TDS2STAC

Mostafa Hadizadeh

Jan 19, 2024

CONTENTS

A Python Ecosystem for Harvesting Datasets information from Thredds Data Server and Cultivating STAC-Metadata

CHAPTER

ONE

OVERVIEW:

STAC specification is a method of exposing spatial and temporal data collections in a standardized manner. Specifically, the SpatioTemporal Asset Catalog (STAC) specification describes and catalogs spatiotemporal assets using a common structure. This package creates STAC metadata by harvesting dataset details from the Thredds data server.

CHAPTER

TWO

INSTALLATION

You can find installation instructions in the [Installation](installation) section.

2.1 Installation

Warning: This page has been automatically generated as has not yet been reviewed by the authors of tds2stac!

To install the *tds2stac* package, we recommend that you install it from PyPi via

```
pip install tds2stac
```

Or install it directly from the source code repository on Gitlab via:

```
pip install git+https://codebase.helmholtz.cloud/cat4kit/ds2stac/tds2stac.git
```

The latter should however only be done if you want to access the development versions.

2.1.1 Installation for development

Please head over to our *Contribution* for installation instruction for development.

CHAPTER

THREE

TUTORIALS

A comprehensive tutorial designed to enhance your understanding of the functionality of this project.

3.1 Tutorials

This tutorial aims to demonstrate the utilization of TDS2STAC. The data sources to be utilized in this study include the following:

3.1.1 Creating the `tag_config.json` Configuration File: A Step-by-Step Guide

The configuration file, `tag_config.json`, contains the essential settings for extracting the textual content and attributes of tag elements from XML-based TDS web services in an external manner (not inside of source code), based on the user's specified preferences. Extensible Markup Language (XML) is a markup language that offers a set of regulations for the specification of data. XML enables the transmission of data alongside its corresponding description and additional particulars, hence safeguarding the preservation of data integrity. The currently available iteration of TDS2STAC is capable of accommodating all XML-based web services provided by Thredds data server, including WCS, WMS, WFS, ISO, DAP4, ncML, and Catalogs. This functionality facilitates the extraction of content from XML files through an innovative and user-friendly approach. In the subsequent discussion, we will go further into the composition of this particular file.

Upon examining the provided JSON file, it becomes evident that the primary keys inside this file correspond to the names of the extensions that are being targeted for harvesting. In the given JSON file, there are two extensions, namely `item_datacube_extension` and `item_scientific_extension`. The objective is to extract specific content from XML files within the TDS pertaining to these extensions.

```
{
    "item_datacube_extension": {},
    "item_scientific_extension": {},
}
```

As seen below, we aim to extract the variables of the extension `item_datacube_extension` such as `horizontal_extent_lon_min`, `horizontal_extent_lon_max`, `horizontal_extent_lat_min`, and `horizontal_extent_lat_max` for each items. To proceed, it is necessary to first comprehend the fundamental framework for defining variables. Subsequently, we can proceed with the definition of the variables that are to be harvested.

```
{
    "item_datacube_extension": {},
    "item_scientific_extension": {
                    "horizontal_extent_lon_min": {},
```

(continues on next page)

(continued from previous page)

```
                    "horizontal_extent_lon_max": {},
                    "horizontal_extent_lat_min": {},
                    "horizontal_extent_lat_max": {}
}
```

Main and constant attributes:

There exist four distinct modes of analysis for each variable in the tag_config.json file. The aforementioned kinds include `str`, `list`, `get`, and `check`. Each of them comprises both constant properties and variable attributes. The constant attributes are:

1. `tds2stac_mode_analyser`: It determines the method of analysis. As previously stated, there exist four distinct modes. In this tutorial, we will provide comprehensive explanations for the terms `str, list, get,` and `check.`
2. `tds2stac_manual_variable`: This attribute is employed when the variable is held constant. This feature is exclusively utilized for variables in the `str` and `list` modes. When a string or list is defined, such as `49.56` or [`EPSG:4326`, `EPSG:3857`], the utilization of this attribute becomes necessary.
3. `tds2stac_webservice_analyser`: This attribute is employed to designate the nomenclature of the webservices that is sought to be harvested from. This feature is exclusively utilized for variables of the `get` and `check` types. It has the capability to accommodate a comprehensive range of activated webservice names within the TDS framework, including but not limited to WCS, WMS, WFS, ISO, DAP4, ncML, and Catalog-XMLfile. They consists `ncml`, `wms`, `catalog`, `dap4`, `iso`.
4. `tds2stac_reference_key`: This attribute is just utilized for the `get` mode analyzer. In the context of the datacube extension, consider a scenario where there is a list consisting of five components representing variable IDs. In this scenario, the objective is to obtain the variable units for each of these elements. However, it is worth noting that three of these variables do not own any units inside the TDS webservie. The aforementioned attribute can be utilized to establish the reference key of the variable based on the variable_ids. It determines the variable_unit, and in cases where variable_unit is absent, it is defined as null. Ultimately, the two collected lists will possess equal lengths, so facilitating their utilization in subsequent stages, thereby rendering the process of evaluating them more manageable.

We shall provide separate descriptions for each of them and strive to understand them thoroughly.

Mode Analyzers

1. `str` TDS2STAC Mode Analyser

The `str` analyzer mode is utilized to declare a fixed string value for a variable within an extension. In the present scenario, if one intends to assign a constant string value such as `46.852` to the variable `horizontal_extent_lon_min` for STAC-Items, this mode can be employed. To gain further insights into this particular category, please refer to the illustrative sample provided below:

```
{
    "item_datacube_extension": {},
    "item_scientific_extension": {
                    "horizontal_extent_lon_min": {
                                                 "tds2stac_mode_analyser": "str",
                                                 "tds2stac_manual_variable": 46.852
                                                 },
                    "horizontal_extent_lon_max": {},
                    "horizontal_extent_lat_min": {},
```

(continues on next page)

(continued from previous page)

```
                    "horizontal_extent_lat_max": {}
}
```

In the aforementioned example, in order to establish a consistent minimum longitude for all STAC-Items within the datacube, it is necessary to designate `tds2stac_mode_analyser` as a string mode analyzer. Subsequently, the constant value of the variable is specified in the `tds2stac_manual_variable` field. In the present scenario, the fixed numerical number is 46.852.

2. `list` TDS2STAC Mode Analyser

The `list` mode analyzer is employed to establish a fixed collection of string values for a variable in an extension. In the present scenario, if there is a need to declare a variable named `variable_ids` that has a constant list of string values such as [`variable_id_1`, `variable_id_2`, `variable_id_3`] for STAC-Items, this particular data type can be utilized. For further elucidation on this particular category, please refer to the illustrative example provided below.

```
{
    "item_datacube_extension": {},
    "item_scientific_extension": {
                    "variable_ids": {
                                    "tds2stac_mode_analyser": "list",
                                    "tds2stac_manual_variable": "[variable_id_1,␣
↪variable_id_2, variable_id_3]"
                                    },
                "horizontal_extent_lon_max": {},
                "horizontal_extent_lat_min": {},
                "horizontal_extent_lat_max": {}
}
```

In the aforementioned example, in order to obtain a consistent list of variable IDs for all STAC-Items in the datacube, it is necessary to designate `tds2stac_mode_analyser` as a `list` mode analyzer. Subsequently, within the `tds2stac_manual_variable` field, the constant list of string values for the variable should be specified. In the present scenario, the constant list of string values consists of the following elements: `variable_id_1`, `variable_id_2`, and `variable_id_3`.

3. `get` TDS2STAC Mode Analyser

The `get` mode analyzer is designed to automatically harvest the characteristics and variables of each extension from the XML-based web services of TDS. As an illustration, our objective is to obtain the four distinct variables, namely `constellation`, `description`, `instruments`, and `horizontal_extent_lon_min`, from four distinct extensions in the form of XML-based web services provided by the TDS for filling out the variables of two extensions and metadata, namely `datacube` and `common_metadata`. These extensions are identified as 'ncml,' 'dap4,' 'iso,' and 'catalog.' For further elucidation on this particular category, please refer to the illustrative example provided below.

```
{

    "item_datacube_extension": {},
    "item_scientific_extension": {
                    "horizontal_extent_lon_min": {
                                        "tds2stac_mode_analyser": "get",
                                        "tds2stac_webservice_analyser": "ncml",
                                        "netcdf": null,
                                        "group": {
                                                "name":"CFMetadata"
                                                },
```

(continues on next page)

(continued from previous page)

```
                                        "attribute": {
                                                "name":"geospatial_lon_min",
                                                "value": null
                                                }
                                        },
                "horizontal_extent_lon_max": {},
                "horizontal_extent_lat_min": {},
                "horizontal_extent_lat_max": {}
                },
   "common_metadata":{
                    "constellation": {
                                    "tds2stac_mode_analyser": "get",
                                    "tds2stac_webservice_analyser": "iso",
                                    "gmi:MI_Metadata": null,
                                    "gmd:contact":null,
                                    "gmd:CI_ResponsibleParty": null,
                                    "gmd:organisationName": null,
                                    "gco:CharacterString": null

                                    },
                    "description": {
                                    "tds2stac_mode_analyser": "get",
                                    "tds2stac_webservice_analyser": "catalog",
                                    "catalog": null,
                                    "dataset":{
                                            "name": null
                                    }
                                    },
                    "instruments": {
                                    "tds2stac_mode_analyser": "get",
                                    "tds2stac_webservice_analyser": "dap4",
                                    "Dataset": null,
                                    "Attribute":{
                                            "name":null,
                                            "type":null
                                            }
                                    }
    },
}
```

As observed in the initial attribute of the first extension in the aforementioned example, namely `horizontal_extent_lon_min`, it is intended to retrieve the content from the webservice. The `tds2stac_mode_analyser` (get) is used to harvest data automatically from a web service, specifically the `tds2stac_webservice_analyser` which is in the format of ncml. The initial two keys (`tds2stac_mode_analyser`, `tds2stac_webservice_analyser`) are considered constant keys, as previously discussed.

However, the remaining keys represent tag elements name within the XML file of the provided web service (ncml). In the given ncml XML file, the objective is to harvest the value of the `geospatial_lon_min` attribute from the `CFMetadata` group tag element.

The initial tag element, denoted as `<netcdf xmlns='url' location='Not supplied due of security concerns.'>`, represents the `netcdf` tag name within the specified XML namespace `url`.

The tag `netcdf` is placed after the attribute `tds2stac_webservice_analyser`. Due to our sole interest in obtaining

the minimum longitude, we opt not to search for additional attributes within `netcdf` tag. Consequently, we assign a null value to the `netcdf` key to just move from this tag. Conversely, since we already possess just one `netcdf` tag name in whole XML file, it is unnecessary to include the attribute of the `netcdf` tag, such as {`xmlns:url location:Not provided because of security concerns.`}.

It is important to acknowledge that when encountering multiple occurrences of the `netcdf` or any other tag name within an XML file, it is advisable to include the attributes of the tag instead of leaving them as null. This approach facilitates more precise and desirable filtering outcomes.

As shown in the subsequent key, denoted as `group,` the presence of many groups necessitated a refinement process resulting in the adoption of the <`group name=`CFMetadata>` tag. The ultimate stage holds paramount significance.

In an imaginary scenario when the minimum longitude is not provided as a value of an attribute in the XML file, but rather as text enclosed within opening and closing tags, our objective is to harvest this information. To achieve this, we may simply prepend `null` as value of `attribute` in order to obtain the desired text. For instance, if the value of the `geospatial_lon_min` attribute in the <attribute name=`geospatial_lon_min` value=`113,361` type=`float`/> tag were structured as <`attribute name=`geospatial_lon_min type=`float`>113,361</attribute>`, it would be more convenient to retrieve it by inserting `null` as value of `attribute` key.

However, in this particular scenario, the attribute's value is being considered. Therefore, it is necessary to include the value of the attribute as a dictionary as value of `attribute` key. The `geospatial_lon_min` is placed as value of `name` key, while null is assigned to the value of `value` key. Using this approach the minimum longitude of each STAC-Item in the dataset can be obtained automatically from the ncml webservice of the TDS. It is important to acknowledge that if all values of tag attributes are included in the nested dictionary, TDS2STAC will search for the corresponding text within the tag, not attribute's value.

```
<netcdf xmlns="url" location="Not provided because of security concerns.">
<attribute name="title" value="IAGOS-CARIBIC netCDF4 data file"/>
<attribute name="creation_date" value="2023-09-26T11:01:56.344149+00:00"/>
<attribute name="mission" value="IAGOS-CARIBIC (CARIBIC-2), http://www.caribic-
↪atmospheric.com/"/>
<attribute name="data_description" value="All continuous measurements (10s averages) for␣
↪IAGOS-CARIBIC (onboard Airbus A340-600 HE of Lufthansa)"/>
<attribute name="data_institute" value="Institute of Meteorology and Climate Research␣
↪(IMK), Karlsruhe Institute of Technology (KIT), 76021 Karlsruhe, P.O. Box 3640, Germany
↪"/>
<attribute name="data_owners" value="A. Zahn; H. Boenisch; T. Gehrlein; F. Obersteiner;␣
↪contact: andreas.zahn@kit.edu"/>
<attribute name="data_contributors" value="https://gitlab.kit.edu/kit/imk-asf-top/IAGOS-
↪CARIBIC/-/blob/cb7705507465023e28937cb4f896a13058f6ebd0/doc/Caribic2_MS-dataset_
↪Contributors/CARIBIC_MS_contributors.md"/>
<attribute name="license" value="https://creativecommons.org/licenses/by/4.0/deed.en"/>
<attribute name="doi" value="10.5281/zenodo.8188548"/>
<attribute name="format_date" value="2023-09-26"/>
<attribute name="format_version" value="0.3"/>
<attribute name="history" value="Converted from NASA Ames format with na_to_nc4 from␣
↪caribic2dp.convert_caribic_na_nc4 module. Might contain only a subset of the␣
↪parameters from the original NASA Ames file. caribic2dp 0.2.16, https://gitlab.kit.edu/
↪FObersteiner/Caribic2dp"/>
<attribute name="conventions" value="CF-1.10"/>
<attribute name="ivar_C_format" value="%d"/>
<attribute name="_NCProperties" value="version=2,netcdf=4.9.3-development,hdf5=1.12.2"/>
<attribute name="_CoordSysBuilder" value="ucar.nc2.internal.dataset.conv.CF1Convention"/>
<dimension name="header_lines" length="197"/>
<dimension name="time" length="659"/>
```

(continues on next page)

(continued from previous page)

```
<group name="CFMetadata">
    <attribute name="geospatial_lon_min" value="113,361" type="float"/>
    <attribute name="geospatial_lat_min" value="14,539" type="float"/>
    <attribute name="geospatial_lon_max" value="121,069" type="float"/>
    <attribute name="geospatial_lat_max" value="23,519" type="float"/>
    <attribute name="geospatial_lon_units" value=""/>
    <attribute name="geospatial_lat_units" value=""/>
    <attribute name="geospatial_lon_resolution" value="0.011714285992561503"/>
    <attribute name="geospatial_lat_resolution" value="0.013647415717684389"/>
    <attribute name="time_coverage_start" value="2005-05-20T12:31:15Z"/>
    <attribute name="time_coverage_end" value="2005-05-20T14:20:55Z"/>
    <attribute name="time_coverage_units" value="seconds"/>
    <attribute name="time_coverage_resolution" value="10.0"/>
    <attribute name="time_coverage_duration" value="P0Y0M0DT1H49M40.000S"/>
</group>
:language: xml
```

In instances where there are many null values for attributes, the TDS2STAC searches for a value among those attributes and compiles them into a list. This can be illustrated using the following example:

```
"instruments": {
                "tds2stac_mode_analyser": "get",
                "tds2stac_webservice_analyser": "dap4",
                "Dataset": null,
                "Attribute":{
                        "name":null,
                        "type":null
                        }
                }
```

4. `check` TDS2STAC Mode Analyser

The `check` mode analyzer is employed to verify the existence of a specific tag element within the XML file of a web service. In the present scenario, if there is a need to verify the existence of a tag element named `geospatial_lon_min` within the XML file of the ncml web service, this particular data type can be utilized. For further elucidation on this particular category, please refer to the illustrative example provided below.

```
{
    "item_datacube_extension": {},
    "item_scientific_extension": {
                    "vertical_axis": {
                                    "tds2stac_mode_analyser": "check",
                                    "tds2stac_manual_variable": "z",
                                    "tds2stac_webservice_analyser": "ncml",
                                    "netcdf": null,
                                    "group": {
                                                "name":"CFMetadata"
                                            },
                                    "attribute": {
                                                "name":"geospatial_vertical_min"
                                                }
                                    },
}
```

In the aforementioned example, the objective is to verify the presence of the necessary attribute or attribute values in the XML file. If they are found, the `tds2stac_manual_variable` is to be added for the variable, namely the `vertical_axis` in this case.

3.1.2 Adding and Configuring Custom Extensions for STAC-Items and STAC-Collections

There exist two distinct methodologies for incorporating a custom extension into STAC-Items and STAC-Collections.

First approach:

If the extension is already present in the PySTAC package (refer to pySTAC extensions), it can be readily utilized by invoking the extension class and incorporating it into the STAC-Item or STAC-Collection, as elucidated below. For instance, we will incorporate the pySTAC scientific extension to the STAC-Item. To achieve this objective, it is necessary to configure the `tag_config.json` file in the following manner. To obtain further details on creating a ztag_config.jsonz file, please refer to the *Creating the tag_config.json Configuration File: A Step-by-Step Guide*.

```
{
"scientific_extension": {
                    "doi": {
                            "tds2stac_mode_analyser": "str",
                            "tds2stac_manual_variable": "10.1080/17550874.2023.2274839"
                            },
                    "citation": {
                            "tds2stac_mode_analyser": "str",
                            "tds2stac_manual_variable": "10.1080/17550874.2023.2274839"
                            },
                    "publications": {
                            "tds2stac_mode_analyser": "list",
                            "tds2stac_manual_variable": "[10.1080/17550874.2023.2274839,
↪'name']"

                            }
                    },

}
```

As observed in the aforementioned JSON file, the extension is denoted by the name "scientific_extension" and encompasses three distinct keys, namely "doi", "citation", and "publications". All keys of this extension are consistently represented as string and constant data types. For more information after incorporating the application into our harvesting_var dictionary, we will observe the presence of three distinct keys, namely `doi`, `citation`, and `publications`. The corresponding values for these keys are specified as the value of the `tds2stac_manual_variable` key.

```
harvesting_var = {
    "doi": "10.1080/17550874.2023.2274839",
    "citation": "10.1080/17550874.2023.2274839",
    "publications": "[10.1080/17550874.2023.2274839, 'name']",
}
```

In this stage, the values will be incorporated into the STAC-Items as a the extension that already exists within the PySTAC package. By employing two distinct methodologies, namely class and function, the provided script enables the seamless integration of code into the STAC-Item.

It is imperative to note that when defining a function or class, the input from the source code must include two parameters. The first parameter, referred to as `item`, represents the STAC-Item object to which we intend to add assets. The second parameter, known as `harvesting_vars`, represents a dictionary containing variables that have already been harvested and are to be added to the STAC-Item.

```
from pystac.extensions.scientific import Publication, ScientificExtension


# First
class Scientific:
    """
    a class-based custom extension
    for the item via the defined extension
    in pystac
    """

    def item(self, item, harvesting_vars):
        item_publication = []
        item_publication = [
            Publication(
                harvesting_vars["publications"][0],
                harvesting_vars["publications"][1],
            )
        ]
        scientific = ScientificExtension.ext(item, add_if_missing=True)
        scientific.apply(
            doi=harvesting_vars["doi"],
            citation=harvesting_vars["citation"],
            publications=item_publication,
        )


# Second
def item(item, harvesting_vars):
    """
    a function-based custom extension
    for the item via the defined extension
    in pystac
    """
    item_publication = []
    item_publication = [
        Publication(
            harvesting_vars["publications"][0],
            harvesting_vars["publications"][1],
        )
    ]
    scientific = ScientificExtension.ext(item, add_if_missing=True)
    scientific.apply(
        doi=harvesting_vars["doi"],
        citation=harvesting_vars["citation"],
        publications=item_publication,
    )
```

To execute the aforementioned script, it is necessary to invoke the TDS2STACIntegrator class in the following manner.

There are two distinct techniques available for this goal.

