*), 249
parse_header_line() (*in module geoips.plugins.modules.readers.wfabba_ascii*), 238
parse_metadata() (*in module geoips.plugins.modules.readers.modis_hdf4*), 218
parse_struct_metadata() (*in module geoips.plugins.modules.readers.modis_hdf4*), 219
percent_not_nan() (*in module geoips.data_manipulations.info*), 65
percent_unmasked() (*in module geoips.data_manipulations.info*), 65
percent_unmasked_rgba() (*in module geoips.image_utils.mpl_utils*), 87
planar_intersection_polygon() (*geoips.plugins.modules.interpolators.utils.boxdefinitions.PlanarPolygonDefinition method*), 160
planar_point_inside() (*geoips.plugins.modules.interpolators.utils.boxdefinitions.PlanarPolygonDefinition method*), 160
PlanarPolygonDefinition (*class in geoips.plugins.modules.interpolators.utils.boxdefinitions*), 159
plot_barbs() (*in module geoips.plugins.modules.output_formatters.imagery_windbarbs*), 169
plot_coverage() (*in module geoips.plugins.modules.coverage_checkers.center_radius_area*), 135
plot_data() (*in module geoips.plugins.modules.procflows.single_source*), 183
plot_image() (*in module geoips.image_utils.mpl_utils*), 87
plot_overlays() (*in module geoips.image_utils.mpl_utils*), 88
plugin_is_valid() (*geoips.interfaces.base.BaseModuleInterface method*), 99
plugin_is_valid() (*geoips.interfaces.base.BaseYamlInterface method*), 101

plugin_is_valid() (*geoips.interfaces.yaml_based.products.ProductsInterface method*), 97
plugin_module_to_obj() (*in module geoips.interfaces.base*), 105
plugin_repr() (*in module geoips.interfaces.base*), 106
plugin_yaml_to_obj() (*in module geoips.interfaces.base*), 106
PluginError, 282
plugins_all_valid() (*geoips.interfaces.base.BaseModuleInterface method*), 100
plugins_all_valid() (*geoips.interfaces.base.BaseYamlInterface method*), 101
predict_overpass_area_def() (*in module geoips.sector_utils.overpass_predictor*), 257
predict_overpass_yaml() (*in module geoips.sector_utils.overpass_predictor*), 258
predict_satellite_overpass() (*in module geoips.sector_utils.overpass_predictor*), 258
print_area_def() (*in module geoips.plugins.modules.procflows.single_source*), 184
print_gunzip_to_file() (*in module geoips.compare_outputs*), 280
print_gzip_to_file() (*in module geoips.compare_outputs*), 280
print_mem_usage() (*in module geoips.utils.memusg*), 271
print_resource_usage() (*in module geoips.utils.memusg*), 271
print_table() (*in module geoips.cli*), 276
process_sectored_data_output() (*in module geoips.plugins.modules.procflows.single_source*), 184
process_unsectored_data_outputs() (*in module geoips.plugins.modules.procflows.config_based*), 179
process_xarray_dict_to_output_format() (*in module geoips.plugins.modules.procflows.single_source*), 184
ProcflowsInterface (*class in geoips.interfaces.module_based.procflows*), 94
produce_current_time() (*in module geoips.dev.output_config*), 70
ProductDefaultsInterface (*class in geoips.interfaces.yaml_based.product_defaults*), 97
ProductsInterface (*class in geoips.interfaces.yaml_based.products*), 97
ProductsPluginValidator (*class in geoips.interfaces.yaml_based.products*), 97
prologue (*geoips.plugins.modules.readers.utils.hrit_reader.H property*), 188
Properties (*geoips.plugins.modules.interpolators.utils.boxde attribute*), 158

R

radToBT() (*in module geoips.plugins.modules.readers.seviri_hrit*), 226
radToRef() (*in module geoips.plugins.modules.readers.seviri_hrit*), 226
RawDescriptionArgumentDefaultsHelpFormatter (*class in geoips.cli*), 276
read10bit() (*in module geoips.plugins.modules.readers.utils.hrit_reader*), 188
read_amsr_data() (*in module geoips.plugins.modules.readers.amsr2_netcdf*), 195
read_amsr_mbt() (*in module geoips.plugins.modules.readers.amsr2_netcdf*), 195
read_amsr_winds() (*in module geoips.plugins.modules.readers.amsr2_netcdf*), 195
read_atms_file() (*in module*

