ASSIMILA

DATACUBE INTERFACE GUIDE

Jane Lewis

September 2019

Document Information

Project Number	2016-042	**Acronym**	PRISE
Full Title	Datacube Interface Guide		
Project URL	\\192.168.11.2\ELC\Shared\Projects\2016-042 IPP PRISE		
Document URL	05\07\02		

Date of Delivery	**Contractual**		**Actual**	

Authors	Jane Lewis (Assimila)

Reviewers	Taylor Day (Assimila)

Revision History			
Date	**Issue**	**Author**	**Description**
Sept 2019	0.1A	J Lewis	DataCube version 2 & DQTools
Sept 2019	1.0	J Lewis	Issued
Oct 2019	1.1	J Lewis	Specify Python version required, and clarify other points.
Feb 2021	1.2	J Lewis	Update Python library dependencies

References		
Reference	**Document**	**Description**
1.	05\07\01 v2.0	UserGuide and API (Knowledge; deprecated)

Contents

User Guide for Assimila DataCube

1. Introduction

This document is designed to give users an overview of the features of the Assimila DataCube and the tools used to access it. A keyfile is required for access.

For details on the use of the web interface and the API, please refer to the User Guide and API documentation (ref 1).

It is assumed in this guide that the user understands Python code.

2. Assimila DataCube

2.1. Overview

The Assimila DataCube may be used by authorised users from their own codebase by obtaining the Python toolset from the GitHub repository (https://github.com/Assimila/DQTools). Their code may query the data and retrieve datasets (herein known as 'products') from the DataCube, subject to permission on those data (as specified by Assimila).

2.2. DataCube Access Requirements

The Assimila DataCube is accessed via a http client which is implemented in Python 3.6.7[1]. There are a number of requirements a user will need to meet in order to submit a DataCube request via the http client. These are detailed below:

- **The Assimila client tools**

  These are the python classes which work together to provide a straightforward interface to the Assimila DataCube. They are Register, Search and Dataset.

- **Python 3.7 environment** with the following additional modules installed**:**
  - pickleshare
  - cloudpickle==1.2.1
  - xarray==0.12.1
  - NumPy
  - pandas
  - PyYaml
  - requests==2.22.0

  A Python environment can be set up using any Python distribution but a recommended way to do this is by using Anaconda. Details of how to do this are outside of the scope of this document but information can be found here.

  To make setting up the environment more straightforward, there is a modules definition file in the DQTools\condaenv folder which Anaconda can use once it is installed; full instructions

[1] A version of DQTools for the older Python 3.5 is on a separate branch in the repository with its own release. This may be deprecated in future versions of DQTools. Compatibility with the primary release branch will be attempted but cannot be guaranteed.

are here. For example on a PC, if you have installed Anaconda to C:\Anaconda3, and have cloned DQTools to C:\workspace\DQTools, running

```
cd c:\Anaconda3\condabin
conda env create -f c:\workspace\DQTools\condaenv\win\dqtools_0.5.yml
```

will create the conda environment.

You will need to activate the conda environment in order to run the DQTools code:

```
conda activate dqtools_0.5
```

- **A security key file** containing:
    - IP address
    - socket number
    - access, permissions and user string
    - associated authentication string

  This file will be provided by Assimila to authenticated users and will be called '.assimila_dq'. This authentication file should be copied into the same directory as the Assimila DQTools keyfile to replace the default one: in the connect/ sub-directory. A file path can be specified to the client module if the key file lives elsewhere on the user's system or has a different name. This file specifies the user's permissions with reference to the DataCube. Permissions are categorised so that a user can: read and/or create and/or write. The public code distribution provides a time-limited read/browse-only keyfile.

- **Verified IP address**

  You must tell Assimila your IP Address so we can grant access through the firewall (enabled to prevent malicious attacks).

With the above four requirements met, a user can utilise the Assimila DQTools to retrieve products from the DataCube. The following sections detail how to do this.

2.3. Getting Your Code Ready

Get the DQTools code from the GitHub repository https://github.com/Assimila/DQTools and always use the latest release tag which can be found on the 'Releases' page.

git clone https://github.com/Assimila/DQTools.git

git checkout tags/<release tag>

As is normal for Python, you will need to import the DQTools classes. To do this, we suggest finding the path from your code file to the parent of the folder of DQTools and then adding this to your system path.