1. If the script mentioned above is saved in a separate file named `custom_extension.py`, it can be invoked in the third element of a tuple to execute it from that specific location.

```
TDS2STACIntegrator(
    "TDS_catalog_url",
    stac_dir="/path/to/stac_dir/",
    extension_properties={
        "item_extensions": [
            "common_metadata",
            (
                "scientific_extension",
                "item",  # or "Scientific.item"
                "/path/to/custom_extension.py",
            ),
        ]
    },
)
```

2. Alternatively, if the TDS2STACIntegrator calling script is a continuation of the aforementioned script, there is no requirement to include the script's address in the third element of the tuple. In this case, the tuple will consist of only two items.

```
TDS2STACIntegrator(
    "TDS_catalog_url",
    stac_dir="/path/to/stac_dir/",
    extension_properties={
        "item_extensions": [
            "common_metadata",
            (
                "scientific_extension",
                "item",  # or "Scientific.item"
            ),
        ]
    },
)
```

Second approach:

The second approach involves the definition of a custom extension, which is based on the STAC extensions list provided by the STAC extensions organization on Github. This process is outlined in the manual available in the pySTAC library. In order to fulfill this objective, a bespoke extension script was developed for the contact extension within the stac extension. This script was inspired by the guidelines provided in the pySTAC documentation.

```
from typing import Any, Dict, Literal, Union

import pystac
from pystac.extensions.base import (
    ExtensionManagementMixin,
    PropertiesExtension,
)
from pystac.utils import get_required, map_opt
```

(continues on next page)

(continued from previous page)

```
CONTACTS = "contacts"
# contact

NAME = "name"
ORGANIZATION = "organization"
IDENTIFIER = "identifier"
EMAILS = "emails"
PHONES = "phones"
POSITION = "position"
LOGO = "logo"
ADDRESSES = "addresses"
LINKS = "links"
CONTACTINSTRUCTIONS = "contactInstructions"
ROLES = "roles"

# Info

VALUE = "value"
ROLES = "roles"

# Address

DELIVERYPOINT = "deliveryPoint"
CITY = "city"
ADMINISTRATIVEAREA = "administrativeArea"
POSTALCODE = "postalCode"
COUNTRY = "country"

# Link

HREF = "href"
REL = "rel"
TYPE = "type"
TITLE = "title"


class Info:
    properties: Dict[str, Any]

    def __init__(self, properties: Dict[str, Any]) -> None:
        self.properties = properties

    @property
    def value(self) -> str:
        return get_required(self.properties.get(VALUE), self, VALUE)

    @value.setter
    def value(self, v: str) -> None:
        self.properties[VALUE] = v

    @property
```

(continues on next page)

(continued from previous page)

```
    def roles(self) -> list | None:
        return self.properties.get(ROLES)

    @roles.setter
    def roles(self, v: list | None) -> None:
        if v is None:
            self.properties.pop(ROLES, None)
        else:
            self.properties[ROLES] = v

    def to_dict(self) -> dict[str, Any]:
        return self.properties

    @staticmethod
    def from_dict(d: dict[str, str]) -> "Info":
        return Info(d.get("value"), d.get("roles"))  # type: ignore


class Address:
    properties: Dict[str, Any]

    def __init__(self, properties: Dict[str, Any]) -> None:
        self.properties = properties

    @property
    def deliveryPoint(self) -> list[str] | None:
        return self.properties.get(DELIVERYPOINT)

    @deliveryPoint.setter
    def deliveryPoint(self, v: list[str]) -> None:
        self.properties[DELIVERYPOINT] = v

    @property
    def city(self):
        return self.properties.get(CITY)

    @city.setter
    def city(self, v: str) -> None:
        self.properties[CITY] = v

    @property
    def administrativeArea(self):
        return self.properties.get(ADMINISTRATIVEAREA)

    @administrativeArea.setter
    def administrativeArea(self, v: str) -> None:
        self.properties[ADMINISTRATIVEAREA] = v

    @property
    def postalCode(self):
        return self.properties.get(POSTALCODE)
```

(continues on next page)

(continued from previous page)

```
    @postalCode.setter
    def postalCode(self, v: str) -> None:
        self.properties[POSTALCODE] = v

    @property
    def country(self):
        return self.properties.get(COUNTRY)

    @country.setter
    def country(self, v: str) -> None:
        self.properties[COUNTRY] = v

    def to_dict(self) -> dict[str, Any]:
        return self.properties

    @staticmethod
    def from_dict(d: dict[str, str]) -> "Address":
        return Address(
            d.get("deliveryPoint"),
            d.get("city"),
            d.get("administrativeArea"),
            d.get("postalCode"),
            d.get("country"),
        )


class Link:
    properties: Dict[str, Any]

    def __init__(self, properties: Dict[str, Any]) -> None:
        self.properties = properties

    @property
    def href(self) -> str:
        return get_required(self.properties.get(HREF), self, HREF)

    @href.setter
    def href(self, v: str) -> None:
        self.properties[HREF] = v

    @property
    def rel(self) -> str:
        return get_required(self.properties.get(REL), self, REL)

    @rel.setter
    def rel(self, v: str) -> None:
        self.properties[REL] = v

    @property
    def type(self):
        return self.properties.get(TYPE)
```

(continues on next page)

(continued from previous page)

```
    @type.setter
    def type(self, v: str) -> None:
        self.properties[TYPE] = v

    @property
    def title(self):
        return self.properties.get(TITLE)

    @title.setter
    def title(self, v: str) -> None:
        self.properties[TITLE] = v

    def to_dict(self) -> dict[str, Any]:
        return self.properties

    @staticmethod
    def from_dict(d: dict[str, str]) -> "Info":
        return Info(d.get("href"), d.get("rel"), d.get("type"), d.get("title"))  # type:␣
↪ignore


class Contact:
    properties: dict[str, str]

    def __init__(self, properties) -> None:
        self.properties = properties

    @property
    def name(self) -> str | None:
        return get_required(self.properties.get(NAME), self, NAME)

    @name.setter
    def name(self, v: str) -> None:
        self.properties[NAME] = v

    @property
    def organization(self) -> str | None:
        return get_required(self.properties.get(ORGANIZATION), self, NAME)

    @organization.setter
    def organization(self, v: str) -> None:
        self.properties[ORGANIZATION] = v

    @property
    def identifier(self) -> str | None:
        return self.properties.get(IDENTIFIER)

    @identifier.setter
    def identifier(self, v: str) -> None:
        self.properties[IDENTIFIER] = v

    @property
```

(continues on next page)

(continued from previous page)

```
    def position(self) -> str | None:
        return self.properties.get(POSITION)

    @position.setter
    def position(self, v: str) -> None:
        self.properties[POSITION] = v

    @property
    def logo(self) -> Link | None:
        return map_opt(Link.from_dict, self.properties.get(LOGO))

    @logo.setter
    def logo(self, v: Link | None) -> None:
        self.properties[LOGO] = map_opt(lambda link: link.to_dict(), v)

    @property
    def phones(self) -> list[Info] | None:
        return map_opt(
            lambda phones: [Info.from_dict(phone) for phone in phones],
            self.properties.get(PHONES),
        )

    @phones.setter
    def phones(self, v: list[Info] | None) -> None:
        self.properties[PHONES] = map_opt(
            lambda phones: [phone.to_dict() for phone in phones], v
        )

    @property
    def emails(self) -> list[Info] | None:
        return map_opt(
            lambda emails: [Info.from_dict(email) for email in emails],
            self.properties.get(EMAILS),
        )

    @emails.setter
    def emails(self, v: list[Info] | None) -> None:
        self.properties[EMAILS] = map_opt(
            lambda emails: [email.to_dict() for email in emails], v
        )

    @property
    def addresses(self) -> list[Address] | None:
        return map_opt(
            lambda addresses: [
                Address.from_dict(address) for address in addresses
            ],
            self.properties.get(ADDRESSES),
        )

    @addresses.setter
    def addresses(self, v: list[Address] | None) -> None:
```

(continues on next page)

(continued from previous page)

```
        self.properties[ADDRESSES] = map_opt(
            lambda addresses: [address.to_dict() for address in addresses],
            v,
        )

    @property
    def links(self) -> list[Link] | None:
        return map_opt(
            lambda links: [Link.from_dict(link) for link in links],
            self.properties.get(LINKS),
        )

    @links.setter
    def links(self, v: list[Link] | None) -> None:
        self.properties[LINKS] = map_opt(
            lambda links: [link.to_dict() for link in links], v
        )

    @property
    def contactInstructions(self) -> str | None:
        return self.properties.get(CONTACTINSTRUCTIONS)

    @contactInstructions.setter
    def contactInstructions(self, v: str) -> None:
        self.properties[CONTACTINSTRUCTIONS] = v

    @property
    def roles(self) -> str | None:
        return self.properties.get(ROLES)

    @roles.setter
    def roles(self, v: str) -> None:
        self.properties[ROLES] = v

    def to_dict(self) -> dict[str, Any]:
        return self.properties

    @staticmethod
    def from_dict(d: dict[str, str]) -> "Contact":
        return Contact(  # type: ignore
            d.get("name"),
            d.get("organization"),
            d.get("identifier"),
            d.get("position"),
            d.get("logo"),
            d.get("phones"),
            d.get("emails"),
            d.get("addresses"),
            d.get("links"),
            d.get("contactInstructions"),
            d.get("roles"),
        )
```

(continues on next page)

(continued from previous page)

```
SCHEMA_URI: str = (
    "https://stac-extensions.github.io/contacts/v0.1.1/schema.json"
)


class ContactsExtension(
    PropertiesExtension,
    ExtensionManagementMixin[Union[pystac.Item, pystac.Collection]],
):
    name: Literal["contacts"] = "contacts"
    obj: pystac.STACObject

    def __init__(self, item: pystac.Item):
        self.item = item
        self.properties = item.properties

    @classmethod
    def get_schema_uri(cls) -> str:
        return SCHEMA_URI

    def apply(
        self,
        contacts: list[Contact],
    ) -> None:
        self.contacts = contacts

    @property
    def contacts(self) -> list[Contact] | None:
        return map_opt(
            lambda conts: [Contact.from_dict(cont) for cont in conts],
            self._get_property(CONTACTS, list[dict[str, Any]]),
        )

    @contacts.setter
    def contacts(self, v: list[Contact] | None) -> None:
        self._set_property(
            CONTACTS,
            map_opt(lambda conts: [cont.to_dict() for cont in conts], v),
        )

    @classmethod
    def ext(
        cls, obj: pystac.Item, add_if_missing: bool = False
    ) -> "ContactsExtension":
        if isinstance(obj, pystac.Item):
            cls.validate_has_extension(obj, add_if_missing)
            return ContactsExtension(obj)
        else:
            raise pystac.ExtensionTypeError(
                f"ContactExtension does not apply to type '{type(obj).__name__}'"
```

(continues on next page)

(continued from previous page)

```
            )
```

Other steps are the same as the first approach.

3.1.3 How to create `extra_metadata.json` file

The dictionary has two primary keys, namely `collection` and `item`, which might have additional subkeys based on the definitions of pystac.Collection and pystac.Item.

Each collection in the STAC-Collection JSON file can have `keywords` and `providers` as list, and `licence` as a string at the top level. To incorporate additional fields into the STAC-Collection JSON file, we can append them to the `extra_fields` dictionary. The addition of `extra_fields` is possible within the Metadata section of the STAC-Collection JSON file.

To include pystac.common_metadata attributes in the STAC-Item JSON file, we can add them to the top level of each `item`. To incorporate additional fields into the STAC-Item JSON file, we can append them to the `properties` dictionary. The Metadata part of the STAC-Item JSON file allows for the inclusion of `properties`. It is important to mention that the `extra_fields` parameter can also be utilised to include more fields in the STAC-Item JSON file.

```
{
    "collection": {
        "extra_fields":{
            "anything": "anything",
            "version": ""
        },
        "keywords": ["something"],
        "license": "something",
        "providers": []

    },
    "item": {
        "extra_fields":{
            "anything": "anything",
            "something_else": "something"
        },
        "properties":{
            "anything": "anything",
            "something_else": "something"
        },
        "created": "It will fill out automatically if set extra_metadata to True",
        "description": "",
        "end_datetime": "It will fill out automatically if set extra_metadata to True",
        "geometry": {},
        "gsd": "",
        "instruments": [],
        "license": "",
        "platform": "",
        "providers": [],
        "start_datetime": "It will fill out automatically if set extra_metadata to True",
        "title": "",
        "updated": "It will fill out automatically if set extra_metadata to True",
    }
}
```

3.1.4 How to make a custom asset for STAC-Collection and STAC-Item:

This tutorial will demonstrate the process of creating a personalised asset for STAC-Collection and STAC-Item.

In order to create a customised asset, it is necessary to understand the composition of the `asset_properties` in both the STAC-Collection and STAC-Item. The *`asset_properties`* has a dictionary with keys that can be used to enable or disable certain services in TDS within the STAC-Collection and STAC-Item.

Two keys associated with custom assets are `collection_custom_asset` and `item_custom_asset`. Both situations use two keys, which are lists of dictionaries. Each dictionary within the lists contains the following keys:

1. `key`: This key is a string used to identify the custom asset.
2. `href`: This attribute is a string that specifies the URL of the custom asset.
3. `title`: This key is a string that specifies the custom asset's title.
4. The `roles` key is a list that specifies the custom asset's roles.

Note: The `roles` key is optional. If it is not specified, the default value is `["data"]`. For more information refer to the roless section of the PySTAC documentation.

Note: The `roles` key is a list. If you want to specify multiple roless, you can use the following format:

```
"roles": ["roles1", "roles2", "roles3", ...]
```

---

5. The `media_type` key is a string that specifies the media type of the custom asset.

Note: The `media_type` key is optional. If it is not specified, the default value is `None`. For more information refer to the Media Type section of the PySTAC documentation.

For example, if we want to make a custom asset for STAC-Collection and STAC-Item, we can use following dictionary:

```
asset_properties = {
            "collection_custom_asset": [
                {
                    "key": "key1",
                    "href": "href1",
                    "title": "title1",
                    "roles": ["roles1"],
                    "media_type": "media_type1",
                },
                {
                    "key": "key2",
                    "href": "href2",
                    "title": "title2",
                    "roles": ["roles2"],
                    "media_type": "media_type2",
                },
                {
                    "key": "key3",
                    "href": "href3",
```

(continues on next page)

(continued from previous page)

```
                "title": "title3",
                "roles": ["roles3"],
                "media_type": "media_type3",
            }
        ],
        "item_custom_asset": [
            {
                "key": "key1",
                "href": "href1",
                "title": "title1",
                "roles": ["roles1"],
                "media_type": "media_type1",
            },
            {
                "key": "key2",
                "href": "href2",
                "title": "title2",
                "roles": ["roles2"],
                "media_type": "media_type2",
            },
            {
                "key": "key3",
                "href": "href3",
                "title": "title3",
                "roles": ["roles3"],
                "media_type": "media_type3",
            }
        ]
    }
```

CHAPTER

FOUR

EXAMPLES

A bunch of examples is provided in this section.

4.1 Examples of use

In this section, we provide a few examples of use of `TDS2STC`. We listed them all in below:

Hint: You can run this notebook in a live session with .

- it is important to acknowledge that the utilization of the is restricted to individuals who possess the necessary credentials as Helmholtz users. It is highly recommended to utilize the `Python scipy` option when constructing the environment in order to mitigate the occurrence of a `404 Bad request` error.

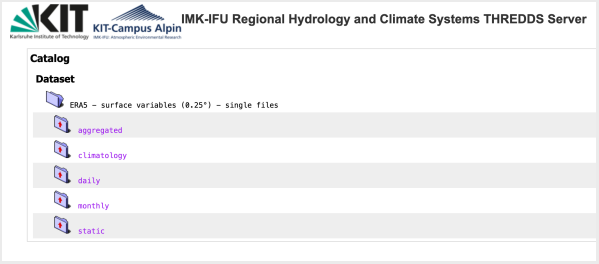
4.1.1 Recognizer

The classification of datasets in Thredds was conducted, resulting in the identification of nine distinct and plausible scenarios. This feature aims to identify the quantity of scenarios, while concurrently providing an assessment of the depth of nested datasets (i.e., datasets that contain subdirectories within them). There are a total of nine distinct scenarios, which are outlined as follows:

First scenario (Nested):

In this particular instance, the `catalogRef` tags are found just beneath the `dataset` element tag, without any separate data adjacent to the `catalogRef`s.

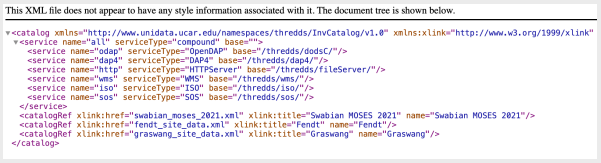
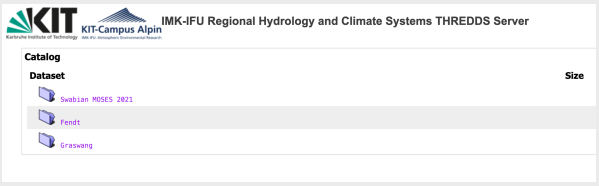
Example: https://thredds.imk-ifu.kit.edu/thredds/catalog/regclim/raster/global/era5/sfc/single/catalog.xml

XML-based	HTML-based
	

Second scenario (Nested):

The `catalogRef`s are positioned below the `catalog` tag element, rather than being nested within a `dataset` element tag. Additionally, there is no distinct data present adjacent to the `catalogRef`s.

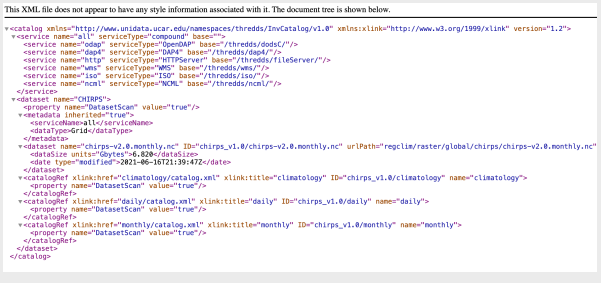
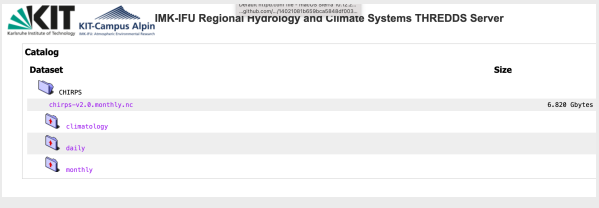
Example: https://thredds.imk-ifu.kit.edu/thredds/catalog/catalogues/sensor_catalog_ext.xml

XML-based	HTML-based
	

Third scenario (Nested):

The current scenario has a resemblance to the first scenario, albeit with the distinction that a distinct `dataset` tag is positioned adjacent to the `catalogRef` tags.

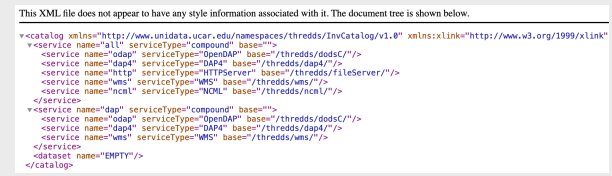
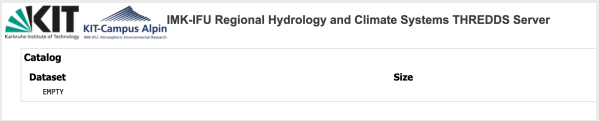
Example: https://thredds.imk-ifu.kit.edu/thredds/catalog/regclim/raster/global/chirps/catalog.xml

XML-based	HTML-based
	

Fourth scenario:

An empty dataset.

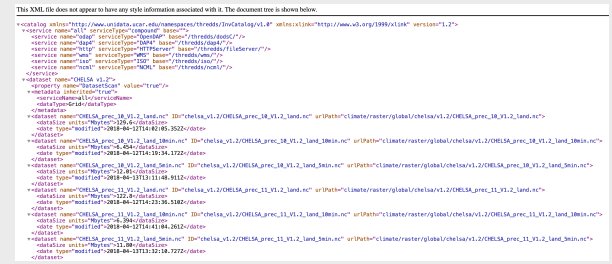
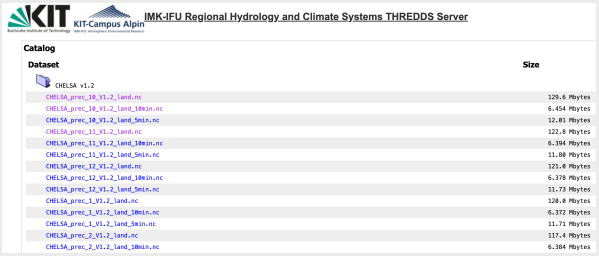
Example: https://thredds.imk-ifu.kit.edu/thredds/catalog/catalogues/bio_geo_chem_catalog_ext.xml

XML-based	HTML-based
	

Fifth scenario:

In this particular scenario, the `catalogRef` tag is absent, and all of the tags present are dataset tags. It means the parent `dataset` tag encompasses a collection of `dataset` tags.

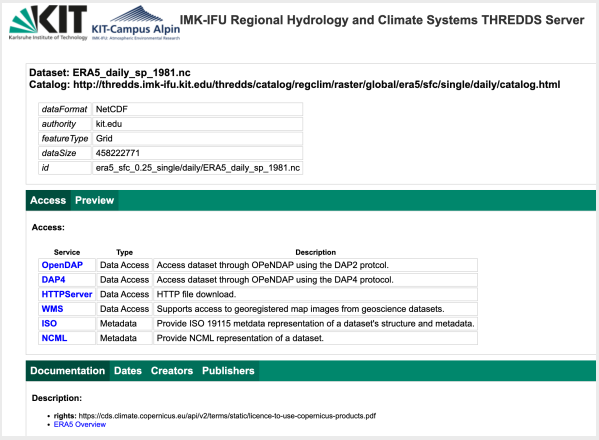
Example: https://thredds.imk-ifu.kit.edu/thredds/catalog/climate/raster/global/chelsa/v1.2/catalog.html

XML-based	HTML-based
	

Sixth scenario:

A single dataset.

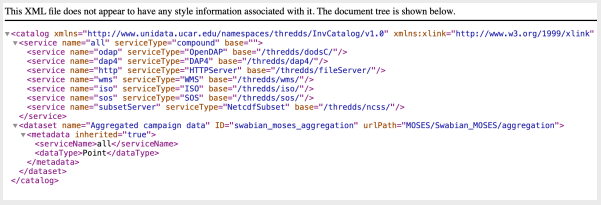
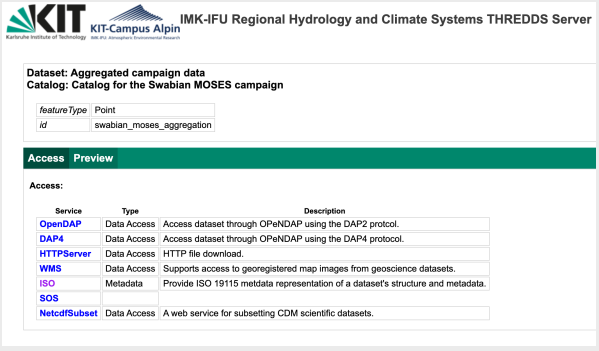
Example: https://thredds.imk-ifu.kit.edu/thredds/catalog/regclim/raster/global/era5/sfc/single/daily/catalog.html?dataset=era5_sfc_0.25_single/daily/ERA5_daily_sp_1981.nc

XML-based	HTML-based
	

Seventh scenario:

An aggregated dataset

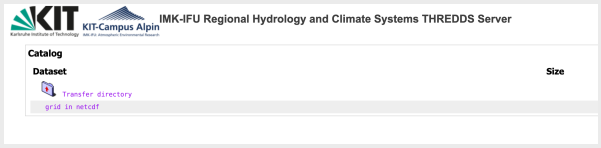
Example: https://thredds.imk-ifu.kit.edu/thredds/catalog/catalogues/swabian_moses_2021.xml?dataset=swabian_moses_aggregation

XML-based	HTML-based
	

Eighth scenario (Nested):

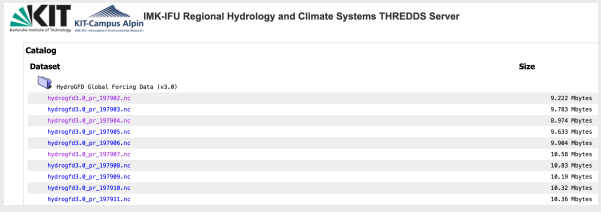
A configuration comprising of `catalogRef` elements and `dataset` tags that are not nested within a parent `dataset` tag. The situation has a resemblance to the second scenario, albeit with the notable distinction of the presence of distinct and individual dataset tags adjacent to `catalogRef`s.