geoips.plugins.modules.readers.atms_hdf5), 208
read_byu_data() (*in module geoips.plugins.modules.readers.ascat_uhr_netcdf*), 205
read_gmi_file() (*in module geoips.plugins.modules.readers.gmi_hdf5*), 213
read_knmi_data() (*in module geoips.plugins.modules.readers.scat_knmi_hdf5*), 222
read_noaa_data() (*in module geoips.plugins.modules.readers.scat_noaa_winds_netcdf*), 224
read_remss_data() (*in module geoips.plugins.modules.readers.utils.remss_reader*), 188
read_sar_data() (*in module geoips.plugins.modules.readers.sar_winds_netcdf*), 221
read_satellite_tle() (*in module geoips.sector_utils.overpass_predictor*), 259
read_smos_data() (*in module geoips.plugins.modules.readers.smos_winds_netcdf*), 230
read_ssmis_data_file() (*in module geoips.plugins.modules.readers.ssmis_binary*), 234
read_wfabba_header() (*in module geoips.plugins.modules.readers.wfabba_ascii*), 238
read_wfabba_text() (*in module geoips.plugins.modules.readers.wfabba_ascii*), 238
read_xarray_netcdf() (*in module geoips.plugins.modules.readers.geoips_netcdf*), 212
ReadersInterface (*class in geoips.interfaces.module_based.readers*), 94
remove_duplicate_storm_positions() (*in module geoips.sector_utils.utils*), 268
remove_duplicates() (*geoips.interfaces.module_based.filename_formatters.FilenameFormattersInterface method*), 92
remove_duplicates() (*in module geoips.filenames.duplicate_files*), 75
remove_duplicates() (*in module geoips.image_utils.mpl_utils*), 88
remove_unsupported_kwargs() (*in module geoips.plugins.modules.procflows.single_source*), 184
replace_geoips_paths() (*in module geoips.geoips_utils*), 285
reprocess_storm() (*in module geoips.sector_utils.tc_tracks_database*), 263
required_args (*geoips.interfaces.module_based.algorithms.Algorithm attribute*), 90
required_args (*geoips.interfaces.module_based.colormappers.Colorm attribute*), 91
required_args (*geoips.interfaces.module_based.coverage_checkers.C attribute*), 91
required_args (*geoips.interfaces.module_based.filename_formatters.F attribute*), 92
required_args (*geoips.interfaces.module_based.interpolators.Interpo attribute*), 92
required_args (*geoips.interfaces.module_based.output_formatters.Ou attribute*), 93
required_args (*geoips.interfaces.module_based.procflows.ProcflowsI attribute*), 94
required_args (*geoips.interfaces.module_based.readers.ReadersInter attribute*), 94
required_args (*geoips.interfaces.module_based.sector_adjusters.Sect attribute*), 94
required_args (*geoips.interfaces.module_based.sector_metadata_gen attribute*), 95
required_args (*geoips.interfaces.module_based.sector_specifications.*

(*geoips.interfaces.module_based.sector_spec_generators.SectorSpecGeneratorsInterface attribute*), 95
required_args (*geoips.interfaces.module_based.title_formatters.TitleFormattersInterface attribute*), 96
required_chan() (*in module geoips.plugins.modules.readers.viirs_netcdf*), 236
required_geo() (*in module geoips.plugins.modules.readers.viirs_netcdf*), 236
required_geo_chan() (*in module geoips.plugins.modules.readers.viirs_netcdf*), 236
required_kwargs (*geoips.interfaces.module_based.algorithms.AlgorithmsInterface attribute*), 90
required_kwargs (*geoips.interfaces.module_based.colormappers.ColormappersInterface attribute*), 91
required_kwargs (*geoips.interfaces.module_based.coverage_checkers.CoverageCheckersInterface attribute*), 91
required_kwargs (*geoips.interfaces.module_based.filename_formatters.FilenameFormattersInterface attribute*), 92
required_kwargs (*geoips.interfaces.module_based.interpolators.InterpolatorsInterface attribute*), 92
required_kwargs (*geoips.interfaces.module_based.output_formatters.OutputFormattersInterface attribute*), 93
required_kwargs (*geoips.interfaces.module_based.procflows.ProcflowsInterface attribute*), 94
required_kwargs (*geoips.interfaces.module_based.readers.ReadersInterface attribute*), 94
required_kwargs (*geoips.interfaces.module_based.sector_adjusters.SectorAdjustersInterface attribute*), 94
required_kwargs (*geoips.interfaces.module_based.sector_metadata_generators.SectorMetadataGeneratorsInterface attribute*), 95
required_kwargs (*geoips.interfaces.module_based.sector_spec_generators.SectorSpecGeneratorsInterface attribute*), 95
required_kwargs (*geoips.interfaces.module_based.title_formatters.TitleFormattersInterface attribute*), 96
requires_bg() (*in module geoips.plugins.modules.procflows.config_based*), 180
rgba_from_arrays() (*in module geoips.image_utils.mpl_utils*), 88