Example: https://thredds.imk-ifu.kit.edu/thredds/catalog/catalogues/transfer.xml

XML-based	HTML-based
	

Ninth scenario (Nested):

This scenario pertains to the presence of more than one individual `dataset` tags located outside the parent dataset tag, adjacent to the `catalogRef` elements. The situation has a resemblance to the third scenario, albeit with the notable distinction that it encompasses multiple (more than one) distinct `dataset` tags in addition to the `catalogRef`s. Example: https://thredds.imk-ifu.kit.edu/thredds/catalog/regclim/raster/global/hydrogfd/v3.0/catalog.xml

XML-based	HTML-based
	

We have provided several examples to illustrate the functioning of the Recognizer here:

```
[ ]: from tds2stac import Recognizer

# First case
Recognizer(
    "https://thredds.imk-ifu.kit.edu/thredds/catalog/regclim/raster/global/era5/sfc/
↪single/catalog.html",
    nested_check=True,
)

# Second case

Recognizer(
    "https://thredds.imk-ifu.kit.edu/thredds/catalog/catalogues/sensor_catalog_ext.html",
    nested_check=True,
)

# Third case

Recognizer(
    "https://thredds.imk-ifu.kit.edu/thredds/catalog/regclim/raster/global/chirps/
↪catalog.html",
```

(continues on next page)

(continued from previous page)

```
    nested_check=True,
)


# Fourth case
Recognizer("https://thredds.imk-ifu.kit.edu/thredds/catalog/catalogues/bio_geo_chem_
↪catalog_ext.html")


# Fifth case
Recognizer("https://thredds.imk-ifu.kit.edu/thredds/catalog/climate/raster/global/chelsa/
↪v1.2/catalog.html")


# Sixth case
Recognizer("https://thredds.imk-ifu.kit.edu/thredds/catalog/regclim/raster/global/era5/
↪sfc/single/daily/catalog.html?dataset=era5_sfc_0.25_single/daily/ERA5_daily_sp_1979.nc
↪")


# Seven case
Recognizer("https://thredds.imk-ifu.kit.edu/thredds/catalog/catalogues/swabian_moses_
↪2021.html?dataset=swabian_moses_aggregation")


# Eighth case
Recognizer(
    "https://thredds.imk-ifu.kit.edu/thredds/catalog/catalogues/transfer.html",
    nested_check=True,
)


# Ninth case
Recognizer(
    "https://thredds.imk-ifu.kit.edu/thredds/catalog/regclim/raster/global/hydrogfd/v3.0/
↪catalog.html",
    nested_check=True,
)


# Finding random case
Recognizer(
    "https://thredds.imk-ifu.kit.edu/thredds/catalog/catalogues/reg_clim_sys_catalog_ext.
↪html",
    nested_check=True,
)
```

Output:

('First Scenario', 1) ('Second Scenario', 2) ('Third Scenario', 2) ('Fourth Scenario', 0) ('Fifth Scenario', 0) ('Sixth Scenario', 0) ('Seventh Scenario', 0) ('Eighth Scenario', 1) ('Ninth Scenario', 0) ('Second Scenario', 5)

Hint: You can run this notebook in a live session with .

- it is important to acknowledge that the utilization of the is restricted to individuals who possess the necessary credentials as Helmholtz users. It is highly recommended to utilize the `Python scipy` option when constructing the environment in order to mitigate the occurrence of a `404 Bad request` error.

4.1.2 Nested Collection

This class focuses on obtaining specific information from nested scenarios, such as those labeled as first, second, third, eighth, and ninth, based on the desired depth layer. The default depth layer is set to zero. The resulting output will consist of a collection of data elements, including the Collection-ID, Collection-Title, and the accompanying URLs. These data elements are associated with a nested numbering system that is assumed to originate from the Recognizer class in the TDS.

Here, we present several illustrative examples showcasing the utilization of this particular function.

[ ]:
```
from tds2stac import NestedCollectionInspector

# First scenario, depth 0
first_case_0 = NestedCollectionInspector(
    "https://thredds.imk-ifu.kit.edu/thredds/catalog/regclim/raster/global/era5/sfc/
↪single/catalog.html",
    nested_number=0,
)

# First scenario, depth greater than zero
first_case_gt_0 = NestedCollectionInspector(
    "https://thredds.imk-ifu.kit.edu/thredds/catalog/regclim/raster/global/era5/sfc/
↪single/catalog.html",
    nested_number=1,
)

# Second scenario, depth zero
second_case_0 = NestedCollectionInspector(
    "https://thredds.imk-ifu.kit.edu/thredds/catalog/catalogues/sensor_catalog_ext.html",
    nested_number=0,
)

# Second scenario, depth greater than zero
second_case_gt_0 = NestedCollectionInspector(
    "https://thredds.imk-ifu.kit.edu/thredds/catalog/catalogues/sensor_catalog_ext.html",
    nested_number=1,
)

# Third scenario, depth zero
third_case_0 = NestedCollectionInspector(
    "https://thredds.imk-ifu.kit.edu/thredds/catalog/regclim/raster/global/chirps/
↪catalog.html",
    nested_number=0,
)

# Third scenario, depth greater than zero
third_case_gt_0 = NestedCollectionInspector(
    "https://thredds.imk-ifu.kit.edu/thredds/catalog/regclim/raster/global/chirps/
```

(continues on next page)

(continued from previous page)

```
↪catalog.html",
    nested_number=1,
)

# Eighth scenario, depth zero
eighth_case_0 = NestedCollectionInspector(
    "https://thredds.imk-ifu.kit.edu/thredds/catalog/catalogues/transfer.html",
    nested_number=0,
)

# Eighth scenario, depth greater than zero
eighth_case_gt_0 = NestedCollectionInspector(
    "https://thredds.imk-ifu.kit.edu/thredds/catalog/catalogues/transfer.html",
    nested_number=1,
)

# Ninth scenario, depth zero
ninth_case_0 = NestedCollectionInspector(
    "https://thredds.imk-ifu.kit.edu/thredds/catalog/regclim/raster/global/hydrogfd/v3.0/
↪catalog.html",
    nested_number=0,
)

# Ninth scenario, depth greater than zero
ninth_case_gt_0 = NestedCollectionInspector(
    "https://thredds.imk-ifu.kit.edu/thredds/catalog/regclim/raster/global/hydrogfd/v3.0/
↪catalog.html",
    nested_number=1,
)
```

Output:

[('https://thredds.imk-ifu.kit.edu/thredds/catalog/regclim/raster/global/era5/sfc/single/catalog.xml', 'catalog$_regclim_raster_global_era5_sfc_single$', '$CatalogRegclimRasterGlobalEra5SfcSingle$', ['$https://thredds.imk-ifu.kit.edu/thredds/catalog/regclim/raster/global/era5/sfc/single/aggregated/catalog.xml$', '$https://thredds.imk-ifu.kit.edu/thredds/catalog/regclim/raster/global/era5/sfc/single/climatology/catalog.xml$', '$https://thredds.imk-ifu.kit.edu/thredds/catalog/regclim/raster/global/era5/sfc/single/daily/catalog.xml$', '$https://thredds.imk-ifu.kit.edu/thredds/catalog/regclim/raster/global/era5/sfc/single/monthly/catalog.xml$', '$https://thredds.imk-ifu.kit.edu/thredds/catalog/regclim/raster/global/era5/sfc/single/static/catalog.xml$'])]

[('https://thredds.imk-ifu.kit.edu/thredds/catalog/regclim/raster/global/era5/sfc/single/aggregated/catalog.xml', 'catalog$_regclim_raster_global_era5_sfc_single_aggregated$', '$CatalogRegclimRasterGlobalEra5SfcSingleAggregated$', []), ('$https://thredds.imk-ifu.kit.edu/thredds/catalog/regclim/raster/global/era5/sfc/single/climatology/catalog.xml$', '$catalog_regclim_r$
$https://thredds.imk-ifu.kit.edu/thredds/catalog/regclim/raster/global/era5/sfc/single/daily/catalog.xml$', '$catalog_regclim_r$
$https://thredds.imk-ifu.kit.edu/thredds/catalog/regclim/raster/global/era5/sfc/single/monthly/catalog.xml$', '$catalog_regcli$
$https://thredds.imk-ifu.kit.edu/thredds/catalog/regclim/raster/global/era5/sfc/single/static/catalog.xml$', '$catalog_regclim_r$

[('https://thredds.imk-ifu.kit.edu/thredds/catalog/catalogues/sensor$_catalog_ext.xml$', '$catalog_catalogues_sensor_catalog_ext$', '$CatalogC$
$https://thredds.imk-ifu.kit.edu/thredds/catalog/MOSES/Swabian_MOSES/single_files/catalog.xml$', '$https://thredds.imk-ifu.kit.edu/thredds/catalog/catalogues/swabian_moses_2021.xml?dataset=swabian_moses_aggregation$', '$https://thredds.imk-ifu.kit.edu/thredds/catalog/fendt/ec/single_files/catalog.xml$', '$https://thredds.imk-ifu.kit.edu/thredds/catalog/catalogues/fendt_site_data.xml?dataset=fendt_eca_aggregation$', '$https://thredds.imk-ifu.kit.edu/thredds/catalog/graswang/ec/single_files/catalog.xml$', '$https://thredds.imk-ifu.kit.edu/thredds/catalog/catalogues/graswang_site_data.xml?dataset=$

$graswang_e c_a ggregation']$)]

[('https://thredds.imk-ifu.kit.edu/thredds/catalog/catalogues/swabian$_m oses_2 021.xml$','$catalog_c atalogues_s wabian_m oses_2 021$','$Cata$
$//thredds.imk-ifu.kit.edu/thredds/catalog/MOSES/Swabian_M OSES/single_f iles/catalog.xml$','$https:$
$//thredds.imk - ifu.kit.edu/thredds/catalog/catalogues/swabian_m oses_2 021.xml?dataset =$
$swabian_m oses_a ggregation']), ('https://thredds.imk-ifu.kit.edu/thredds/catalog/catalogues/fendt_s ite_d ata.xml$','$catalog$
$//thredds.imk - ifu.kit.edu/thredds/catalog/fendt/ec/single_f iles/catalog.xml$','$https://thredds.imk -$
$ifu.kit.edu/thredds/catalog/catalogues/fendt_s ite_d ata.xml?dataset = fendt_e c_a ggregation']), ('https:$
$//thredds.imk-ifu.kit.edu/thredds/catalog/catalogues/graswang_s ite_d ata.xml$','$catalog_c atalogues_g raswang_s ite_d ata$','$Ca$
$//thredds.imk - ifu.kit.edu/thredds/catalog/graswang/ec/single_f iles/catalog.xml$','$https:$
$//thredds.imk - ifu.kit.edu/thredds/catalog/catalogues/graswang_s ite_d ata.xml?dataset =$
$graswang_e c_a ggregation']$)]

[('https://thredds.imk-ifu.kit.edu/thredds/catalog/regclim/raster/global/chirps/catalog.xml',
'catalog$_r egclim_r aster_g lobal_c hirps$','$CatalogRegclimRasterGlobalChirps$',['$https://thredds.imk -$
$ifu.kit.edu/thredds/catalog/regclim/raster/global/chirps/climatology/0.05/catalog.xml$','$https:$
$//thredds.imk-ifu.kit.edu/thredds/catalog/regclim/raster/global/chirps/climatology/0.1/catalog.xml$','$https:$
$//thredds.imk-ifu.kit.edu/thredds/catalog/regclim/raster/global/chirps/daily/0.05/catalog.xml$','$https:$
$//thredds.imk-ifu.kit.edu/thredds/catalog/regclim/raster/global/chirps/daily/0.1/catalog.xml$','$https:$
$//thredds.imk-ifu.kit.edu/thredds/catalog/regclim/raster/global/chirps/monthly/0.05/catalog.xml$','$https:$
$//thredds.imk - ifu.kit.edu/thredds/catalog/regclim/raster/global/chirps/catalog.xml?dataset =$
$chirps_v 1.0/chirps - v2.0.monthly.nc']$)]

[('https://thredds.imk-ifu.kit.edu/thredds/catalog/regclim/raster/global/chirps/climatology/catalog.xml',
'catalog$_r egclim_r aster_g lobal_c hirps_c limatology$','$CatalogRegclimRasterGlobalChirpsClimatology$',['$https:$
$//thredds.imk-ifu.kit.edu/thredds/catalog/regclim/raster/global/chirps/climatology/0.05/catalog.xml$','$https:$
$//thredds.imk-ifu.kit.edu/thredds/catalog/regclim/raster/global/chirps/climatology/0.1/catalog.xml']), ('https:$
$//thredds.imk-ifu.kit.edu/thredds/catalog/regclim/raster/global/chirps/daily/catalog.xml$','$catalog_r egclim_r aster_g loba$
$//thredds.imk-ifu.kit.edu/thredds/catalog/regclim/raster/global/chirps/daily/0.05/catalog.xml$','$https:$
$//thredds.imk-ifu.kit.edu/thredds/catalog/regclim/raster/global/chirps/daily/0.1/catalog.xml']), ('https:$
$//thredds.imk-ifu.kit.edu/thredds/catalog/regclim/raster/global/chirps/monthly/catalog.xml$','$catalog_r egclim_r aster_g l$
$//thredds.imk-ifu.kit.edu/thredds/catalog/regclim/raster/global/chirps/monthly/0.05/catalog.xml']), ('https:$
$//thredds.imk - ifu.kit.edu/thredds/catalog/regclim/raster/global/chirps/catalog.xml?dataset =$
$chirps_v 1.0/chirps-v2.0.monthly.nc$','$catalog_r egclim_r aster_g lobal_c hirps_d ataset_c hirps_v 1_{0c} hirps_v 2_{0m} onthly$','$CatalogRegclim$

[('https://thredds.imk-ifu.kit.edu/thredds/catalog/catalogues/transfer.xml', 'catalog$_c atalogues_t ransfer$','$CatalogCataloguesTransf$
$//thredds.imk - ifu.kit.edu/thredds/catalog/transfer/catalog.xml$','$https://thredds.imk -$
$ifu.kit.edu/thredds/catalog/catalogues/transfer.xml?dataset = s3_t est_g pcc_n etcdf']$)]

[('https://thredds.imk-ifu.kit.edu/thredds/catalog/transfer/catalog.xml', 'catalog$_t ransfer$','$CatalogTransfer$', []), ('https:$
$//thredds.imk - ifu.kit.edu/thredds/catalog/catalogues/transfer.xml?dataset =$
$s3_t est_g pcc_n etcdf$','$catalog_c atalogues_t ransfer_d ataset_s 3_t est_g pcc_n etcdf$','$CatalogCataloguesTransferDatasetS3TestGpccNe$

[]

[]

Hint: You can run this notebook in a live session with .

- it is important to acknowledge that the utilization of the is restricted to individuals who possess the necessary credentials as Helmholtz users. It is highly recommended to utilize the `Python scipy` option when constructing the environment in order to mitigate the occurrence of a `404 Bad request` error.

4.1.3 TDS2STACIntegrator

In this section, we present a set of illustrative examples that demonstrate the process of working with TDS2STAC.

Harvesting and creating without any additional extension or asset:

In order to achieve this objective, it is necessary to complete two distinct arguments while the remaining arguments will be automatically populated.

- TDS_catalog
- stac_dir

In the present example, we intend to extract data from the dataset located at IMK-IFU Thredds and thereafter save the generated STAC-metadata in the "/stac" directory. We have incorporated a limited_number parameter to choose extract a single data point from the collection. It should be mentioned the primary objective of limited_number is to facilitate unit-testing and development activities.

[ ]:
```
from tds2stac import TDS2STACIntegrator

TDS2STACIntegrator(
    TDS_catalog="https://thredds.imk-ifu.kit.edu/thredds/catalog/climate/raster/global/
↪chelsa/v1.2/catalog.html",
    stac_dir="stac/",
    limited_number=1,
    logger_properties={"logger_handler":"StreamHandler"},
)
```

Output:

```
stac/
└── stac
    ├── catalog.json
    └── catalog_climate_raster_global_chelsa_v1_2_catalog_html_collection
        ├── chelsa_v1_2_chelsa_prec_10_v1_2_land
        │   └── chelsa_v1_2_chelsa_prec_10_v1_2_land.json
        └── collection.json

4 directories, 3 files
```

Verify the presence of a catalog in the directory

This functionality is intended for scenarios when a STAC-Collections or STAC-Item is available and requires updating. As demonstrated in the preceding example, a STAC-Catalog was generated and assigned to the `/stac` directory. We seek to modify the previously established catalog and collection in the following manner. Prior to making any updates, it is advisable to review the STAC-Catalog and STAC-Collection.

The JSON file provided exhibits the structure of STAC-Catalog and STAC-Collections. As an illustrative instance, we aim to initially examine the STAC-Metadata and thereafter employ the stac_existence and stac_existence_collection

STAC-Catalog:

```
{
  "type": "Catalog",
```

(continues on next page)

(continued from previous page)


```
  "id": "TDS2STAC",
  "stac_version": "1.0.0",
  "description": "[This is a STAC catalog created by tds2stac](https://thredds.imk-ifu.
↪kit.edu/thredds/catalog/climate/raster/global/chelsa/v1.2/catalog.html)",
  "links": [
    {
      "rel": "root",
      "href": "./catalog.json",
      "type": "application/json",
      "title": "TDS2STAC"
    },
    {
      "rel": "child",
      "href": "./catalog_climate_raster_global_chelsa_v1_2_catalog_html_collection/
↪collection.json",
      "type": "application/json",
      "title": "Catalog Climate Raster Global Chelsa V1 2 Catalog Html Collection"
    }
  ],
  "title": "TDS2STAC"
}
```


STAC-Collection:


```
{
  "type": "Collection",
  "id": "catalog_climate_raster_global_chelsa_v1_2_catalog_html_collection",
  "stac_version": "1.0.0",
  "description": "[Link to TDS](https://thredds.imk-ifu.kit.edu/thredds/catalog/climate/
↪raster/global/chelsa/v1.2/catalog.html)",
  "links": [
    {
      "rel": "root",
      "href": "../catalog.json",
      "type": "application/json",
      "title": "TDS2STAC"
    },
    {
      "rel": "item",
      "href": "./chelsa_v1_2_chelsa_prec_10_v1_2_land/chelsa_v1_2_chelsa_prec_10_v1_2_
↪land.json",
      "type": "application/json"
    },
    {
      "rel": "parent",
      "href": "../catalog.json",
      "type": "application/json",
      "title": "TDS2STAC"
    }
  ],
  "title": "Catalog Climate Raster Global Chelsa V1 2 Catalog Html Collection",
  "extent": {
```


(continues on next page)

(continued from previous page)


```
    "spatial": {
      "bbox": [
        [
          -179.99597,
          -89.99597,
          179.99569,
          83.99569
        ]
      ]
    },
    "temporal": {
      "interval": [
        [
          "1996-10-01T01:10:36Z",
          "1996-10-01T01:10:36Z"
        ]
      ]
    }
  },
  "license": "proprietary"
}
```


In the current setting, the utilization of stac_existence and stac_existence_collection is employed to facilitate the process of updating the catalog and collections. In this part, the value of limited_number is modified to 2 in order to see the disparities within the STAC-Collection. Additionally, another collection is harvested to assess the distinctions present within the STAC-Catalog.

[ ]:
```
from tds2stac import TDS2STACIntegrator

TDS2STACIntegrator(
    TDS_catalog="https://thredds.imk-ifu.kit.edu/thredds/catalog/climate/raster/global/
↪chelsa/v1.2/catalog.html",
    stac_dir="stac/",
    limited_number=2,
    stac_existence=True,
    stac_existence_collection=True,
    logger_properties={"logger_handler":"StreamHandler"},
)

TDS2STACIntegrator(
    TDS_catalog="https://thredds.imk-ifu.kit.edu/thredds/catalog/regclim/raster/regional/
↪eobs/v23.1e/catalog.html",
    stac_dir="stac/",
    limited_number=1,
    stac_existence=True,
    stac_existence_collection=True,
    logger_properties={"logger_handler":"StreamHandler"},
)
```

Output:

```
stac/
```

(continues on next page)

(continued from previous page)

```
└── stac
    ├── catalog.json
    ├── catalog_climate_raster_global_chelsa_v1_2_catalog_html_collection
    │   ├── chelsa_v1_2_chelsa_prec_10_v1_2_land
    │   │   └── chelsa_v1_2_chelsa_prec_10_v1_2_land.json
    │   ├── chelsa_v1_2_chelsa_prec_10_v1_2_land_10min
    │   │   └── chelsa_v1_2_chelsa_prec_10_v1_2_land_10min.json
    │   └── collection.json
    └── catalog_regclim_raster_regional_eobs_v23_1e_catalog_html_collection
        ├── collection.json
        └── eobs_daily_0_1_v23_1e_pp_ens_mean_0_1deg_reg_v23_1e
            └── eobs_daily_0_1_v23_1e_pp_ens_mean_0_1deg_reg_v23_1e.json

7 directories, 6 files
```

Change the ID, name, and description of STAC-Collections

To modify the information of STAC-Collections, the initial step is executing the `NestedCollectionInspector` to identify the automated Collection IDs. Subsequently, the auto-generated IDs serve as the basis for making the desired changes to all relevant collections. However, prior to utilizing the NestedCollectionInspector class, it is necessary to determine the depth of the dataset catalog in order to organize the collections into a given layer. In order to accomplish this objective, it is necessary to utilize the Recognizer to get the deepest layer number and subsequently employ the NestedCollectionInspector. Once the dataset depth has been determined and a specific layer has been selected, and in the case of having five distinct STAC-Collections in our STAC-Catalog harvesting process, if there is a need to modify the ID, name, and description of two of these collections, the following steps should be taken.

Firstly, we must index two auto-generated IDs as the first element of tuples within a list of tuples (collection_tuples). Subsequently, the desired ID should be added as the second element of each tuple, while the name and description should be added as the third and fourth elements, respectively.

To be exact, we can have a look at the following example.

[ ]:
```
from tds2stac import Recognizer, NestedCollectionInspector, TDS2STACIntegrator

print("Finding depth of the dataset:")
Recognizer(
    main_catalog_url="https://thredds.atmohub.kit.edu/thredds/catalog/icon/ACCMIP/
↪catalog.html",
    nested_check=True,
    logger_properties={
        "logger_handler": "StreamHandler",
    },
)

print("Zero depth of dataset:")
NestedCollectionInspector(
    main_catalog_url="https://thredds.atmohub.kit.edu/thredds/catalog/icon/ACCMIP/
↪catalog.html",
    nested_number=0,  # by default the depth layer is zero. In these situation there is␣
↪no need to write the zero down.
    logger_properties={
        "logger_handler": "StreamHandler",
```

(continues on next page)

(continued from previous page)

```
    },
)
print("First depth of dataset:")
NestedCollectionInspector(
    main_catalog_url="https://thredds.atmohub.kit.edu/thredds/catalog/icon/ACCMIP/
↪catalog.html",
    nested_number=1,
    logger_properties={
        "logger_handler": "StreamHandler",
    },
)
print("Second depth of dataset:")
NestedCollectionInspector(
    main_catalog_url="https://thredds.atmohub.kit.edu/thredds/catalog/icon/ACCMIP/
↪catalog.html",
    nested_number=2,
    logger_properties={
        "logger_handler": "StreamHandler",
    },
)
print("Third depth of dataset:")
NestedCollectionInspector(
    main_catalog_url="https://thredds.atmohub.kit.edu/thredds/catalog/icon/ACCMIP/
↪catalog.html",
    nested_number=3,
    logger_properties={
        "logger_handler": "StreamHandler",
    },
)
```

Output:

Finding depth of the dataset:

INFO 2023-11-15 12:10:56,495 logger.py @function **init** line 402 - ('First Scenario', 4)

Zero depth of dataset:

INFO 2023-11-15 12:12:55,736 logger.py @function **init** line 402 - [('https://thredds.atmohub.kit.edu/thredds/catalog/icon/ACCMIP/catalog.xml', 'catalog_icon_accmip', 'Catalog Icon Accmip', [...])]

First depth of dataset:

INFO 2023-11-15 12:15:56,435 logger.py @function **init** line 402 - [('https://thredds.atmohub.kit.edu/thredds/catalog/icon/ACCMIP/C2H6/catalog.xml', 'catalog_icon_accmip_c2h6', 'Catalog Icon Accmip C2H6', [...]), ('https://thredds.atmohub.kit.edu/thredds/catalog/icon/ACCMIP/CH3COCH3/catalog.xml', 'catalog_icon_accmip_ch3coch3', 'Catalog Icon Accmip Ch3Coch3', [...]), ('https://thredds.atmohub.kit.edu/thredds/catalog/icon/ACCMIP/CH4/catalog.xml', 'catalog_icon_accmip_ch4', 'Catalog Icon Accmip Ch4', [...]), ('https://thredds.atmohub.kit.edu/thredds/catalog/icon/ACCMIP/CO/catalog.xml', 'catalog_icon_accmip_co', 'Catalog Icon Accmip Co', [...]), ('https://thredds.atmohub.kit.edu/thredds/catalog/icon/ACCMIP/NOx/catalog.xml', 'catalog_icon_accmip_nox', 'Catalog Icon Accmip Nox', [...])]

Second depth of dataset:

INFO 2023-11-15 12:18:51,031 logger.py @function **init** line 402 - [('https://thredds.atmohub.kit.edu/thredds/catalog/icon/ACCMIP/C2H6/ANT/catalog.xml', 'catalog_icon_accmip_c2h6_ant', 'Catalog Icon Accmip C2H6

Ant', [...]), ('https://thredds.atmohub.kit.edu/thredds/catalog/icon/ACCMIP/C2H6/BBE/catalog.xml', 'catalog_icon_accmip_c2h6_bbe', 'Catalog Icon Accmip C2H6 Bbe', [...]), ('https://thredds.atmohub.kit.edu/thredds/catalog/icon/ACCMIP/CH3COCH3/BBE/catalog.xml', 'catalog_icon_accmip_ch3coch3_bbe', 'Catalog Icon Accmip Ch3Coch3 Bbe', [...]), ('https://thredds.atmohub.kit.edu/thredds/catalog/icon/ACCMIP/CH4/ANT/catalog.xml', 'catalog_icon_accmip_ch4_ant', 'Catalog Icon Accmip Ch4 Ant', [...]), ('https://thredds.atmohub.kit.edu/thredds/catalog/icon/ACCMIP/CH4/BBE/catalog.xml', 'catalog_icon_accmip_ch4_bbe', 'Catalog Icon Accmip Ch4 Bbe', [...]), ('https://thredds.atmohub.kit.edu/thredds/catalog/icon/ACCMIP/CO/ANT/catalog.xml', 'catalog_icon_accmip_co_ant', 'Catalog Icon Accmip Co Ant', [...]), ('https://thredds.atmohub.kit.edu/thredds/catalog/icon/ACCMIP/CO/BBE/catalog.xml', 'catalog_icon_accmip_co_bbe', 'Catalog Icon Accmip Co Bbe', [...]), ('https://thredds.atmohub.kit.edu/thredds/catalog/icon/ACCMIP/NOx/ANT/catalog.xml', 'catalog_icon_accmip_nox_ant', 'Catalog Icon Accmip Nox Ant', [...]), ('https://thredds.atmohub.kit.edu/thredds/catalog/icon/ACCMIP/NOx/BBE/catalog.xml', 'catalog_icon_accmip_nox_bbe', 'Catalog Icon Accmip Nox Bbe', [...])]

Third depth of dataset:

INFO 2023-11-15 12:21:43,703 logger.py @function **init** line 402 - [('https://thredds.atmohub.kit.edu/thredds/catalog/icon/ACCMIP/C2H6/ANT/RCP26/catalog.xml', 'catalog_icon_accmip_c2h6_ant_rcp26', 'Catalog Icon Accmip C2H6 Ant Rcp26', [...]), ('https://thredds.atmohub.kit.edu/thredds/catalog/icon/ACCMIP/C2H6/ANT/RCP26_ships/catalog.xml', 'catalog_icon_accmip_c2h6_ant_rcp26_ships', 'Catalog Icon Accmip C2H6 Ant Rcp26 Ships', [...]), ('https://thredds.atmohub.kit.edu/thredds/catalog/icon/ACCMIP/C2H6/ANT/hist/catalog.xml', 'catalog_icon_accmip_c2h6_ant_hist', 'Catalog Icon Accmip C2H6 Ant Hist', [...]), ('https://thredds.atmohub.kit.edu/thredds/catalog/icon/ACCMIP/C2H6/ANT/hist_ships/catalog.xml', 'catalog_icon_accmip_c2h6_ant_hist_ships', 'Catalog Icon Accmip C2H6 Ant Hist Ships', [...]), ('https://thredds.atmohub.kit.edu/thredds/catalog/icon/ACCMIP/C2H6/BBE/RCP26/catalog.xml', 'catalog_icon_accmip_c2h6_bbe_rcp26', 'Catalog Icon Accmip C2H6 Bbe Rcp26', [...]), ('https://thredds.atmohub.kit.edu/thredds/catalog/icon/ACCMIP/C2H6/BBE/hist/catalog.xml', 'catalog_icon_accmip_c2h6_bbe_hist', 'Catalog Icon Accmip C2H6 Bbe Hist', [...]), ('https://thredds.atmohub.kit.edu/thredds/catalog/icon/ACCMIP/CH3COCH3/BBE/RCP26/catalog.xml', 'catalog_icon_accmip_ch3coch3_bbe_rcp26', 'Catalog Icon Accmip Ch3Coch3 Bbe Rcp26', [...]), ('https://thredds.atmohub.kit.edu/thredds/catalog/icon/ACCMIP/CH3COCH3/BBE/hist/catalog.xml', 'catalog_icon_accmip_ch3coch3_bbe_hist', 'Catalog Icon Accmip Ch3Coch3 Bbe Hist', [...]), ('https://thredds.atmohub.kit.edu/thredds/catalog/icon/ACCMIP/CH4/ANT/RCP26/catalog.xml', 'catalog_icon_accmip_ch4_ant_rcp26', 'Catalog Icon Accmip Ch4 Ant Rcp26', [...]), ('https://thredds.atmohub.kit.edu/thredds/catalog/icon/ACCMIP/CH4/ANT/RCP26_ships/catalog.xml', 'catalog_icon_accmip_ch4_ant_rcp26_ships', 'Catalog Icon Accmip Ch4 Ant Rcp26 Ships', [...]), ('https://thredds.atmohub.kit.edu/thredds/catalog/icon/ACCMIP/CH4/ANT/hist/catalog.xml', 'catalog_icon_accmip_ch4_ant_hist', 'Catalog Icon Accmip Ch4 Ant Hist', [...]), ('https://thredds.atmohub.kit.edu/thredds/catalog/icon/ACCMIP/CH4/ANT/hist_ships/catalog.xml', 'catalog_icon_accmip_ch4_ant_hist_ships', 'Catalog Icon Accmip Ch4 Ant Hist Ships', [...]), ('https://thredds.atmohub.kit.edu/thredds/catalog/icon/ACCMIP/CH4/BBE/RCP26/catalog.xml', 'catalog_icon_accmip_ch4_bbe_rcp26', 'Catalog Icon Accmip Ch4 Bbe Rcp26', [...]), ('https://thredds.atmohub.kit.edu/thredds/catalog/icon/ACCMIP/CH4/BBE/hist/catalog.xml', 'catalog_icon_accmip_ch4_bbe_hist', 'Catalog Icon Accmip Ch4 Bbe Hist', [...]), ('https://thredds.atmohub.kit.edu/thredds/catalog/icon/ACCMIP/CO/ANT/RCP26/catalog.xml', 'catalog_icon_accmip_co_ant_rcp26', 'Catalog Icon Accmip Co Ant Rcp26', [...]), ('https://thredds.atmohub.kit.edu/thredds/catalog/icon/ACCMIP/CO/ANT/RCP26_ships/catalog.xml', 'catalog_icon_accmip_co_ant_rcp26_ships', 'Catalog Icon Accmip Co Ant Rcp26 Ships', [...]), ('https://thredds.atmohub.kit.edu/thredds/catalog/icon/ACCMIP/CO/ANT/hist/catalog.xml', 'catalog_icon_accmip_co_ant_hist', 'Catalog Icon Accmip Co Ant Hist', [...]), ('https://thredds.atmohub.kit.edu/thredds/catalog/icon/ACCMIP/CO/ANT/hist_ships/catalog.xml', 'catalog_icon_accmip_co_ant_hist_ships', 'Catalog Icon Accmip Co Ant Hist Ships', [...]), ('https://thredds.atmohub.kit.edu/thredds/catalog/icon/ACCMIP/CO/BBE/RCP26/catalog.xml', 'catalog_icon_accmip_co_bbe_rcp26', 'Catalog Icon Accmip Co Bbe Rcp26', [...]), ('https://thredds.atmohub.kit.edu/thredds/catalog/icon/ACCMIP/CO/BBE/hist/catalog.xml', 'catalog_icon_accmip_co_bbe_hist', 'Catalog Icon Accmip Co Bbe Hist', [...]), ('https://thredds.atmohub.kit.edu/thredds/catalog/icon/ACCMIP/NOx/ANT/RCP26/catalog.xml', 'catalog_icon_accmip_nox_ant_rcp26', 'Catalog Icon Accmip Nox Ant Rcp26', [...]), ('https://thredds.atmohub.kit.edu/thredds/catalog/icon/ACCMIP/NOx/ANT/RCP26_aircraft/catalog.xml', 'catalog_icon_accmip_nox_ant_rcp26_aircraft', 'Catalog Icon Accmip Nox Ant Rcp26 Aircraft', [...]), ('https://thredds.atmohub.kit.edu/thredds/catalog/icon/ACCMIP/NOx/ANT/RCP26_ships/catalog.xml', 'catalog_icon_accmip_nox_ant_rcp26_ships', 'Catalog Icon Accmip Nox Ant Rcp26

Ships', [...]), ('https://thredds.atmohub.kit.edu/thredds/catalog/icon/ACCMIP/NOx/ANT/hist/catalog.xml', 'catalog_icon_accmip_nox_ant_hist', 'Catalog Icon Accmip Nox Ant Hist', [...]), ('https://thredds.atmohub.kit.edu/thredds/catalog/icon/ACCMIP/NOx/ANT/hist_aircraft/catalog.xml', 'catalog_icon_accmip_nox_ant_hist_aircraft', 'Catalog Icon Accmip Nox Ant Hist Aircraft', [...]), ('https://thredds.atmohub.kit.edu/thredds/catalog/icon/ACCMIP/NOx/ANT/hist_ships/catalog.xml', 'catalog_icon_accmip_nox_ant_hist_ships', 'Catalog Icon Accmip Nox Ant Hist Ships', [...]), ('https://thredds.atmohub.kit.edu/thredds/catalog/icon/ACCMIP/NOx/BBE/RCP26/catalog.xml', 'catalog_icon_accmip_nox_bbe_rcp26', 'Catalog Icon Accmip Nox Bbe Rcp26', [...]), ('https://thredds.atmohub.kit.edu/thredds/catalog/icon/ACCMIP/NOx/BBE/hist/catalog.xml', 'catalog_icon_accmip_nox_bbe_hist', 'Catalog Icon Accmip Nox Bbe Hist', [...])]

Now, we possess knowledge on the default ID and name of Collections, which is determined by the depth of the dataset. Consequently, we have the ability to modify either a single or all of these default values. This can be achieved by implementing the following changes:

[ ]:
```
print("Using `collection_tuples` to change the STAC-Collection details in zero dataset␣
↪depth")
TDS2STACIntegrator(
    TDS_catalog="https://thredds.imk-ifu.kit.edu/thredds/catalog/regclim/raster/regional/
↪eobs/v23.1e/catalog.html",
    stac_dir="stac/",
    depth_number=0,
    limited_number=1,
    collection_tuples = [("catalog_icon_accmip",
                          "user-defined_catalog_icon_accmip",
                          "user-defined Catalog Icon Accmip",
                          "This is a description created by user")],
    logger_properties={
        "logger_handler": "StreamHandler",
    },
)
print("Using `collection_tuples` to change the STAC-Collection details in second dataset␣
↪depth")
TDS2STACIntegrator(
    TDS_catalog="https://thredds.imk-ifu.kit.edu/thredds/catalog/regclim/raster/regional/
↪eobs/v23.1e/catalog.html",
    stac_dir="stac/",
    limited_number=1,
    depth_number=2,
    collection_tuples = [("catalog_icon_accmip_c2h6_ant",
                          "user-defined_catalog_icon_accmip_c2h6_ant",
                          "user-defined Catalog Icon Accmip C2H6 Ant",
                          "This is a description created by user"),
                         ("catalog_icon_accmip_co_bbe",
                          "user-defined_catalog_icon_accmip_co_bbe",
                          "user-defined Catalog Icon Accmip Co Bbe",
                          "This is a description created by user")],
    logger_properties={
        "logger_handler": "StreamHandler",
    },
)
```

Date Time Filter

In this section, the objective is to apply a temporal filter (datetime_filter) to the dataset within a specified catalog. The outcome will be the creation of a new STAC-Catalog and STAC-Collections, or an update of existing ones. The purpose of this argument is to provide a strategy that can be employed in order to prevent the duplication of efforts in catalog harvesting, hence expediting the process of generating STAC-Metadata. By setting the variables stac_existence and stac_existence_collection to True, it becomes possible to verify the presence of the STAC-Catalog and STAC-Collections in the local storage and TDS catalog `modified_datetime` simultaneously.

Caution should be exercised while utilizing the limited_number parameter in order to prevent any conflicts in obtaining datetime-based data.

To illustrate this point, we have provided an example below:

```
[ ]: from tds2stac import TDS2STACIntegrator
TDS2STACIntegrator(
    TDS_catalog="https://thredds.imk-ifu.kit.edu/thredds/catalog/regclim/raster/global/
↪era5_land/daily/catalog.html",
    stac_dir="stac/",
    datetime_filter=["2023-04-12T00:00:00.000Z","2023-04-12T23:59:59.000Z"],
    logger_properties = {"logger_handler": "StreamHandler"}
)
```

Aggregated Dataset's URL

Due to the malfunction of the HTTPServer web-service in Thredds for aggregated datasets, it is necessary to substitute the HTTPServer link with the URL of the dataset containing non-aggregated data (aggregated_dataset_url). This modification aims to address the issue of the inability to download the aggregated data.

```
[ ]: TDS2STACIntegrator(
    TDS_catalog="https://thredds.imk-ifu.kit.edu/thredds/catalog/catalogues/climate_
↪catalog_ext.html?dataset=eobs_daily_0.1_aggregated_v27",
    stac_dir="stac/",
    limited_number=1,
    aggregated_dataset_url="https://thredds.imk-ifu.kit.edu/thredds/catalog/regclim/
↪raster/regional/eobs/v27.0e/catalog.html",
    logger_properties = {"logger_handler": "StreamHandler"}
)
```

Spatial information

Certain datasets exhibit a deficiency in spatial coordination, which can be rectified by incorporating Latitude and Longitude information into the STAC-Metadata. (spatial_information)

```
[ ]: from tds2stac import TDS2STACIntegrator
#Raster dataset, POLYGON geometry
TDS2STACIntegrator(
    TDS_catalog="https://thredds.imk-ifu.kit.edu/thredds/catalog/sawam_data/karun/frcst/
↪regav/hydrol/catalog.html",
    stac_dir="stac/",
    limited_number=1,
    spatial_information=[0,0,0,0],
```

(continues on next page)

(continued from previous page)

```
    logger_properties = {"logger_handler": "StreamHandler"}
)
# Station-based dataset, POINT geometry
TDS2STACIntegrator(
    TDS_catalog="https://thredds.imk-ifu.kit.edu/thredds/catalog/sawam_data/karun/frcst/
↪point/hydrol/catalog.html",
    stac_dir="stac/",
    limited_number=1,
    spatial_information=[0,0],
    logger_properties = {"logger_handler": "StreamHandler"}
)
```

Temporal information with dataset name

Certain datasets may lack temporal information, and to address this limitation, this approach aims to infer temporal information by using the dataset's name. To obtain further information regarding the naming conventions for each variable, it is advisable to refer to the documentation on strftime() and strptime() Format Codes.(temporal_format_by_dataname)

https://docs.python.org/3/library/datetime.html#strftime-and-strptime-format-codes

[ ]:
```
from tds2stac import TDS2STACIntegrator

TDS2STACIntegrator(
    TDS_catalog="https://thredds.imk-ifu.kit.edu/thredds/catalog/regclim/atmos_obs/lidar/
↪Data/20220714/catalog.html?dataset=lidar/Data/20220714/e2271407.590015",
    stac_dir="stac/",
    limited_number=2,
    spatial_information=[0,0,0,0],
    temporal_format_by_dataname="e%y%m%d%H.%M%S%f",
    logger_properties = {"logger_handler": "StreamHandler"}
)
```

Switch the geometry to LineString

Certain datasets contain LineString geometry, which, similar to Polygon geometry, have attributes such as minimum latitude, maximum latitude, minimum longitude, and maximum longitude. However, these attributes may not be readily identifiable in the dataset's webservice, making it difficult to distinguish them from Polygon geometry. Therefore, this argument is intended to modify the representation of Polygons in such cases.(item_geometry_linestring)

[ ]:
```
TDS2STACIntegrator(
    TDS_catalog="https://thredds.atmohub.kit.edu/thredds/catalog/caribic/IAGOS-CARIBIC_
↪MS_files_collection_20231115/CARIBIC-1/catalog.html",
    stac_dir="stac/",
    limited_number=1,
    item_geometry_linestring=True
)
```

Extension properties

The extension_properties argument can be employed to specify extended properties for STAC-Collections, referred to as `collection_extensions`, and for STAC-Items, referred to as `item_extensions`. The structural framework employed for both entities is same. In the present iteration, the utilization of two predefined keywords, namely `common_metadata` and `item_datacube_extension`, enables the incorporation of common_metadata and datacube extension components into our items. To create custom extensions, it is possible to specify new ones as detailed in the provided documentation. Presented below is an illustrative example that warrants examination.

In the shown example, two distinct types of extensions were utilized. The custom extension, which was both user-defined and in existence. The `common_metadata` and `item_datacube_extension` have been preconfigured, while the user is responsible for specifying two more tuples. The `scientific_extension` is a STAC extension that builds upon the extension function of pySTAC. Similarly, the contacts_extension is a custom function created by us, which is derived from the definition of this extension found in its repository. The primary key of the JSON file corresponds to the first element in these tuples, which is determined by the definition of the key in the `tag_file.json`. The second parameter refers to the identifier of the function or class, while the third parameter pertains to the location of the script.

```
[ ]: TDS2STACIntegrator(
        "https://thredds.imk-ifu.kit.edu/thredds/catalog/regclim/raster/global/era5/sfc/
    ↪single/daily/catalog.html",
        stac_dir="/Users/hadizadeh-m/stac/",
        limited_number=2,
        extension_properties={
            "item_extensions": [
                "common_metadata",
                "item_datacube_extension",
                (
                    "scientific_extension",
                    "item",
                    "./custom_based_on_defined_extension_pystac.py",
                ),
                (
                    "contact_extension",
                    "item",
                    "./custom_based_on_none_defined_extension_pystac.py",
                ),
            ],
        },
```

Webservice properties

This dictionary serves the purpose of identifying the location of the self-created `tag_file.json`. If the tag file is not explicitly defined in this context, the default tag file stored in the main directory of TDS2STAC is utilized.

To obtain instructions on creating a `tag_file.json`, please refer to this documentation. In this particular instance, our objective is to designate our tag configuration file as `tag_example.json`. Subsequently, we aim to ascertain the value of `web_service_config_file` key by referring to the subsequent example.

```
[ ]: from tds2stac import TDS2STACIntegrator

    TDS2STACIntegrator(
        "https://thredds.imk-ifu.kit.edu/thredds/catalog/regclim/raster/global/era5/sfc/
```

(continues on next page)

(continued from previous page)

```
↪single/daily/catalog.html",
    stac_dir="stac/",
    limited_number=1,
    depth_number=1,
    webservice_properties={
        "web_service_config_file": "./tag_example.json",
    },
    logger_properties={"logger_handler":"StreamHandler"},
)
```

Asset properties

By utilizing asset_properties argument, we are able to effectively administer our assets within each STAC-Item and STAC-Collection. To obtain further information regarding asset properties, go to the aforementioned webpage.

In the subsequent illustration, we employ all conceivable premises of this contention.