S

save_image() (*in module geoips.image_utils.mpl_utils*), 88
scale_geotiff_data() (*in module geoips.plugins.modules.output_formatters.geotiff_standard*), 165
schemas (*geoips.interfaces.base.YamlPluginValidator attribute*), 101
sector_xarray_dataset() (*in module geoips.xarray_utils.data*), 272
sector_xarray_spatial() (*in module geoips.xarray_utils.data*), 273
sector_xarray_temporal() (*in module geoips.xarray_utils.data*), 273
sector_xarrays() (*in module geoips.xarray_utils.data*), 274
SectorAdjustersInterface (*class in geoips.interfaces.module_based.sector_adjusters*), 94
SectorMetadataGeneratorsInterface (*class in geoips.interfaces.module_based.sector_metadata_generators*), 95
SectorsInterface (*class in geoips.interfaces.yaml_based.sectors*), 98
SectorSpecGeneratorsInterface (*class in geoips.interfaces.module_based.sector_spec_generators*), 95
segment (*geoips.plugins.modules.readers.utils.hrit_reader.HritFile property*), 188
set_clat_clon_proj_info() (*in module geoips.plugins.modules.sector_spec_generators.center_coordinates*), 251

set_comparison_path() (*in module geoips.plugins.modules.procflows.config_based*), 181
set_fonts() (*in module geoips.image_utils.mpl_utils*), 89
set_gridlines_info_dict() (*in module geoips.image_utils.maps*), 82
set_lonlat_spacing() (*in module geoips.dev.output_config*), 71
set_matplotlib_colors_rgb() (*in module geoips.image_utils.colormap_utils*), 78
set_matplotlib_colors_standard() (*in module geoips.image_utils.colormap_utils*), 78
set_mpl_colors_info_dict() (*in module geoips.image_utils.colormap_utils*), 79
set_tc_area_def() (*in module geoips.sector_utils.tc_tracks*), 261
set_tc_coverage_check_area_def() (*in module geoips.sector_utils.utils*), 268
set_text_area_def() (*in module geoips.sector_utils.utils*), 268
set_title() (*in module geoips.image_utils.mpl_utils*), 89
set_variable_metadata() (*in module geoips.plugins.modules.readers.ahi_hsd*), 193
setup_logging() (*in module geoips.commandline.log_setup*), 57
shape (*geoips.plugins.modules.interpolators.utils.boxdefinitions.MaskedCornersSwathDefinition attribute*), 157
size (*geoips.plugins.modules.interpolators.utils.boxdefinitions.MaskedCornersSwathDefinition attribute*), 157
sort_by_band_and_seg() (*in module geoips.plugins.modules.readers.ahi_hsd*), 193
start (*geoips.plugins.modules.interpolators.utils.boxdefinitions.Line attribute*), 157
start_datetime (*geoips.plugins.modules.readers.utils.hrit_reader.HritFile property*), 188
storm_locations_match() (*in module geoips.sector_utils.utils*), 269

T

tc_fname_remove_duplicates() (*in module geoips.plugins.modules.filename_formatters.tc_fname*), 150
tc_storm_basedir() (*in module geoips.plugins.modules.filename_formatters.utils.tc_fil*), 138
test_interface() (*geoips.interfaces.base.BaseModuleInterface method*), 100
test_interface() (*geoips.interfaces.base.BaseYamlInterface method*), 101
test_interface() (*geoips.interfaces.yaml_based.products.ProductsInterf method*), 97
test_output_config_interface() (*in module geoips.dev.output_config*), 71
test_product() (*in module geoips.compare_outputs*), 280
text_match() (*in module geoips.compare_outputs*), 281
TitleFormattersInterface (*class in geoips.interfaces.module_based.title_formatters*), 95
trackfile_to_area_defs() (*in module geoips.sector_utils.tc_tracks*), 262
type (*geoips.plugins.modules.readers.seviri_hrit.Chan property*), 224
types (*geoips.plugins.modules.readers.utils.hrit_reader.HritD attribute*), 187

U

unit_conversion() (*in module geoips.data_manipulations.conversions*), 58
update_extra_field() (*in module geoips.plugins.modules.filename_formatters.utils.tc_fil*), 139
update_fields() (*in module geoips.sector_utils.tc_tracks_database*), 263
update_output_dict_from_command_line_args() (*in module*

geoips.plugins.modules.procflows.config_based), 181
update_sector_info_with_coverage() (*in module geoips.plugins.modules.output_formatters.yaml_tc*), 173
update_sector_info_with_data_times() (*in module geoips.plugins.modules.output_formatters.metadata_tc*), 173
update_sector_info_with_default_metadata() (*in module geoips.plugins.modules.output_formatters.metadata_default*), 171

V

validate() (*geoips.interfaces.base.YamlPluginValidator method*), 103
validate() (*geoips.interfaces.yaml_based.products.ProductsPluginValidator method*), 98
validate_list() (*geoips.interfaces.base.YamlPluginValidator method*), 103
validate_product() (*geoips.interfaces.yaml_based.products.ProductsPluginValidator method*), 98
validator (*geoips.interfaces.base.BaseYamlInterface attribute*), 101
validator (*geoips.interfaces.yaml_based.products.ProductsInterface attribute*), 97
validators (*geoips.interfaces.base.YamlPluginValidator attribute*), 103
verify_area_def() (*in module geoips.plugins.modules.procflows.single_source*), 184

W

write_text_winds() (*in module geoips.plugins.modules.output_formatters.text_winds*), 174
write_xarray_netcdf() (*in module geoips.plugins.modules.output_formatters.netcdf_xarray*), 174
write_yamldict() (*in module geoips.sector_utils.yaml_utils*), 270

X

XritError, 225

Y

YamlPluginValidator (*class in geoips.interfaces.base*), 101