[ ]:
```
from tds2stac import TDS2STACIntegrator

TDS2STACIntegrator(
    "https://thredds.imk-ifu.kit.edu/thredds/catalog/regclim/raster/global/era5/sfc/
↪single/daily/catalog.html",
    stac_dir="stac/",
    limited_number=1,
    asset_properties={
        "item_thumbnail": True,
        "item_getminmax_thumbnail": True,
        "explore_data": True,
        "verify_explore_data": True,
        "jupyter_notebook": True,
        "collection_thumbnail": "link",
        "collection_overview": "wms",
        "collection_link": "https://images.fineartamerica.com/images-medium-large-5/
↪global-precipitation-william-putmannasa-goddard-space-flight-center.jpg",
        "assets_list_allowed": ["wms", "wcs", "wfs"],
        "assets_list_avoided": ["http"],
    },
   logger_properties={"logger_handler":"StreamHandler"},
)
```

CHAPTER

FIVE

API REFERENCE

This API reference is auto-generated from the Python docstrings. The table of contents on the left is organized by module.

5.1 API Reference

5.1.1 tds2stac package

TDS2STAC

TDS2STAC

class tds2stac.**CollectionHarvester**(*url: str*, *recognizer: str | None*, *subdirs: list | None = []*, *collection_tuples: list[tuple] | None = None*, *logger_properties: dict = {}*, *requests_properties: dict = {}*)

Bases: `object`

This class harvests data pertaining to Collections from TDS catalogs. Depending on the sort of dataset scenario, it returns one of the five variables below. `collection_id` , `collection_title` , `collection_description` , `collection_url` , and `collection_subdirs`.

Parameters

- **url** (*str*) – TDS catalog URL address
- **recognizer** (*str*) – `status` scenario number of *Recognizer*
- **subdirs** (*list*) – subdirs is a list of url, id, title, and subdirs of a nested dataset
- **collection_tuples** (*list*) – a tuple of STAC collection's auto-generated ID, user-ID, user-Title and user-Description defined by user.
- **logger_properties** (*dict*) – dictionary of logger properties
- **requests_properties** (*dict*) – dictionary of requests properties

collection_id_desc_maker(*url: str*, *collection_tuples: list[tuple] | None = None*, *recognizer_output: str | None = None*)

A function for getting collection id and description from the TDS catalog urls and pre-defined `collection_tuples` for scenarios number 4, 5, 6, 7 ,and 9

Parameters

- **url** (*str*) – TDS catalog URL address

- **collection_tuples** (*list*) – a tuple of STAC collection's auto-generated ID, user-ID, user-Title and user-Description defined by user.
- **recognizer_output** (*str*) – `status` scenario output of Recognizer class

collection_tuples: list[tuple] | None

a tuple of STAC collection's auto-generated ID, user-ID, user-Title and user-Description defined by user.

logger_properties: dict

dictionary of logger properties, more information in *Logger*

recognizer: str | None

`status` scenario output of Recognizer class

requests_properties: dict

To obtain additional information on this topic, refer to the `requests_properties`. The default value is an empty dictionary.

subdirs: list | None

subdirs is a list of url, id, title, and subdirs of a nested dataset

url: str

TDS catalog URL address. Initial point of harvesting e.g. https://thredds.atmohub.kit.edu/thredds/catalog/caribic/IAGOS-CARIBIC_MS_files_collection_20231017/catalog.html (*)

class tds2stac.**Datacube**

Bases: `object`

This class is responsible for adding the datacube extension to the STAC Item. :param item: The STAC Item to be extended. :type item: pystac.Item :param harvesting_vars: The dictionary of the variables and dimensions of the dataset. :type harvesting_vars: dict :param logger_properties: The dictionary of the logger properties. :type logger_properties: dict

item_extension(*item*, *harvesting_vars*, *logger_properties: dict = {}*)

class tds2stac.**ExistenceValidator**(*stac_dir: str = '/home/docs/checkouts/readthedocs.org/user_builds/tds2stac/checkouts/stable/docs'*, *logger_properties: dict | None = {}*)

Bases: `object`

A class for verifying the main STAC catalog's existence. This class is implemented in *STACCreator*.

Parameters

- **stac_dir** (*st, Optional*) – Directory of the main STAC catalog (*)
- **logger_properties** (*dict, optional*) – A dictionary of properties for logger. default is None.

logger_properties: dict | None

A dictionary of properties for logger. default is None. You can look at keys in *Logger* class.

stac_dir: str

Directory of the main STAC catalog. It can be a relative or absolute path.

class tds2stac.**ItemHarvester**(*url: str*, *elem: Element*, *harvesting_vars: dict*, *web_service_dict: dict | None*, *datetime_after: datetime | None = None*, *datetime_before: datetime | None = None*, *spatial_information: list | None = None*, *temporal_format_by_dataname: str | None = None*, *extension_properties: dict | None = None*, *linestring: bool = False*, *requests_properties: dict = {}*, *logger_properties: dict = {}*)

Bases: object

This class harvests information about an Item from TDS data catalogs. It ultimately returns a dictionary of harvesting variables, based on the type of dataset scenario and activated extensions.

Parameters

- **url** (*str*) – TDS catalog URL address
- **elem** (*str*) – xml element of the data in dataset
- **harvesting_vars** (*dict*) – dictionary of harvesting variables that is going to be filled
- **web_service_dict** (*dict*) – web service that the user wants to harvest from
- **datetime_after** (*str*) – datetime that the user wants to harvest data after that
- **datetime_before** (*str*) – datetime that the user wants to harvest data before that
- **spatial_information** (*list*) – Spatial information of 2D datasets e.g. [minx, maxx, miny, maxy] or 1D dataset e.g. [x,y]
- **temporal_format_by_dataname** (*str*) – datetime format for datasets that have datetime in their name e.g `e%y%m%d%H.%M%S%f` (optional),
- **extension_properties** (*dict*) – dictionary of extension properties (optional)
- **linestring** (*bool*) – using this attribute, user activate making LineString instead of Polygon (True and False) (optional)
- **logger_properties** (*dict*) – dictionary of logger properties

datetime_after: str | None

datetime that the user wants to harvest data after that

datetime_before: str | None

datetime that the user wants to harvest data before that

elem: Element

xml element of the data in dataset. It's an element of the xml file that is going to be harvested

extension_properties: dict | None

dictionary of extension properties (optional)

harvesting_vars: dict

dictionary of harvesting variables that is going to be filled

linestring: bool

using this attribute, user activate making LineString instead of Polygon (True and False) (optional)

logger_properties: dict

dictionary of logger properties, more information in *Logger*

requests_properties: dict

To obtain additional information on this topic, refer to the `requests_properties`. The default value is an empty dictionary.

spatial_information: list | None

Spatial information of 2D datasets e.g. [minx, maxx, miny, maxy] or 1D dataset e.g. [x,y] (optional)

temporal_format_by_dataname: str | None

datetime format for datasets that have datetime in their name e.g `e%y%m%d%H.%M%S%f` (optional)

usl: str

TDS catalog URL address. Initial point of harvesting e.g. https://thredds.atmohub.kit.edu/thredds/catalog/caribic/IAGOS-CARIBIC_MS_files_collection_20231017/catalog.html (*)

web_service_dict: dict | None

web service that the user wants to harvest from

class tds2stac.**JSONFileWebServiceListScraper**(*json_file: str*, *logger_properties: dict = {}*)

Bases: object

A class to get all `tds2stac_webservice_analyser` in tag_config.json file when `tds2stac_mode_analyser` is `get` or `check`.

load_and_process_json()

class tds2stac.**Logger**(*logger_properties: dict[str, Any] | None = {}*)

Bases: object

A class-based logger for TDS2STAC. It supports all the handlers from the standard python logging library.

Parameters

logger_properties (*dict, optional*) – Logger properties. Defaults to `dict()`. It's optional and has the following keys:

logger_msg (str, optional)

logger_handler (str, optional)

logger_name (str, optional)

logger_id (str, optional)

logger_level (str, optional)

logger_formatter (str, optional)

logger_handler_host (str, optional)

logger_handler_port (str, optional)

logger_handler_url (str, optional)

logger_handler_method (str, optional)

logger_handler_secure (bool, optional)

logger_handler_credentials (tuple, optional)

logger_handler_context (tuple, optional)

logger_handler_filename (str, optional)

logger_handler_mode (str, optional)

logger_handler_encoding (str, optional)

logger_handler_delay (bool, optional)

logger_handler_errors (str, optional)

logger_handler_mailhost (str, optional)

logger_handler_fromaddr (str, optional)

logger_handler_toaddrs (str, optional)

logger_handler_subject (str, optional)

logger_handler_timeout (str, optional)

`Null_Handler()`

This is a function to return a NullHandler

`logger_properties: dict[str, Any] | None`

A dictionary that contains all the logger properties.

It is optional and it is set to `None` by default. The following keys are supported:

logger_msg (str, optional):
Logger message. Defaults to `None`. But it is required when you want to log a message.

logger_handler (str, optional):
Logger handler. Defaults to `NullHandler`. Check the following website for more information:

https://docs.python.org/3/library/logging.handlers.html#module-logging.handlers

logger_name (str, optional):
Logger name. Defaults to `INSUPDEL4STAC`. It's required when you choose `HTTPHandler` as logger_handler.

logger_id (str, optional):
Logger id. Defaults to 1. It's required when you choose `HTTPHandler` as logger_handler.

logger_level (str, optional):
Logger level. Defaults to `DEBUG`. It's optional. For more information check the following website:

https://docs.python.org/3/library/logging.html#levels

logger_formatter (str, optional):
Logger format. Defaults to `%(levelname)-8s %(asctime)s t %(filename)s @function %(funcName)s line %(lineno)s - %(message)s`. For more information check the following website:

https://docs.python.org/3/library/logging.html#formatter-objects

logger_handler_host (str, optional):
Logger host. Sets the value to 'None' by default. It is required when `HTTPHandler` or `SocketHandler` are selected as the `logger_handler`. The `logger_handler` will be set to 'NullHandler' if `HTTPHandler` or `SocketHandler` is selected as the `logger_handler` value and neither `logger_handler_host` nor `logger_handler_port` nor are specified.

logger_handler_port (str, optional):
Logger port. Sets the value to 'None' by default. It is required when `HTTPHandler` or `SocketHandler` are selected as the `logger_handler`. The `logger_handler` will be set to 'NullHandler' if `HTTPHandler` or `SocketHandler` is selected as the `logger_handler` value and neither `logger_handler_host` nor `logger_handler_port` are specified.

logger_handler_url (str, optional):
Logger url. Sets the value to 'None' by default. It is required when `HTTPHandler` is selected as the `logger_handler`. The `logger_handler` will be set to 'NullHandler' if `HTTPHandler` is selected as the `logger_handler` value and neither `logger_handler_url` is specified.

logger_handler_method (str, optional):
HTTP methods. It supports sending logging messages to a web server, using either GET or POST semantics. Sets the value to 'None' by default. It is required when `HTTPHandler` is selected as the `logger_handler`. The `logger_handler` will be set to 'NullHandler' if

HTTPHandler is selected as the logger_handler value and logger_handler_method is not specified.

logger_handler_secure (bool, optional):
HTTP secure. Sets the value to 'False' by default. It is utilized when HTTPHandler or SMTPHandler are selected as the logger_handler. But it is optional in both logger handlers.

logger_handler_credentials (tuple, optional):
HTTP credentials. Sets the value to 'None' by default. It is utilized when HTTPHandler or SMTPHandler are selected as the logger_handler. But it is optional in both logger handlers.

logger_handler_context (tuple, optional):
HTTP context. Sets the value to 'None' by default. It is utilized when HTTPHandler is selected as the logger_handler. But it is optional in both logger handlers.

logger_handler_filename (str, optional):
File name. Sets the value to 'None' by default. It is required when FileHandler or WatchedFileHandler are selected as the logger_handler. The logger_handler will be set to 'NullHandler' if FileHandler or WatchedFileHandler is selected as the logger_handler value and logger_handler_filename is not specified.

logger_handler_mode (str, optional):
File mode. Sets the value to 'None' by default. It is required when FileHandler or WatchedFileHandler are selected as the logger_handler. The logger_handler will be set to 'NullHandler' if FileHandler or WatchedFileHandler is selected as the logger_handler value and logger_handler_mode is not specified.

logger_handler_encoding (str, optional):
File encoding. Sets the value to 'None' by default. It is utilized when FileHandler or WatchedFileHandler are selected as the logger_handler. But it is optional in both logger handlers.

logger_handler_delay (bool, optional):
File delay. Sets the value to 'False' by default. It is utilized when FileHandler or WatchedFileHandler are selected as the logger_handler. But it is optional in both logger handlers.

logger_handler_errors (str, optional):
File errors. Sets the value to 'None' by default. It is utilized when FileHandler or WatchedFileHandler are selected as the logger_handler. But it is optional in both logger handlers.

logger_handler_mailhost (str, optional):
Mail host. Sets the value to 'None' by default. It is required when SMTPHandler is selected as the logger_handler. The logger_handler will be set to 'NullHandler' if SMTPHandler is selected as the logger_handler value and logger_handler_mailhost is not specified.

logger_handler_fromaddr (str, optional):
Mail from address. Sets the value to 'None' by default. It is required when SMTPHandler is selected as the logger_handler. The logger_handler will be set to 'NullHandler' if SMTPHandler is selected as the logger_handler value and logger_handler_fromaddr is not specified.

logger_handler_toaddrs (str, optional):
Mail to address. Sets the value to 'None' by default. It is required when SMTPHandler is selected as the logger_handler. The logger_handler will be set to 'NullHandler' if

SMTPHandler is selected as the logger_handler value and logger_handler_toaddrs is not specified.

logger_handler_subject (str, optional):
Mail subject. Sets the value to 'None' by default. It is utilized when SMTPHandler is selected as the logger_handler. But it is optional in both logger handlers.

logger_handler_timeout (str, optional):
Mail timeout. Sets the value to 'None' by default. It is utilized when SMTPHandler is selected as the logger_handler. But it is optional in both logger handlers.

class tds2stac.**NestedCollectionInspector**(*main_catalog_url: str*, *nested_number: int | None = None*, *logger_properties: dict = {}*, *requests_properties: dict = {}*)

Bases: object

This class will generate Collection IDs, Titles and their corresponding URLs for a presumed nested number originating from the Recognizer class in TDS. Only works for nested scenarios number 1,2,3,8 and 9 in Recognizer class. The output will be a list of the tuples: (Root collection URL, Collection ID, Collection Title, corresponding subset URLs)

Parameters

- **main_catalog_url** (*str*) – The URL of the TDS catalog
- **nested_number** (*int, optional*) – Number of depth for nested datasets
- **logger_properties** (*dict, optional*) – A dictionary for logger properties
- **requests_properties** (*dict, optional*) – A dictionary for requests properties

aslist()

A function for returning the list of tuples

end_point_url_extractor_dict(*d: dict*)

A function for extracting the end point URLs of a nested dictionary.

Parameters
d (*dict*) – A nested dictionary

end_point_url_extractor_list(*list_: list*)

A function for extracting the end point URLs of a nested list.

Parameters
list (*list*) – A nested list

final_collections_details_returner(*url: str*)

A function for returning the URLs of input URL in First and Third cases in TDS

Parameters
url (*str*) – The URL of the TDS catalog

logger_properties: dict

A dictionary for logger properties. For more information see *Logger*

main_catalog_url: str

The URL of the TDS catalog

n_level(*d: dict*, *layer: int*)

For decoding the generator object of to_level function. https://stackoverflow.com/a/68228562

Parameters

- **d** (*dict*) – A nested dictionary

- **layer** (*int*) – The depth of the dictionary

nested_dict_returner(*url: str*, *dict: dict*)

A function for getting the nested dictionary of a given URL.

Parameters

- **url** (*str*) – The URL of the TDS catalog
- **dict** (*dict*) – A nested dictionary

nested_number: int | None

Number of depth for nested datasets

requests_properties: dict

To obtain additional information on this topic, refer to the `requests_properties`. The default value is an empty dictionary.

to_level(*d: dict*, *layer: int*)

A function for getting the a dictionary in a given depth. https://stackoverflow.com/a/68228562

Parameters

- **d** (*dict*) – A nested dictionary
- **layer** (*int*) – The depth of the dictionary

class tds2stac.**Recognizer**(*main_catalog_url: str*, *nested_check: bool = False*, *logger_properties: dict = {}*, *requests_properties: dict = {}*)

Bases: `object`

A class for recognizing nine different and possible scenarios in management of TDS datasets. We will explain each scenario in the following.

First scenario: Just `catalogRef` tags are located directly under the dataset element tag.
tag `https://thredds.imk-ifu.kit.edu/thredds/catalog/regclim/raster/global/era5/sfc/single/catalog.xml` (nested)

Second senarion: `CatalogRefs` are not under a dataset element tag and directly come below the `catalog`.
`https://thredds.imk-ifu.kit.edu/thredds/catalog/catalogues/sensor_catalog_ext.xml` (nested)

Third scenario: One single `dataset` tag is located next to `CatalogRef` tags. All are under a `dataset` tag.
`https://thredds.imk-ifu.kit.edu/thredds/catalog/regclim/raster/global/chirps/catalog.xml` (nested)

Fourth scenario: An empty datasets.
`https://thredds.imk-ifu.kit.edu/thredds/catalog/catalogues/bio_geo_chem_catalog_ext.xml` or `https://thredds.atmohub.kit.edu/thredds/catalog/snowfogs/catalog.xml`

Fifth scenario: There is no `CatalogRef` tag and all are `dataset` tag. All of them are under a `dataset` tag.
`https://thredds.imk-ifu.kit.edu/thredds/catalog/climate/raster/global/chelsa/v1.2/catalog.html`

Sixth scenario: A single dataset
`https://thredds.imk-ifu.kit.edu/thredds/catalog/regclim/raster/global/era5/sfc/single/catalog.xml?dataset=regclim/raster/global/era5/sfc/single/era5_sfc_20210101.nc`

Seventh scenario: An aggregated dataset
`https://thredds.imk-ifu.kit.edu/thredds/catalog/catalogues/swabian_moses_2021.xml?dataset=swabian_moses_aggregation`

Eighth scenario: A combination of `caralogRef` and `dataset` tags that is not under a `dataset` tag.It's similar to second scenario but with datasets
`https://thredds.imk-ifu.kit.edu/thredds/catalog/catalogues/transfer.xml`

Ninth scenario: When we have a bunch of single `dataset` tags next to catalogref. It's similar to third scenario but with more datasets.
`https://thredds.imk-ifu.kit.edu/thredds/catalog/regclim/raster/global/hydrogfd/v3.0/catalog.xml` (nested)

Parameters

- **main_catalog_url** – TDS Catalog url to start harvesting
- **nested_check** – An option for checking nested datasets in TDS (True or False)
- **auth** – Authentication for TDS catalog e.g.('user', 'password')
- **logger_properties** – A dictionary for logger properties.
- **requests_properties** – A dictionary for requests properties.

logger_properties: dict

A dictionary for logger properties. For more information see *Logger*

main_catalog_url: str

TDS Catalog url to start harvesting (*)

nested_check: bool

An option for checking nested datasets in TDS (True or False) (optional)

nested_checker(*url: str*)

A function for returning the depth of nested datasets in TDS for scenarios 1, 3, ,and 9

nested_checker_exceptions(*url: str*)

A function for returning the depth of nested datasets in TDS for scenarios 2 and 8

recognition_function(*url: str*, *xml_content*)

A function for recognizing number of scenarios in TDS

requests_properties: dict

To obtain additional information on this topic, refer to the `requests_properties`. The default value is an empty dictionary.

class tds2stac.**STACCreator**

Bases: `object`

A class for creating STAC catalog, -Collections and its -Items from TDS datasets catalogs.

STACCatalog(*url: str*, *stac_id: str*, *stac_title: str | None*, *stac_desc: str | None*, *stac_dir: str*, *stac_existence: bool = False*, *logger_properties: dict = {}*, *requests_properties: dict = {}*)

A function for creating STAC catalog from TDS dataset catalog.

Parameters

- **url** – The URL of the TDS catalog.
- **stac_id** – The ID of the STAC catalog.

- **stac_title** – The title of the STAC catalog.
- **stac_desc** – The description of the STAC catalog.
- **stac_dir** – The directory of saving the STAC catalog.
- **stac_existence** – If it is True, it means that the STAC catalog already exists in the directory and for the harvesting, there is no need to create a new STAC-Catalog and import new collections In the existed STAC-Catalog. False by default.
- **logger_properties** – The properties of the logger. For more information please check the *Logger* class.
- **requests_properties** – The properties of the requests. For more information please check the `requests_properties` class.

STACCollection(*catalog: Catalog*, *collection_id: str*, *collection_title: str*, *collection_description: str*, *stac_existence_collection: bool = False*, *logger_properties: dict = {}*, *extra_metadata: dict = {}*)

This is a function for creating STAC collection from harvested information from TDS dataset catalog. This function returns a dictionary with two keys:

1. collection: The STAC collection

2. existed_items_id_list: The list of the items that already exist in the STAC collection and it is going to be used for the harvesting process.

Parameters

- **catalog** – The STAC catalog.
- **collection_id** – The ID of the STAC collection.
- **collection_title** – The title of the STAC collection.
- **collection_description** – The description of the STAC collection.
- **collection_scientific** – The scientific extension of the STAC collection.
- **stac_existence_collection** – If it is True, it means that the STAC collection already exists in the catalog and for the harvesting, there is no need to create a new STAC-Collection and import new items In the existed STAC-Collection. False by default.
- **logger_properties** – The properties of the logger. For more information please check the *Logger* class.

STACItem(*url: str*, *catalog: Catalog*, *harvesting_vars: dict*, *Recognizer_output: str | None*, *collection_id: str*, *aggregated_dataset_url: str | None = None*, *extension_properties: dict | None = None*, *asset_properties: dict | None = {}*, *logger_properties: dict = {}*, *extra_metadata: dict = {}*, *stac_existence_collection: bool = False*, *collection_bbox_existed: list | None = None*, *collection_interval_time_final_existed: list | None = None*)

This is a function for creating STAC item from harvested data in TDS dataset catalog.

Parameters

- **url** – The URL of the TDS catalog.
- **catalog** – The STAC catalog.
- **harvesting_vars** – The harvested data from TDS catalog.
- **Recognizer_output** – The output of the Recognizer class.
- **collection_id** – The ID of the STAC collection.

- **aggregated_dataset_url** – The URL of the aggregated dataset that whole of data is located there.
- **extension_properties** – The properties of the extensions.
- **asset_properties** – The properties of the assets.
- **logger_properties** – The properties of the logger. For more information please check the *Logger* class.

SaveCatalog(*catalog*, *catalog_dir*, *logger_properties: dict = {}*)

class tds2stac.**Spatial**

Bases: object

harvester(*main_dict*, *linestring=None*)

regulator(*main_dict*, *spatial_information*)

A function for regulating the spatial information of a catalog

class tds2stac.**TDS2STACIntegrator**(*TDS_catalog: str*, *stac_dir: str = '/home/docs/checkouts/readthedocs.org/user_builds/tds2stac/checkouts/stable/docs'*, *stac_id: str = 'TDS2STAC'*, *stac_title: str | None = 'TDS2STAC'*, *stac_description: str | None = None*, *stac_existence: bool = False*, *stac_existence_collection: bool = False*, *collection_tuples: list[tuple] | None = None*, *datetime_filter: list | None = None*, *aggregated_dataset_url: str | None = None*, *depth_number: int | None = None*, *limited_number: int | None = None*, *spatial_information: list | None = None*, *temporal_format_by_dataname: str | None = None*, *item_geometry_linestring: bool = False*, *webservice_properties: dict | None = {}*, *asset_properties: dict | None = {}*, *extension_properties: dict | None = {}*, *logger_properties: dict = {}*, *requests_properties: dict = {}*, *extra_metadata: dict = {}*)

Bases: object

This class is the central component of the TDS2STAC. It harvests the TDS catalog and then generates the STAC-Catalog, -Collections, and -Items through the TDS catalogs, based on the user's input. This class mainly defines all configurations related to harvesting and STAC creation. In the first step, it recognizes the scenario of the TDS catalog using *Recognizer*. If it is recognized as a nested collection, *NestedCollectionInspector* is responsible for determining the nested collection's `ID`, `Title`, and `url` of subdirectories. Other procedures follow in succession. For example, *CollectionHarvester* harvests the collection's information and *STACCreator* creates the STAC-Catalog and -Collection. Then, *ItemHarvester* harvests the item's information and *STACCreator* creates the STAC-Item and connect them to the related STAC-Collections. At the end each STAC-Collection will be connected to the main STAC-Catalog.

Parameters

- **TDS_catalog** (*str*) – The URL address of the TDS catalog that will be harvested.
- **stac_dir** (*str, Optional*) – Directory of saving created STAC catalogs.
- **stac_id** (*str, Optional*) – STAC catalog ID. default value is 'TDS2STAC'.
- **stac_title** (*str, optional*) – STAC catalog Title. default value is 'TDS2STAC'.
- **stac_description** (*str, optional*) – STAC catalog description.
- **stac_existence** (*bool, optional*) – Verifying the presence of the STAC catalog in order to update an existing catalog; if not, a new catalog will be generated.

- **stac_existence_collection** (*bool, optional*) – Verifying the presence of the STAC Collection in order to update an existing catalog; if not, a new collection will be generated.
- **collection_tuples** (*list, optional*) – The elements of this tuple comprise the auto-TDS2STAC-generated ID , the user-defined ID, title, and description of the STAC-Collection respectively. (auto-ID, user-ID, user-title, user-description).
- **datetime_filter** (*list, optional*) – Datetime-based filtering of harvesting. It works based on the `modified` tag in each dataset at TDS.
- **aggregated_dataset_url** (*str, optional*) – Dataset's URL of each data entry in the Aggregated datasets of TDS.
- **depth_number** (*int, optional*) – depth number of nested datasets if it is a nested collection. default value is 0.
- **limited_number** (*int, optional*) – The objective is to reduce the quantity of harvested items in each collection. It is beneficial for developing and testing purposes.
- **spatial_information** (*list, optional*) – Spatial information of 2D datasets e.g. [minx, maxx, miny, maxy] or 1D dataset e.g. [x,y]. default value is None.
- **temporal_format_by_dataname** (*str, optional*) – A preferred datetime format for datasets that include the time period in their names. e.g "e%y%m%d%H.%M%S%f"
- **item_geometry_linestring** (*bool, optional*) – Set True to make a LineString geometry for STAC-Items from wms service. Otherwise it makes Polygon geometry for the given Item. default value is False.
- **extension_properties** (*dict, optional*) – A dictionary of properties for extensions. default is None. For more information about the keys, please refer to the *extension_properties*.
- **webservice_properties** (*dict, optional*) – A dictionary of properties for web_service. default is None (optional) For more information about the keys, please refer to the *webservice_properties*.
- **asset_properties** (*dict, optional*) – A dictionary of properties for assets. default is None (optional) For more information about the keys, please refer to the *asset_properties*.
- **logger_properties** (*dict, optional*) – A dictionary of properties for logger. default is `None`.
- **requests_properties** – A dictionary that modify the requests to URLs. To obtain additional information on this topic, refer to the `requests_properties`. The default value is an empty dictionary.

TDS_catalog: str

TDS catalog URL address. Initial point of harvesting e.g. https://thredds.atmohub.kit.edu/thredds/catalog/caribic/IAGOS-CARIBIC_MS_files_collection_20231017/catalog.html

aggregated_dataset_url: str | None

Dataset's URL of each data entry in the Aggregated datasets of TDS.. default value `None`. The `HTTPServer` is not functional in the aggregated dataset. Therefore, in order to utilize this service as an asset in our STAC-Item, we should employ the `aggregated_dataset_url`, which links the individual datasets to the `HTTPServer` asset of the relevant Item.

asset_properties: dict | None

A dictionary of properties for assets. default is `None`. When it's None, keys look like the following example:

collection_thumbnail (str, optional):
A `thumbnail` asset for STAC-collection sourced from the Web Map Service (WMS) of the TDS. It can be chosen from `wms`, `link`, or `None`. The default value is set to None.

collection_overview (str, optional):
A `overview` asset for STAC-collection sourced from the Web Map Service (WMS) of the TDS. It can be chosen from `wms`, `link`, or `None`. The default value is set to None.

collection_thumbnail_link (str, optional):
This property is reliant upon the values of `collection_thumbnail` and `collection_overview`. When the value of either of these attributes is set to `link`, it allows for the inclusion of a hyperlink to an image for `collection_thumbnail` or `collection_overview`.

collection_overview_link (str, optional):
This property is reliant upon the values of `collection_thumbnail` and `collection_overview`. When the value of either of these attributes is set to `link`, it allows for the inclusion of a hyperlink to an image for `collection_thumbnail` or `collection_overview`.

collection_custom_assets (list, optional):
This is a list of asset dictionaris that includes the `key`, `href`, and `title`, `role` (as a list), and `media_type` of the asset. The default value is set to None. For more information, refer to the *How to make a custom asset for STAC-Collection and STAC-Item:*.

item_thumbnail (bool, optional):
A `thumbnail` asset for STAC-Items sourced from the Web Map Service (WMS) of the TDS. The default value is set to False.

item_overview (bool, optional):
A `overview` asset for STAC-Items sourced from the Web Map Service (WMS) of the TDS. The default value is set to False.

item_getminmax_thumbnail (bool, optional):
The TDS offers a function that allows users to obtain the minimum and maximum values of the colorbar associated with an image through the use of `metadata`. The aforementioned attribute is contingent upon both the `item_thumbnail` and `item_overview`. The default value is set to False.

assets_list_allowed (list, optional):
This is a list of permissible web services that will be incorporated as assets in the STAC-Item. The `WebServiceScraper` class provides access to the list of available web services. Default value is None.

assets_list_avoided (list, optional):
This is a list of web services that will be excluded from the STAC-Item asset list. The `WebServiceScraper` class provides access to the list of available webservices. Default value is None.

explore_data (bool, optional):
By enabling the `True` setting, the inclusion of Godiva3 as an exploration asset will be implemented.

verify_explore_data (bool, optional):
This argument verifies the availability of the `GetMetadata` function. The provided function facilitates the retrieval of data necessary for generating maps using the Web Map Service (WMS) protocol. However, an error occurs when attempting to open `Godiva3` when this function doesn't work. In order to mitigate such errors, it would be advisable to establish this argument.

jupyter_notebook (bool, optional):
This argument posits the inclusion of the Jupyter Notebook as an asset.

collection_tuples: list[tuple] | None

STAC collection auto-generated ID, user-ID, user-Title and user-Description defined by user. It is worth mentioning that in order to obtain the list of automatically generated collection IDs, one can employ the *NestedCollectionInspector* for the given TDS Catalog and subsequently utilize this argument. Warning - Identifiers should consist of only lowercase characters, numbers, '_', and '-'. Default value `None`. e.g. (ID, Title, Description)

datetime_filter: list | None

Datetime-based filtering. e.g. `['2010-02-18T00:00:00.000Z','2020-02-22T00:00:00.000Z']` Default value `None`. It should be noted it works based on the `modified` tag in each dataset at TDS.

depth_number: int | None

The depth refers to the number of layered datasets. If the collection is nested, this argument is applicable; otherwise, employing this argument would be futile. default value `None` (optional)

extension_properties: dict | None

A dictionary of properties for extensions. default is `None`.

item_extensions (list[str, tuple], optional):
The argument can consist of either a list of extension names (string) or a list of tuples containing three elements: the extension name, the function name or class name associated with the extension, and the Python script required for execution. For more explanation, refer to the *Adding and Configuring Custom Extensions for STAC-Items and STAC-Collections*.

collection_extensions (Union[list, tuple], optional):
It works as same as `item_extensions` argument. For more explanation, refer to the *Adding and Configuring Custom Extensions for STAC-Items and STAC-Collections*.

extra_metadata: dict

A dictionary of extra metadata that you want to add to the STAC-Collection and STAC-Items. It has two main keys, `extra_metadata` that is boolean and `extra_metadata_file` that is the address of `extra_metadata.json` JSON file. For getting more information about making the `extra_metadata.json` file, please refer to *How to create extra_metadata.json file*. By default, if 'extra_metadata' is set to True, the 'extra_metadata.json' file is utilised for the 'extra_metadata_file' key, which is situated in the'sta2stac' main directory.

item_geometry_linestring: Literal[False]

The default value for the LineString geometry in the STAC Items from the WMS service is set to False and the default geometry type for the STAC-Item is Polygon. However, in instances where the item has a POINT geometry, it can be automatically detected. However, in order to obtain the LineString geometry, it is necessary to set this argument to True.

limited_number: int | None

The objective is to reduce the quantity of harvested items in each collection. It is beneficial for developing and testing purposes.. default value `None` (optional)

logger_properties: dict

A dictionary of properties for logger. default is `None`. You can look at keys in *Logger* class.

requests_properties: dict

A dictionary of properties that adjust the requests to URLs. It contains the following keys:

verify (bool, optional):
It is a boolean that if it is True, it verifies the SSL certificate. By default it is False.

timeout (int, optional):
It is an integer that sets the timeout of the requests. By default it is 10 seconds.

auth (tuple, optional):
It is a tuple that contains the username and password for the authentication. By default it is None.

spatial_information: list | None

Spatial information of 2D datasets e.g. [minx, maxx, miny, maxy] or 1D dataset e.g. [x,y]. Default value `None`(optional)

stac_description: str | None

STAC catalog description

stac_dir: str

Directory of saving created STAC catalogs e.g. /path/to/stac/directory/

stac_existence: Literal[False]

Verifying the existence of STAC catalog. If the catalog exists in the directory, it updates a existed catalog, otherwise it creates new catalog. default value `False`

stac_existence_collection: Literal[False]

Verifying the existence of STAC Collections. If the collection exists in the directory, it updates a existed collection, otherwise it creates new collection. default value `False`

stac_id: str

STAC catalog ID. default value `TDS2STAC`

stac_title: str | None

STAC catalog Title. default value `TDS2STAC`

temporal_format_by_dataname: str | None

A preferred datetime format for datasets that include the time period in their names e.g "e%y%m%d%H.%M%S%f". Default value `None` (optional)

webservice_properties: dict | None

A dictionary of properties for web_service. default is `None`.

It has the following keys.

web_service_config_file(str, opntional):
The primary `tag_config.json` file is situated in the primary directory of the installed TDS2STAC. However, the user has the ability to declare an alternative `tag_config.json` file, which allows for customization of the settings. The user can specify the location of their own JSON file in this section. To obtain further details on the creation of a `tag_config.json` file, refer: *Creating the tag_config.json Configuration File: A Step-by-Step Guide*. The default value is set to `tag_config.json` in the root directory of the installed app.

class tds2stac.**Temporal**

Bases: object

parse_datetime_with_fallback(*datetime_str*, *primary_format*, *fallback_format*, *tzinfo*)

regulator(*main_dict*, *temporal_format_by_dataname*, *data_name*)

safe_strip(*value*)

class tds2stac.**Thumbnails**

Bases: object

This class is used to create thumbnail images for STAC-Collections and STAC-Items.

collection(*collection_thumbnail: str*, *collection_overview: str*, *services: Element*, *dataset: dict*, *harvesting_vars: dict*, *collection_id: str*, *url: str*, *catalog: Catalog*, *collection_thumbnail_link: str*, *collection_overview_link: str*, *logger_properties: dict = {}*)

A function to create thumbnail images for STAC-Collections.

Parameters

- **collection_thumbnail** (*str*) – The type of thumbnail image for STAC-Collections. It can be `wms` or `link`.
- **collection_overview** (*str*) – The type of overview image for STAC-Collections. It can be `wms` or `link`.
- **services** (*list*) – A list of services for STAC-Collections.
- **dataset** (*dict*) – A dictionary of dataset information.
- **harvesting_vars** (*dict*) – A dictionary of harvesting variables.
- **collection_id** (*str*) – The ID of STAC-Collections.
- **url** (*str*) – The URL of STAC-Catalog.
- **catalog** (*pystac.Catalog*) – A STAC-Catalog.
- **collection_thumbnail_link** (*str*) – The link of thumbnail image for STAC-Collections when `collection_thumbnail` or `collection_overview` set as `link`.
- **collection_overview_link** (*str*) – The link of overview image for STAC-Collections when `collection_thumbnail` or `collection_overview` set as `link`.
- **logger_properties** (*dict*) – A dictionary of logger properties. For more information, please see *Logger* class.

item(*service: Element*, *dataset: dict*, *harvesting_vars: dict*, *url: str*, *item: Item*, *item_thumbnail: bool*, *item_overview: bool*, *item_getminmax_thumbnail: bool*, *logger_properties: dict = {}*)

A function to create thumbnail images for STAC-Items.

Parameters

- **service** (*list*) – A list of services for STAC-Items.
- **dataset** (*dict*) – A dictionary of dataset information.
- **harvesting_vars** (*dict*) – A dictionary of harvesting variables.
- **url** (*str*) – The URL of STAC-Catalog.
- **item** (*pystac.Item*) – A STAC-Item.
- **item_thumbnail** (*bool*) – A boolean to create thumbnail image for STAC-Items.
- **item_overview** (*bool*) – A boolean to create overview image for STAC-Items.
- **item_getminmax_thumbnail** (*bool*) – A boolean to create thumbnail image for STAC-Items based on minmax.
- **logger_properties** (*dict*) – A dictionary of logger properties. For more information, please see *Logger* class.

class tds2stac.**Verifier**

Bases: `object`

This class is responsible for verifying the properties of the dictionary arguments.

asset_properties(*asset_properties: dict*)

This function is responsible for refining the values of the asset_properties dictionary.

extension_properties(*extension_properties: dict*)

This function is responsible for refining the values of the extension_properties dictionary.

extra_metadata(*extra_metadata: dict*) → dict

This function is responsible for refining the values of the extra_metadata dictionary.

logger_properties(*logger_properties: dict*) → dict

This function is responsible for refining the values of the logger_properties dictionary.

requests_properties(*requests_properties: dict*) → dict

This function is responsible for refining the values of the requests_properties dictionary.

webservice_properties(*webservice_properties: dict*)

This function is responsible for refining the values of the webservice_properties dictionary.

class tds2stac.**WebServiceContentScraper**(*root: _Element*, *service_url: str*, *json_file: str*, *extensions_list: list*, *harvesting_vars: dict | None = None*, *logger_properties: dict = {}*)

Bases: object

The functionality of the existing class is dependent on the settings specified in the `tag_config.json` file in order to harvest targeted information from a selected web service. For comprehensive instructions on configuring the `tag_config.json` file, refer to the following link: *Creating the tag_config.json Configuration File: A Step-by-Step Guide*.

Args:

root (etree._Element): The root of the XML-based web service json_file (str): The path to the `tag_config.json` file extensions_list (list): The list of extensions to be harvested

from the web service (main keys in the `tag_config.json` file)

harvesting_vars (dict, optional): The dictionary of harvesting variables logger_properties (dict, optional):The dictionary of the logger properties.

extensions_list: list

The list of extensions to be harvested from the web service. Main keys in the `tag_config.json` file. For example `item_datacube_extension` and so on.

harvester(*root*, *service_url*, *json_file*, *ext_name*, *harvesting_vars=None*)

harvesting_vars: dict | None

It's a dictionary that keys are variable names and values are the result of harvesting.

json_file: str

The path to the `tag_config.json` file

logger_properties: dict

The dictionary of the logger properties. You can look at keys in *Logger* class.

root: _Element

Etree root object of the XML-based web service

class tds2stac.**WebServiceListScraper**(*url: str*, *logger_properties: dict = {}*, *requests_properties: dict = {}*)

Bases: object

A class for getting the list of available web services of a TDS catalogs.

Args:
url (str): The catalog URL from TDS to provide its web services logger_properties (dict, optional):The dictionary of the logger properties. requests_properties (dict, optional): A dictionary that modify the requests to URLs.

`aslist()`

`logger_properties: dict`

The dictionary of the logger properties. You can look at keys in *Logger* class.

`requests_properties: dict`

To obtain additional information on this topic, refer to the `requests_properties`. The default value is an empty dictionary.

`url: str`

url is the url of the TDS catalog

Subpackages

tds2stac.analysers package

Submodules

tds2stac.analysers.existence_validator module

`class tds2stac.analysers.existence_validator.ExistenceValidator(stac_dir: str = '/home/docs/checkouts/readthedocs.org/user_builds logger_properties: dict | None = {})`

Bases: `object`

A class for verifying the main STAC catalog's existence. This class is implemented in *STACCreator*.

Parameters

- **stac_dir** (*st, Optional*) – Directory of the main STAC catalog (*)
- **logger_properties** (*dict, optional*) – A dictionary of properties for logger. default is None.

`logger_properties: dict | None`

A dictionary of properties for logger. default is None. You can look at keys in *Logger* class.

`stac_dir: str`

Directory of the main STAC catalog. It can be a relative or absolute path.

tds2stac.analysers.nested_collections module

class tds2stac.analysers.nested_collections.**NestedCollectionInspector**(*main_catalog_url: str*, *nested_number: int | None = None*, *logger_properties: dict = {}*, *requests_properties: dict = {}*)

Bases: object

This class will generate Collection IDs, Titles and their corresponding URLs for a presumed nested number originating from the Recognizer class in TDS. Only works for nested scenarios number 1,2,3,8 and 9 in Recognizer class. The output will be a list of the tuples: (Root collection URL, Collection ID, Collection Title, corresponding subset URLs)

Parameters

- **main_catalog_url** (*str*) – The URL of the TDS catalog
- **nested_number** (*int, optional*) – Number of depth for nested datasets
- **logger_properties** (*dict, optional*) – A dictionary for logger properties
- **requests_properties** (*dict, optional*) – A dictionary for requests properties

all_nested_dict: dict

all_urls: list

aslist()

A function for returning the list of tuples

corresponding_urls_ids: list | None

depth_addresses: list

end_point_url_extractor_dict(*d: dict*)

A function for extracting the end point URLs of a nested dictionary.

Parameters
d (*dict*) – A nested dictionary

end_point_url_extractor_list(*list_: list*)

A function for extracting the end point URLs of a nested list.

Parameters
list (*list*) – A nested list

final_collections_details_returner(*url: str*)

A function for returning the URLs of input URL in First and Third cases in TDS

Parameters
url (*str*) – The URL of the TDS catalog

layer_dict: dict

logger_properties: dict

A dictionary for logger properties. For more information see *Logger*

main_catalog_url: str
The URL of the TDS catalog

n_level(*d: dict*, *layer: int*)
For decoding the generator object of `to_level` function. https://stackoverflow.com/a/68228562

Parameters

- **d** (*dict*) – A nested dictionary
- **layer** (*int*) – The depth of the dictionary

nested_dict_returner(*url: str*, *dict: dict*)
A function for getting the nested dictionary of a given URL.

Parameters

- **url** (*str*) – The URL of the TDS catalog
- **dict** (*dict*) – A nested dictionary

nested_number: int | None
Number of depth for nested datasets

requests_properties: dict
To obtain additional information on this topic, refer to the `requests_properties`. The default value is an empty dictionary.

to_level(*d: dict*, *layer: int*)
A function for getting the a dictionary in a given depth. https://stackoverflow.com/a/68228562

Parameters

- **d** (*dict*) – A nested dictionary
- **layer** (*int*) – The depth of the dictionary

tds2stac.analysers.properties_verifier module

class tds2stac.analysers.properties_verifier.**Verifier**
Bases: object

This class is responsible for verifying the properties of the dictionary arguments.

asset_properties(*asset_properties: dict*)
This function is responsible for refining the values of the asset_properties dictionary.

extension_properties(*extension_properties: dict*)
This function is responsible for refining the values of the extension_properties dictionary.

extra_metadata(*extra_metadata: dict*) → dict
This function is responsible for refining the values of the extra_metadata dictionary.

logger_properties(*logger_properties: dict*) → dict
This function is responsible for refining the values of the logger_properties dictionary.

requests_properties(*requests_properties: dict*) → dict
This function is responsible for refining the values of the requests_properties dictionary.

webservice_properties(*webservice_properties: dict*)
This function is responsible for refining the values of the webservice_properties dictionary.

tds2stac.analysers.recognizer module

class tds2stac.analysers.recognizer.**Recognizer**(*main_catalog_url: str*, *nested_check: bool = False*, *logger_properties: dict = {}*, *requests_properties: dict = {}*)

Bases: `object`

A class for recognizing nine different and possible scenarios in management of TDS datasets. We will explain each scenario in the following.

First scenario: Just `catalogRef` tags are located directly under the dataset element tag.
tag `https://thredds.imk-ifu.kit.edu/thredds/catalog/regclim/raster/global/era5/sfc/single/catalog.xml` (nested)

Second senarion: `CatalogRefs` are not under a dataset element tag and directly come below the `catalog`.
`https://thredds.imk-ifu.kit.edu/thredds/catalog/catalogues/sensor_catalog_ext.xml` (nested)

Third scenario: One single `dataset` tag is located next to `CatalogRef` tags. All are under a `dataset` tag.
`https://thredds.imk-ifu.kit.edu/thredds/catalog/regclim/raster/global/chirps/catalog.xml` (nested)

Fourth scenario: An empty datasets.
`https://thredds.imk-ifu.kit.edu/thredds/catalog/catalogues/bio_geo_chem_catalog_ext.xml` or `https://thredds.atmohub.kit.edu/thredds/catalog/snowfogs/catalog.xml`

Fifth scenario: There is no `CatalogRef` tag and all are `dataset` tag. All of them are under a `dataset` tag.
`https://thredds.imk-ifu.kit.edu/thredds/catalog/climate/raster/global/chelsa/v1.2/catalog.html`

Sixth scenario: A single dataset
`https://thredds.imk-ifu.kit.edu/thredds/catalog/regclim/raster/global/era5/sfc/single/catalog.xml?dataset=regclim/raster/global/era5/sfc/single/era5_sfc_20210101.nc`

Seventh scenario: An aggregated dataset
`https://thredds.imk-ifu.kit.edu/thredds/catalog/catalogues/swabian_moses_2021.xml?dataset=swabian_moses_aggregation`

Eighth scenario: A combination of `caralogRef` and `dataset` tags that is not under a `dataset` tag.It's similar to second scenario but with datasets
`https://thredds.imk-ifu.kit.edu/thredds/catalog/catalogues/transfer.xml`

Ninth scenario: When we have a bunch of single `dataset` tags next to catalogref. It's similar to third scenario but with more datasets.
`https://thredds.imk-ifu.kit.edu/thredds/catalog/regclim/raster/global/hydrogfd/v3.0/catalog.xml` (nested)

Parameters

- **`main_catalog_url`** – TDS Catalog url to start harvesting
- **`nested_check`** – An option for checking nested datasets in TDS (True or False)
- **`auth`** – Authentication for TDS catalog e.g.('user', 'password')
- **`logger_properties`** – A dictionary for logger properties.

- **requests_properties** – A dictionary for requests properties.

all_dirs: list

all_dirs_extensions: list

all_href: list

logger_properties: dict

A dictionary for logger properties. For more information see *Logger*

main_catalog_url: str

TDS Catalog url to start harvesting (*)

nested_check: bool

An option for checking nested datasets in TDS (True or False) (optional)

nested_checker(*url: str*)

A function for returning the depth of nested datasets in TDS for scenarios 1, 3, ,and 9

nested_checker_exceptions(*url: str*)

A function for returning the depth of nested datasets in TDS for scenarios 2 and 8

nested_num: int

nested_num_temp: int

recognition_function(*url: str*, *xml_content*)

A function for recognizing number of scenarios in TDS

requests_properties: dict

To obtain additional information on this topic, refer to the `requests_properties`. The default value is an empty dictionary.

status: str | None

tds2stac.dimensions package

Submodules

tds2stac.dimensions.spatial module

class tds2stac.dimensions.spatial.**Spatial**

Bases: `object`

harvester(*main_dict*, *linestring=None*)

regulator(*main_dict*, *spatial_information*)

A function for regulating the spatial information of a catalog

tds2stac.dimensions.temporal module

class tds2stac.dimensions.temporal.**Temporal**

Bases: object

parse_datetime_with_fallback(*datetime_str*, *primary_format*, *fallback_format*, *tzinfo*)

regulator(*main_dict*, *temporal_format_by_dataname*, *data_name*)

safe_strip(*value*)

tds2stac.extensions package

Submodules

tds2stac.extensions.common_metadata module

class tds2stac.extensions.common_metadata.**CommonMetadata**

Bases: object

item(*item*, *harvesting_vars*, *logger_properties: dict = {}*)

tds2stac.extensions.datacube module

class tds2stac.extensions.datacube.**Datacube**

Bases: object

This class is responsible for adding the datacube extension to the STAC Item. :param item: The STAC Item to be extended. :type item: pystac.Item :param harvesting_vars: The dictionary of the variables and dimensions of the dataset. :type harvesting_vars: dict :param logger_properties: The dictionary of the logger properties. :type logger_properties: dict

item_extension(*item*, *harvesting_vars*, *logger_properties: dict = {}*)

tds2stac.extensions.extra_metadata module

class tds2stac.extensions.extra_metadata.**ExtraMetadata**(*logger_properties: dict = {}*)

Bases: object

A class to add extra metadata to the STAC items and collections. :param logger_properties: A dictionary containing the logger properties. :type logger_properties: dict

collection(*collection: Collection*, *extra_metadata: dict = {}*)

Add extra metadata to the STAC collection.

item(*item: Item*, *extra_metadata: dict = {}*, *harvesting_vars: dict = {}*)

Add extra metadata to the STAC item.

open_json_file(*file_path: str*) → dict

Open the JSON file.

tds2stac.statics package

Submodules

tds2stac.statics.constants module

tds2stac.webservices package

Submodules

tds2stac.webservices.core module

class tds2stac.webservices.core.**ConfigFileWebServicesScraper**(*json_file: str*, *logger_properties: dict = {}*)

Bases: object

A class for getting the list of webservices that used in the tag_config.json file. :param json_file: The path to the tag_config.json file :type json_file: str :param logger_properties: The dictionary of the logger properties. :type logger_properties: dict

aslist()

A function for returning the list of webservices

get_values(*json_obj: dict | list*, *key: str*)

A function for getting the list of values of a specific key in a json file.

json_file: str

The path to the tag_config.json file.To obtain further details on the creation of a tag_config.json file, refer: *Creating the tag_config.json Configuration File: A Step-by-Step Guide*.

logger_properties: dict

The dictionary of the logger properties. You can look at keys in *Logger* class.

class tds2stac.webservices.core.**JSONFileWebServiceListScraper**(*json_file: str*, *logger_properties: dict = {}*)

Bases: object

A class to get all tds2stac_webservice_analyser in tag_config.json file when tds2stac_mode_analyser is get or check.

load_and_process_json()

class tds2stac.webservices.core.**WebServiceContentScraper**(*root: _Element*, *service_url: str*, *json_file: str*, *extensions_list: list*, *harvesting_vars: dict | None = None*, *logger_properties: dict = {}*)

Bases: object

The functionality of the existing class is dependent on the settings specified in the tag_config.json file in order to harvest targeted information from a selected web service. For comprehensive instructions on configuring the tag_config.json file, refer to the following link: *Creating the tag_config.json Configuration File: A Step-by-Step Guide*.

Args:

root (etree._Element): The root of the XML-based web service json_file (str): The path to the tag_config.json file extensions_list (list): The list of extensions to be harvested

from the web service (main keys in the `tag_config.json` file)

harvesting_vars (dict, optional): The dictionary of harvesting variables logger_properties (dict, optional):The dictionary of the logger properties.

extensions_list: list

The list of extensions to be harvested from the web service. Main keys in the `tag_config.json` file. For example `item_datacube_extension` and so on.

harvester(*root*, *service_url*, *json_file*, *ext_name*, *harvesting_vars=None*)

harvesting_vars: dict | None

It's a dictionary that keys are variable names and values are the result of harvesting.

json_file: str

The path to the `tag_config.json` file

list_of_all_tags: List[Any]

logger_properties: dict

The dictionary of the logger properties. You can look at keys in *Logger* class.

root: _Element

Etree root object of the XML-based web service

class tds2stac.webservices.core.**WebServiceListScraper**(*url: str*, *logger_properties: dict = {}*, *requests_properties: dict = {}*)

Bases: object

A class for getting the list of available web services of a TDS catalogs.

Args:

url (str): The catalog URL from TDS to provide its web services logger_properties (dict, optional):The dictionary of the logger properties. requests_properties (dict, optional): A dictionary that modify the requests to URLs.

aslist()

logger_properties: dict

The dictionary of the logger properties. You can look at keys in *Logger* class.

requests_properties: dict

To obtain additional information on this topic, refer to the `requests_properties`. The default value is an empty dictionary.

url: str

url is the url of the TDS catalog

Submodules

tds2stac.assets module

class tds2stac.assets.**Assets**

Bases: object

This class is tasked with the responsibility of incorporating assets into STAC-Collections and STAC-Items.

collection(*harvesting_vars: dict*, *collection_dict: dict*, *stac_catalog: Catalog*, *asset_properties: dict = {}*, *logger_properties: dict = {}*)

This is a function for adding assets to STAC-Collections.

Parameters

- **asset_properties** (*dict*) – A dictionary containing the properties of the assets for more information refer to *asset_properties*.
- **harvesting_vars** (*dict*) – A dictionary containing the variables required for harvesting.
- **collection_dict** (*dict*) – A dictionary containing the properties of the collection.
- **stac_catalog** (*pystac.Catalog*) – A STAC-Catalog.
- **logger_properties** (*dict*) – A dictionary containing the properties of the logger for more information refer to *logger_properties*.

item(*harvesting_vars: dict*, *item: Item*, *url: str*, *Recognizer_output: str | None = None*, *aggregated_dataset_url: str | None = None*, *asset_properties: dict | None = {}*, *logger_properties: dict = {}*)

This is a function for adding assets to STAC-Items.

Parameters

- **harvesting_vars** (*dict*) – A dictionary containing the variables required for harvesting.
- **item** (*pystac.Item*) – A STAC-Item.
- **Recognizer_output** (*dict*) – A dictionary containing the scenario output of the *Recognizer*.
- **aggregated_dataset_url** (*str*) – The URL of the aggregated dataset.
- **url** (*str*) – The URL of the catalog. It will be used for thumbnails
- **asset_properties** (*dict*) – A dictionary containing the properties of the assets for more information refer to *asset_properties*.
- **logger_properties** (*dict*) – A dictionary containing the properties of the logger for more information refer to *logger_properties*.

tds2stac.creator module

class tds2stac.creator.**STACCreator**

Bases: object

A class for creating STAC catalog, -Collections and its -Items from TDS datasets catalogs.

STACCatalog(*url: str*, *stac_id: str*, *stac_title: str | None*, *stac_desc: str | None*, *stac_dir: str*, *stac_existence: bool = False*, *logger_properties: dict = {}*, *requests_properties: dict = {}*)

A function for creating STAC catalog from TDS dataset catalog.

Parameters

- **url** – The URL of the TDS catalog.
- **stac_id** – The ID of the STAC catalog.
- **stac_title** – The title of the STAC catalog.
- **stac_desc** – The description of the STAC catalog.

- **stac_dir** – The directory of saving the STAC catalog.
- **stac_existence** – If it is True, it means that the STAC catalog already exists in the directory and for the harvesting, there is no need to create a new STAC-Catalog and import new collections In the existed STAC-Catalog. False by default.
- **logger_properties** – The properties of the logger. For more information please check the *Logger* class.
- **requests_properties** – The properties of the requests. For more information please check the `requests_properties` class.

STACCollection(*catalog: Catalog*, *collection_id: str*, *collection_title: str*, *collection_description: str*, *stac_existence_collection: bool = False*, *logger_properties: dict = {}*, *extra_metadata: dict = {}*)

This is a function for creating STAC collection from harvested information from TDS dataset catalog. This function returns a dictionary with two keys:

1. collection: The STAC collection

2. existed_items_id_list: The list of the items that already exist in the STAC collection and it is going to be used for the harvesting process.

Parameters

- **catalog** – The STAC catalog.
- **collection_id** – The ID of the STAC collection.
- **collection_title** – The title of the STAC collection.
- **collection_description** – The description of the STAC collection.
- **collection_scientific** – The scientific extension of the STAC collection.
- **stac_existence_collection** – If it is True, it means that the STAC collection already exists in the catalog and for the harvesting, there is no need to create a new STAC-Collection and import new items In the existed STAC-Collection. False by default.
- **logger_properties** – The properties of the logger. For more information please check the *Logger* class.

STACItem(*url: str*, *catalog: Catalog*, *harvesting_vars: dict*, *Recognizer_output: str | None*, *collection_id: str*, *aggregated_dataset_url: str | None = None*, *extension_properties: dict | None = None*, *asset_properties: dict | None = {}*, *logger_properties: dict = {}*, *extra_metadata: dict = {}*, *stac_existence_collection: bool = False*, *collection_bbox_existed: list | None = None*, *collection_interval_time_final_existed: list | None = None*)

This is a function for creating STAC item from harvested data in TDS dataset catalog.

Parameters

- **url** – The URL of the TDS catalog.
- **catalog** – The STAC catalog.
- **harvesting_vars** – The harvested data from TDS catalog.
- **Recognizer_output** – The output of the Recognizer class.
- **collection_id** – The ID of the STAC collection.
- **aggregated_dataset_url** – The URL of the aggregated dataset that whole of data is located there.

- **extension_properties** – The properties of the extensions.
- **asset_properties** – The properties of the assets.
- **logger_properties** – The properties of the logger. For more information please check the *Logger* class.

SaveCatalog(*catalog*, *catalog_dir*, *logger_properties: dict = {}*)

tds2stac.harvester module

class tds2stac.harvester.**CollectionHarvester**(*url: str*, *recognizer: str | None*, *subdirs: list | None = []*, *collection_tuples: list[tuple] | None = None*, *logger_properties: dict = {}*, *requests_properties: dict = {}*)

Bases: `object`

This class harvests data pertaining to Collections from TDS catalogs. Depending on the sort of dataset scenario, it returns one of the five variables below. `collection_id` , `collection_title` , `collection_description` , `collection_url` , and `collection_subdirs`.

Parameters

- **url** (*str*) – TDS catalog URL address
- **recognizer** (*str*) – `status` scenario number of *Recognizer*
- **subdirs** (*list*) – subdirs is a list of url, id, title, and subdirs of a nested dataset
- **collection_tuples** (*list*) – a tuple of STAC collection's auto-generated ID, user-ID, user-Title and user-Description defined by user.
- **logger_properties** (*dict*) – dictionary of logger properties
- **requests_properties** (*dict*) – dictionary of requests properties

collection_description: str | None

collection_id: str | None

collection_id_desc_maker(*url: str*, *collection_tuples: list[tuple] | None = None*, *recognizer_output: str | None = None*)

A function for getting collection id and description from the TDS catalog urls and pre-defined `collection_tuples` for scenarios number 4, 5, 6, 7 ,and 9

Parameters

- **url** (*str*) – TDS catalog URL address
- **collection_tuples** (*list*) – a tuple of STAC collection's auto-generated ID, user-ID, user-Title and user-Description defined by user.
- **recognizer_output** (*str*) – `status` scenario output of Recognizer class

collection_subdirs: list | None

collection_title: str | None

collection_tuples: list[tuple] | None

a tuple of STAC collection's auto-generated ID, user-ID, user-Title and user-Description defined by user.

collection_url: str | None

logger_properties: dict

dictionary of logger properties, more information in *Logger*

recognizer: str | None

`status` scenario output of Recognizer class

requests_properties: dict

To obtain additional information on this topic, refer to the `requests_properties`. The default value is an empty dictionary.

subdirs: list | None

subdirs is a list of url, id, title, and subdirs of a nested dataset

url: str

TDS catalog URL address. Initial point of harvesting e.g. https://thredds.atmohub.kit.edu/thredds/catalog/caribic/IAGOS-CARIBIC_MS_files_collection_20231017/catalog.html (*)

class tds2stac.harvester.**ItemHarvester**(*url: str*, *elem: Element*, *harvesting_vars: dict*, *web_service_dict: dict | None*, *datetime_after: datetime | None = None*, *datetime_before: datetime | None = None*, *spatial_information: list | None = None*, *temporal_format_by_dataname: str | None = None*, *extension_properties: dict | None = None*, *linestring: bool = False*, *requests_properties: dict = {}*, *logger_properties: dict = {}*)

Bases: `object`

This class harvests information about an Item from TDS data catalogs. It ultimately returns a dictionary of harvesting variables, based on the type of dataset scenario and activated extensions.

Parameters

- **url** (*str*) – TDS catalog URL address
- **elem** (*str*) – xml element of the data in dataset
- **harvesting_vars** (*dict*) – dictionary of harvesting variables that is going to be filled
- **web_service_dict** (*dict*) – web service that the user wants to harvest from
- **datetime_after** (*str*) – datetime that the user wants to harvest data after that
- **datetime_before** (*str*) – datetime that the user wants to harvest data before that
- **spatial_information** (*list*) – Spatial information of 2D datasets e.g. [minx, maxx, miny, maxy] or 1D dataset e.g. [x,y]
- **temporal_format_by_dataname** (*str*) – datetime format for datasets that have datetime in their name e.g `e%y%m%d%H.%M%S%f` (optional),
- **extension_properties** (*dict*) – dictionary of extension properties (optional)
- **linestring** (*bool*) – using this attribute, user activate making LineString instead of Polygon (True and False) (optional)
- **logger_properties** (*dict*) – dictionary of logger properties

datetime_after: str | None

datetime that the user wants to harvest data after that

datetime_before: str | None

datetime that the user wants to harvest data before that

elem: Element

xml element of the data in dataset. It's an element of the xml file that is going to be harvested

extension_properties: dict | None

dictionary of extension properties (optional)

harvesting_vars: dict

dictionary of harvesting variables that is going to be filled

linestring: bool

using this attribute, user activate making LineString instead of Polygon (True and False) (optional)

logger_properties: dict

dictionary of logger properties, more information in *Logger*

requests_properties: dict

To obtain additional information on this topic, refer to the `requests_properties`. The default value is an empty dictionary.

spatial_information: list | None

Spatial information of 2D datasets e.g. [minx, maxx, miny, maxy] or 1D dataset e.g. [x,y] (optional)

temporal_format_by_dataname: str | None

datetime format for datasets that have datetime in their name e.g `e%y%m%d%H.%M%S%f` (optional)

usl: str

TDS catalog URL address. Initial point of harvesting e.g. https://thredds.atmohub.kit.edu/thredds/catalog/caribic/IAGOS-CARIBIC_MS_files_collection_20231017/catalog.html (*)

web_service_dict: dict | None

web service that the user wants to harvest from

tds2stac.logger module

class tds2stac.logger.**IgnoreRepeatedLogFilter**

Bases: Filter

filter(*record*)

Determine if the specified record is to be logged.

Returns True if the record should be logged, or False otherwise. If deemed appropriate, the record may be modified in-place.

class tds2stac.logger.**Logger**(*logger_properties: dict[str, Any] | None = {}*)

Bases: object

A class-based logger for TDS2STAC. It supports all the handlers from the standard python logging library.

Parameters

logger_properties (*dict, optional*) – Logger properties. Defaults to `dict()`. It's optional and has the following keys:

logger_msg (str, optional)

logger_handler (str, optional)

logger_name (str, optional)

logger_id (str, optional)

logger_level (str, optional)

logger_formatter (str, optional)

logger_handler_host (str, optional)

logger_handler_port (str, optional)

logger_handler_url (str, optional)

logger_handler_method (str, optional)

logger_handler_secure (bool, optional)

logger_handler_credentials (tuple, optional)

logger_handler_context (tuple, optional)

logger_handler_filename (str, optional)

logger_handler_mode (str, optional)

logger_handler_encoding (str, optional)

logger_handler_delay (bool, optional)

logger_handler_errors (str, optional)

logger_handler_mailhost (str, optional)

logger_handler_fromaddr (str, optional)

logger_handler_toaddrs (str, optional)

logger_handler_subject (str, optional)

logger_handler_timeout (str, optional)

`Null_Handler()`

This is a function to return a NullHandler

`logger_properties: dict[str, Any] | None`

A dictionary that contains all the logger properties.

It is optional and it is set to `None` by default. The following keys are supported:

logger_msg (str, optional):
Logger message. Defaults to `None`. But it is required when you want to log a message.

logger_handler (str, optional):
Logger handler. Defaults to `NullHandler`. Check the following website for more information:

https://docs.python.org/3/library/logging.handlers.html#module-logging.handlers

logger_name (str, optional):
Logger name. Defaults to `INSUPDEL4STAC`. It's required when you choose `HTTPHandler` as logger_handler.

logger_id (str, optional):
Logger id. Defaults to 1. It's required when you choose `HTTPHandler` as logger_handler.

logger_level (str, optional):
Logger level. Defaults to `DEBUG`. It's optional. For more information check the following website:

https://docs.python.org/3/library/logging.html#levels

logger_formatter (str, optional):
Logger format. Defaults to `%(levelname)-8s %(asctime)s t %(filename)s @function %(funcName)s line %(lineno)s - %(message)s`. For more information check the following website:

https://docs.python.org/3/library/logging.html#formatter-objects

logger_handler_host (str, optional):
Logger host. Sets the value to 'None' by default. It is required when `HTTPHandler` or `SocketHandler` are selected as the `logger_handler`. The `logger_handler` will be set to 'NullHandler' if `HTTPHandler` or `SocketHandler` is selected as the `logger_handler` value and neither `logger_handler_host` nor `logger_handler_port` nor are specified.

logger_handler_port (str, optional):
Logger port. Sets the value to 'None' by default. It is required when `HTTPHandler` or `SocketHandler` are selected as the `logger_handler`. The `logger_handler` will be set to 'NullHandler' if `HTTPHandler` or `SocketHandler` is selected as the `logger_handler` value and neither `logger_handler_host` nor `logger_handler_port` are specified.

logger_handler_url (str, optional):
Logger url. Sets the value to 'None' by default. It is required when `HTTPHandler` is selected as the `logger_handler`. The `logger_handler` will be set to 'NullHandler' if `HTTPHandler` is selected as the `logger_handler` value and neither `logger_handler_url` is specified.

logger_handler_method (str, optional):
HTTP methods. It supports sending logging messages to a web server, using either GET or POST semantics. Sets the value to 'None' by default. It is required when `HTTPHandler` is selected as the `logger_handler`. The `logger_handler` will be set to 'NullHandler' if `HTTPHandler` is selected as the `logger_handler` value and `logger_handler_method` is not specified.

logger_handler_secure (bool, optional):
HTTP secure. Sets the value to 'False' by default. It is utilized when `HTTPHandler` or `SMTPHandler` are selected as the `logger_handler`. But it is optional in both logger handlers.

logger_handler_credentials (tuple, optional):
HTTP credentials. Sets the value to 'None' by default. It is utilized when `HTTPHandler` or `SMTPHandler` are selected as the `logger_handler`. But it is optional in both logger handlers.

logger_handler_context (tuple, optional):
HTTP context. Sets the value to 'None' by default. It is utilized when `HTTPHandler` is selected as the `logger_handler`. But it is optional in both logger handlers.

logger_handler_filename (str, optional):
File name. Sets the value to 'None' by default. It is required when `FileHandler` or `WatchedFileHandler` are selected as the `logger_handler`. The `logger_handler` will be set to 'NullHandler' if `FileHandler` or `WatchedFileHandler` is selected as the `logger_handler` value and `logger_handler_filename` is not specified.

logger_handler_mode (str, optional):
File mode. Sets the value to 'None' by default. It is required when `FileHandler` or `WatchedFileHandler` are selected as the `logger_handler`. The `logger_handler` will be set to 'NullHandler' if `FileHandler` or `WatchedFileHandler` is selected as the `logger_handler` value and `logger_handler_mode` is not specified.

logger_handler_encoding (str, optional):
File encoding. Sets the value to 'None' by default. It is utilized when `FileHandler` or `WatchedFileHandler` are selected as the `logger_handler`. But it is optional in both logger handlers.

logger_handler_delay (bool, optional):
File delay. Sets the value to 'False' by default. It is utilized when `FileHandler` or `WatchedFileHandler` are selected as the `logger_handler`. But it is optional in both logger handlers.

logger_handler_errors (str, optional):
File errors. Sets the value to 'None' by default. It is utilized when `FileHandler` or `WatchedFileHandler` are selected as the `logger_handler`. But it is optional in both logger handlers.

logger_handler_mailhost (str, optional):
Mail host. Sets the value to 'None' by default. It is required when `SMTPHandler` is selected as the `logger_handler`. The `logger_handler` will be set to 'NullHandler' if `SMTPHandler` is selected as the `logger_handler` value and `logger_handler_mailhost` is not specified.

logger_handler_fromaddr (str, optional):
Mail from address. Sets the value to 'None' by default. It is required when `SMTPHandler` is selected as the `logger_handler`. The `logger_handler` will be set to 'NullHandler' if `SMTPHandler` is selected as the `logger_handler` value and `logger_handler_fromaddr` is not specified.

logger_handler_toaddrs (str, optional):
Mail to address. Sets the value to 'None' by default. It is required when `SMTPHandler` is selected as the `logger_handler`. The `logger_handler` will be set to 'NullHandler' if `SMTPHandler` is selected as the `logger_handler` value and `logger_handler_toaddrs` is not specified.

logger_handler_subject (str, optional):
Mail subject. Sets the value to 'None' by default. It is utilized when `SMTPHandler` is selected as the `logger_handler`. But it is optional in both logger handlers.

logger_handler_timeout (str, optional):
Mail timeout. Sets the value to 'None' by default. It is utilized when `SMTPHandler` is selected as the `logger_handler`. But it is optional in both logger handlers.

tds2stac.tds2stac module

class tds2stac.tds2stac.**TDS2STACIntegrator**(*TDS_catalog: str*, *stac_dir: str = '/home/docs/checkouts/readthedocs.org/user_builds/tds2stac/checkouts/stable/a*, *stac_id: str = 'TDS2STAC'*, *stac_title: str | None = 'TDS2STAC'*, *stac_description: str | None = None*, *stac_existence: bool = False*, *stac_existence_collection: bool = False*, *collection_tuples: list[tuple] | None = None*, *datetime_filter: list | None = None*, *aggregated_dataset_url: str | None = None*, *depth_number: int | None = None*, *limited_number: int | None = None*, *spatial_information: list | None = None*, *temporal_format_by_dataname: str | None = None*, *item_geometry_linestring: bool = False*, *webservice_properties: dict | None = {}*, *asset_properties: dict | None = {}*, *extension_properties: dict | None = {}*, *logger_properties: dict = {}*, *requests_properties: dict = {}*, *extra_metadata: dict = {}*)

Bases: `object`

This class is the central component of the TDS2STAC. It harvests the TDS catalog and then generates the STAC-Catalog, -Collections, and -Items through the TDS catalogs, based on the user's input. This class mainly defines all configurations related to harvesting and STAC creation. In the first step, it recognizes the scenario of the TDS catalog using *Recognizer*. If it is recognized as a nested collection, *NestedCollectionInspector* is responsible for determining the nested collection's `ID`, `Title`, and `url` of subdirectories. Other procedures follow in succession. For example, *CollectionHarvester* harvests the collection's information and *STACCreator* creates the STAC-Catalog and -Collection. Then, *ItemHarvester* harvests the item's information and *STACCreator* creates the STAC-Item and connect them to the related STAC-Collections. At the end each STAC-Collection will be connected to the main STAC-Catalog.

Parameters

- **TDS_catalog** (*str*) – The URL address of the TDS catalog that will be harvested.
- **stac_dir** (*str, Optional*) – Directory of saving created STAC catalogs.
- **stac_id** (*str, Optional*) – STAC catalog ID. default value is 'TDS2STAC'.
- **stac_title** (*str, optional*) – STAC catalog Title. default value is 'TDS2STAC'.
- **stac_description** (*str, optional*) – STAC catalog description.
- **stac_existence** (*bool, optional*) – Verifying the presence of the STAC catalog in order to update an existing catalog; if not, a new catalog will be generated.
- **stac_existence_collection** (*bool, optional*) – Verifying the presence of the STAC Collection in order to update an existing catalog; if not, a new collection will be generated.
- **collection_tuples** (*list, optional*) – The elements of this tuple comprise the auto-TDS2STAC-generated ID , the user-defined ID, title, and description of the STAC-Collection respectively. (auto-ID, user-ID, user-title, user-description).
- **datetime_filter** (*list, optional*) – Datetime-based filtering of harvesting. It works based on the `modified` tag in each dataset at TDS.
- **aggregated_dataset_url** (*str, optional*) – Dataset's URL of each data entry in the Aggregated datasets of TDS.
- **depth_number** (*int, optional*) – depth number of nested datasets if it is a nested collection. default value is 0.

- **limited_number** (*int, optional*) – The objective is to reduce the quantity of harvested items in each collection. It is beneficial for developing and testing purposes.
- **spatial_information** (*list, optional*) – Spatial information of 2D datasets e.g. [minx, maxx, miny, maxy] or 1D dataset e.g. [x,y]. default value is None.
- **temporal_format_by_dataname** (*str, optional*) – A preferred datetime format for datasets that include the time period in their names. e.g "e%y%m%d%H.%M%S%f"
- **item_geometry_linestring** (*bool, optional*) – Set True to make a LineString geometry for STAC-Items from wms service. Otherwise it makes Polygon geometry for the given Item. default value is False.
- **extension_properties** (*dict, optional*) – A dictionary of properties for extensions. default is None. For more information about the keys, please refer to the *extension_properties*.
- **webservice_properties** (*dict, optional*) – A dictionary of properties for web_service. default is None (optional) For more information about the keys, please refer to the *webservice_properties*.
- **asset_properties** (*dict, optional*) – A dictionary of properties for assets. default is None (optional) For more information about the keys, please refer to the *asset_properties*.
- **logger_properties** (*dict, optional*) – A dictionary of properties for logger. default is None.
- **requests_properties** – A dictionary that modify the requests to URLs. To obtain additional information on this topic, refer to the `requests_properties`. The default value is an empty dictionary.

TDS_catalog: str

TDS catalog URL address. Initial point of harvesting e.g. https://thredds.atmohub.kit.edu/thredds/catalog/caribic/IAGOS-CARIBIC_MS_files_collection_20231017/catalog.html

aggregated_dataset_url: str | None

Dataset's URL of each data entry in the Aggregated datasets of TDS.. default value `None`. The `HTTPServer` is not functional in the aggregated dataset. Therefore, in order to utilize this service as an asset in our STAC-Item, we should employ the `aggregated_dataset_url`, which links the individual datasets to the `HTTPServer` asset of the relevant Item.

asset_properties: dict | None

A dictionary of properties for assets. default is `None`. When it's None, keys look like the following example:

collection_thumbnail (str, optional):
A `thumbnail` asset for STAC-collection sourced from the Web Map Service (WMS) of the TDS. It can be chosen from `wms`, `link`, or `None`. The default value is set to None.

collection_overview (str, optional):
A `overview` asset for STAC-collection sourced from the Web Map Service (WMS) of the TDS. It can be chosen from `wms`, `link`, or `None`. The default value is set to None.

collection_thumbnail_link (str, optional):
This property is reliant upon the values of `collection_thumbnail` and `collection_overview`. When the value of either of these attributes is set to `link`, it allows for the inclusion of a hyperlink to an image for `collection_thumbnail` or `collection_overview`.

collection_overview_link (str, optional):
This property is reliant upon the values of `collection_thumbnail` and

`collection_overview`. When the value of either of these attributes is set to `link`, it allows for the inclusion of a hyperlink to an image for `collection_thumbnail` or `collection_overview`.

collection_custom_assets (list, optional):
This is a list of asset dictionaris that includes the `key`, `href`, and `title`, `role` (as a list), and `media_type` of the asset. The default value is set to None. For more information, refer to the *How to make a custom asset for STAC-Collection and STAC-Item:*.

item_thumbnail (bool, optional):
A `thumbnail` asset for STAC-Items sourced from the Web Map Service (WMS) of the TDS. The default value is set to False.

item_overview (bool, optional):
A `overview` asset for STAC-Items sourced from the Web Map Service (WMS) of the TDS. The default value is set to False.

item_getminmax_thumbnail (bool, optional):
The TDS offers a function that allows users to obtain the minimum and maximum values of the colorbar associated with an image through the use of `metadata`. The aforementioned attribute is contingent upon both the `item_thumbnail` and `item_overview`. The default value is set to False.

assets_list_allowed (list, optional):
This is a list of permissible web services that will be incorporated as assets in the STAC-Item. The `WebServiceScraper` class provides access to the list of available web services. Default value is None.

assets_list_avoided (list, optional):
This is a list of web services that will be excluded from the STAC-Item asset list. The `WebServiceScraper` class provides access to the list of available webservices. Default value is None.

explore_data (bool, optional):
By enabling the `True` setting, the inclusion of Godiva3 as an exploration asset will be implemented.

verify_explore_data (bool, optional):
This argument verifies the availability of the `GetMetadata` function. The provided function facilitates the retrieval of data necessary for generating maps using the Web Map Service (WMS) protocol. However, an error occurs when attempting to open `Godiva3` when this function doesn't work. In order to mitigate such errors, it would be advisable to establish this argument.

jupyter_notebook (bool, optional):
This argument posits the inclusion of the Jupyter Notebook as an asset.

`collection_tuples: list[tuple] | None`

STAC collection auto-generated ID, user-ID, user-Title and user-Description defined by user. It is worth mentioning that in order to obtain the list of automatically generated collection IDs, one can employ the *`NestedCollectionInspector`* for the given TDS Catalog and subsequently utilize this argument. Warning - Identifiers should consist of only lowercase characters, numbers, '_', and '-'. Default value `None`. e.g. (ID, Title, Description)

`datetime_filter: list | None`

Datetime-based filtering. e.g. `['2010-02-18T00:00:00.000Z','2020-02-22T00:00:00.000Z']` Default value `None`. It should be noted it works based on the `modified` tag in each dataset at TDS.

depth_number: int | None

The depth refers to the number of layered datasets. If the collection is nested, this argument is applicable; otherwise, employing this argument would be futile. default value `None` (optional)

extension_properties: dict | None

A dictionary of properties for extensions. default is `None`.

item_extensions (list[str, tuple], optional):
The argument can consist of either a list of extension names (string) or a list of tuples containing three elements: the extension name, the function name or class name associated with the extension, and the Python script required for execution. For more explanation, refer to the *Adding and Configuring Custom Extensions for STAC-Items and STAC-Collections*.

collection_extensions (Union[list, tuple], optional):
It works as same as `item_extensions` argument. For more explanation, refer to the *Adding and Configuring Custom Extensions for STAC-Items and STAC-Collections*.

extra_metadata: dict

A dictionary of extra metadata that you want to add to the STAC-Collection and STAC-Items. It has two main keys, `extra_metadata` that is boolean and `extra_metadata_file` that is the address of `extra_metadata.json` JSON file. For getting more information about making the `extra_metadata.json` file, please refer to *How to create extra_metadata.json file*. By default, if 'extra_metadata' is set to True, the 'extra_metadata.json' file is utilised for the 'extra_metadata_file' key, which is situated in the'sta2stac' main directory.

item_geometry_linestring: Literal[False]

The default value for the LineString geometry in the STAC Items from the WMS service is set to False and the default geometry type for the STAC-Item is Polygon. However, in instances where the item has a POINT geometry, it can be automatically detected. However, in order to obtain the LineString geometry, it is necessary to set this argument to True.

limited_number: int | None

The objective is to reduce the quantity of harvested items in each collection. It is beneficial for developing and testing purposes.. default value `None` (optional)

logger_properties: dict

A dictionary of properties for logger. default is `None`. You can look at keys in *Logger* class.

requests_properties: dict

A dictionary of properties that adjust the requests to URLs. It contains the following keys:

verify (bool, optional):
It is a boolean that if it is True, it verifies the SSL certificate. By default it is False.

timeout (int, optional):
It is an integer that sets the timeout of the requests. By default it is 10 seconds.

auth (tuple, optional):
It is a tuple that contains the username and password for the authentication. By default it is None.

spatial_information: list | None

Spatial information of 2D datasets e.g. [minx, maxx, miny, maxy] or 1D dataset e.g. [x,y]. Default value `None`(optional)

stac_description: str | None

STAC catalog description

stac_dir: str

Directory of saving created STAC catalogs e.g. /path/to/stac/directory/

stac_existence: Literal[False]

Verifying the existence of STAC catalog. If the catalog exists in the directory, it updates a existed catalog, otherwise it creates new catalog. default value `False`

stac_existence_collection: Literal[False]

Verifying the existence of STAC Collections. If the collection exists in the directory, it updates a existed collection, otherwise it creates new collection. default value `False`

stac_id: str

STAC catalog ID. default value `TDS2STAC`

stac_title: str | None

STAC catalog Title. default value `TDS2STAC`

temporal_format_by_dataname: str | None

A preferred datetime format for datasets that include the time period in their names e.g "e%y%m%d%H.%M%S%f". Default value `None` (optional)

webservice_properties: dict | None

A dictionary of properties for web_service. default is `None`.

It has the following keys.

web_service_config_file(str, opntional):
The primary `tag_config.json` file is situated in the primary directory of the installed TDS2STAC. However, the user has the ability to declare an alternative `tag_config.json` file, which allows for customization of the settings. The user can specify the location of their own JSON file in this section. To obtain further details on the creation of a `tag_config.json` file, refer: *Creating the tag_config.json Configuration File: A Step-by-Step Guide*. The default value is set to `tag_config.json` in the root directory of the installed app.

tds2stac.thumbnails module

class tds2stac.thumbnails.**Thumbnails**

Bases: `object`

This class is used to create thumbnail images for STAC-Collections and STAC-Items.

collection(*collection_thumbnail: str*, *collection_overview: str*, *services: Element*, *dataset: dict*, *harvesting_vars: dict*, *collection_id: str*, *url: str*, *catalog: Catalog*, *collection_thumbnail_link: str*, *collection_overview_link: str*, *logger_properties: dict = {}*)

A function to create thumbnail images for STAC-Collections.

Parameters

- **collection_thumbnail** (*str*) – The type of thumbnail image for STAC-Collections. It can be `wms` or `link`.
- **collection_overview** (*str*) – The type of overview image for STAC-Collections. It can be `wms` or `link`.
- **services** (*list*) – A list of services for STAC-Collections.
- **dataset** (*dict*) – A dictionary of dataset information.
- **harvesting_vars** (*dict*) – A dictionary of harvesting variables.

- **collection_id** (*str*) – The ID of STAC-Collections.
- **url** (*str*) – The URL of STAC-Catalog.
- **catalog** (*pystac.Catalog*) – A STAC-Catalog.
- **collection_thumbnail_link** (*str*) – The link of thumbnail image for STAC-Collections when `collection_thumbnail` or `collection_overview` set as `link`.
- **collection_overview_link** (*str*) – The link of overview image for STAC-Collections when `collection_thumbnail` or `collection_overview` set as `link`.
- **logger_properties** (*dict*) – A dictionary of logger properties. For more information, please see *Logger* class.

item(*service: Element*, *dataset: dict*, *harvesting_vars: dict*, *url: str*, *item: Item*, *item_thumbnail: bool*, *item_overview: bool*, *item_getminmax_thumbnail: bool*, *logger_properties: dict = {}*)

A function to create thumbnail images for STAC-Items.

Parameters

- **service** (*list*) – A list of services for STAC-Items.
- **dataset** (*dict*) – A dictionary of dataset information.
- **harvesting_vars** (*dict*) – A dictionary of harvesting variables.
- **url** (*str*) – The URL of STAC-Catalog.
- **item** (*pystac.Item*) – A STAC-Item.
- **item_thumbnail** (*bool*) – A boolean to create thumbnail image for STAC-Items.
- **item_overview** (*bool*) – A boolean to create overview image for STAC-Items.
- **item_getminmax_thumbnail** (*bool*) – A boolean to create thumbnail image for STAC-Items based on minmax.
- **logger_properties** (*dict*) – A dictionary of logger properties. For more information, please see *Logger* class.

tds2stac.utils module

tds2stac.utils.**get_xml**(*url*, *request_properties*)

A function for getting XML content from url

tds2stac.utils.**html2xml**(*url*)

A function for making a an xml URL from html URL

tds2stac.utils.**merge_bboxes**(*bbox1*, *bbox2*)

tds2stac.utils.**merge_intervals**(*interval1*, *interval2*)

tds2stac.utils.**opener_module**(*service_url*, *requests_properties*)

tds2stac.utils.**references_urls**(*url*, *additional*)

tds2stac.utils.**replacement_func**(*url*)

A function for making a an id from catalog URL for collections

tds2stac.utils.**replacement_func_collection_item_id**(*url*)

A function for making a an id from catalog URL for collections

tds2stac.utils.**validate_catalog_url**(*url*, *requests_properties*)

A function for validating the catalog URL

tds2stac.utils.**xml2html**(*url*)

A function for making a an html URL from xml URL

tds2stac.utils.**xml_processing**(*catalog*, *request_properties*)

A function for getting out XML details of a catalog URL

tds2stac.utils.**xml_tag_finder**(*input_xml*, *web_service*, *var_name*)

A function for finding the tag names in all TDS webservices

tds2stac.utils.**xml_tag_name_ncml**(*input_xml*, *var_name*)

A function for finding the tag names in NcML XML files

This API reference is auto-generated from the Python docstrings. The table of contents on the left is organized by module. The section right is sub sections of this page.

CHAPTER

SIX

CONTRIBUTION

6.1 Contribution

6.1.1 Contribution and development hints

Warning: This page has been automatically generated as has not yet been reviewed by the authors of TDS2STAC!

The TDS2STAC project is developed by the Karlsruher Institut für Technologie. It is open-source as we believe that this analysis can be helpful for reproducibility and collaboration, and we are looking forward for your feedback, questions and especially for your contributions.

- **If you want to ask a question, are missing a feature or have**
  comments on the docs, please open an issue at the source code repository.
- **If you have suggestions for improvement, please let us know in an**
  issue, or fork the repository and create a merge request.

6.1.2 Contributing in the development

Note: We use automated formatters to ensure a high quality and maintanability of our source code. Getting familiar with these techniques can take quite some time and you might get error messages that are hard to understand.

We not slow down your development and we do our best to support you with these techniques. If you have any troubles, just commit with `git commit --no-verify` (see below) and the maintainers will take care of the tests and continuous integration.

Thanks for your wish to contribute to this project!! The source code of the *tds2stac* package is hosted at https://codebase.helmholtz.cloud/cat4kit/ds2stac/tds2stac.

This is an open gitlab where you can register via the Helmholtz AAI. If your home institution is not listed in the Helmholtz AAI, please use one of the social login providers, such as Google, GitHub or OrcID.

Once you created an account in this gitlab, you can fork this repository to your own user account and implement the changes.

Afterwards, please make a merge request into the main repository. If you have any questions, please do not hesitate to create an issue on gitlab and contact the maintainers of this package.

Once you created you fork, you can clone it via

```
git clone https://codebase.helmholtz.cloud/<your-user>/tds2stac.git
```

we recommend that you change into the directory and create a virtual environment via:

```
cd tds2stac
python -m venv venv
source venv/bin/activate # (or venv/Scripts/Activate.bat on windows)
```

and install it in development mode with the `[dev]` option via:

```
pip install -e ./tds2stac/[dev]
```

6.1.3 Helpers

Shortcuts with make

There are several shortcuts available with the `Makefile` in the root of the repository. On Linux, you can execute `make help` to get an overview.

Annotating licenses

If you want to create new files, you need to set license and copyright statements correctly. We use `reuse` to check that the licenses are correctly encoded. As a helper script, you can use the script at `.reuse/add_license.py` that provides several shortcuts from `.reuse/shortcuts.yaml`. Please select the correct shortcut, namely

- If you create a new python file, you should run

  ```
  python .reuse/add_license.py code <file-you-created>.py
  ```

- If you created a new file for the docs, you should run

  ```
  python .reuse/add_license.py docs <file-you-created>.py
  ```

- If you created any other non-code file, you should run

  ```
  python .reuse/add_license.py supp <file-you-created>.py
  ```

If you have any questions on how licenses are handled, please do not hesitate to contact the maintainers of *tds2stac*.

6.1.4 Fixing the docs

The documentation for this package is written in restructured Text and built with sphinx and deployed on readthedocs.

If you found something in the docs that you want to fix, head over to the `docs` folder, install the necessary requirements via `pip install -r requirements.txt ../[docs]` and build the docs with `make html` (or `make.bat` on windows). The docs are then available in `docs/_build/html/index.html` that you can open with your local browser.

Implement your fixes in the corresponding `.rst`-file and push them to your fork on gitlab.

6.1.5 Contributing to the code

We use automated formatters (see their config in `pyproject.toml`), namely

- Black for standardized code formatting
- blackdoc for standardized code formatting in documentation
- Flake8 for general code quality
- isort for standardized order in imports.
- mypy for static type checking on type hints
- reuse for handling of licenses
- cffconvert for validating the `CITATION.cff` file.

We highly recommend that you setup pre-commit hooks to automatically run all the above tools every time you make a git commit. This can be done by running:

```
pre-commit install
```

from the root of the repository. You can skip the pre-commit checks with `git commit --no-verify` but note that the CI will fail if it encounters any formatting errors.

You can also run the `pre-commit` step manually by invoking:

```
pre-commit run --all-files
```

6.1.6 Updating the skeleton for this package

This package has been generated from the template https://codebase.helmholtz.cloud/hcdc/software-templates/python-package-template.git.

See the template repository for instructions on how to update the skeleton for this package.

PYTHON MODULE INDEX

t

INDEX

E

F

G

H

I

N

O

P

R

S

T

U

V

W

X