For example, if the file needing to do the import is in /workspace/one/two/three/ and the tools are in /workspace/DQTools/, you would write this:

```
import os
import sys
wkspace_root = os.path.join(__file__, '../../..')
sys.path.append(os.path.normpath(os.path.join(wkspace_root, 'DQTools')))
from DQTools import Dataset, Search, Register
```

2.4. Accessing Products from the DataCube

This section explains each of the DQTools classes and gives examples of their use.

2.4.1. Search

Several search methods are provided so that a list of available products, sub-products and tiles can be found quickly. This enables the exact dataset to be specified for retrieval.

```
s = Search()                # create a Search object
tiles = s.tiles()           # find all the tiles in the DataCube
prods = s.products()        # find all the products in the DataCube
sub_prods = s.subproducts() # find all the sub-products in the DataCube
```

Each returned item is a pandas DataFrame which may be queried for its contents, e.g.

```
print(prods.name)
```

yields:

0	chirps
1	tamsat
2	era5
3	gfs
4	ecmwf_operational_archive
5	biofix
6	generations
7	degree_day_delta
8	larval_emergence

The specific sub-products available for a product are found with:

```
prod_sub_prods = s. get_subproduct_list_of_product(<prod name>)
```

where using, for example, ‘era5’, yields:

['skt',
 't2m',
'skt_ensemble_mean',
'land_sea_mask',
't2m_ensemble_spread',
't2m_ensemble_mean',
'skt_ensemble_spread']

Product DataFrame columns are: idproduct, name, longname, description, keywords, link.

Subproduct DataFrame columns are: idsubproduct, name, longname, description, units, minvalue, maxvalue, keywords, link, datascalefactor, dataoffset, datafillvalue, idproduct (parent), frequency. Note that scaling and offset is for informaion only as all calculations are done automatically.

Tile DataFrame columns are: idtile, name, idboundingbox (internal use only).

2.4.2. Dataset

Once you have identifed the product and sub-product needed, you can create a Dataset object to gather the metadata and then to retrieve the data: this is a two step process. When creating the Dataset object, you may optionally specify a tile or region (though this latter is not yet fully tested), a resolution, and a keyfile (if you've not put it in the recommended location or have changed its name). The human readable file of available regions is DQTools\regions\regions.yaml. The name of the product, sub-product and tile MUST exist in the DataCube inventory.

Specifying the resolution of the output data required will ultimately perform a GDAL Warp inside the DataCube to give you the required resolution within the bounds defined in either tile or region.

Note that if a tile or region is specified, then metadata pertains only to that tile or region. If no tile or region is defined then metadata is returned for the entire subproduct extent.

Both the Dataset creation and its methods use Python named arguments.

Create Dataset

Examples:

Create a Dataset object for a complete (global) tamsat-rfe dataset:

```
tamsat = Dataset(product='tamsat',
                 subproduct='rfe')
```

Create a Dataset object for Kenyan *Tuta Absoluta* generations:

```
gen_ken = Dataset(product='generations',
                  subproduct='gen_tuta_absoluta',
                  tile='ken_prise')
```

This loads the product's metadata into the Dataset object where it may be interrogated. To see the complete information:

```
print(tamsat)
```

or for specific metadata items:

```
tamsat.first_timestep
tamsat.last_timestep
gen_ken.last_gold
tamsat.all_subproduct_tiles
gen_ken.description
gen_ken.frequency
```

Note that metadata such as '`all_subproduct_tiles`' is subject to the comment above i.e. specifying the tile means that it will return only that tile, not specifying the tile means that it will return all available tiles.

Retrieve data

The Dataset object provides a method '`get_data()`' which reads the dataset from the Assimila DataCube and returns it as a Python xarray. The details of the xarray library are beyond the scope of this document but help is readily available.

The '`get_data()`' method must be given a start and end time and then will optionally accept region or tile, and resolution with the same behaviour as detailed above. Note that specifying

different parameters to those in the Dataset creation is acceptable though possibly a little odd. Using a region for which there is no data will result in an empty xarray. If you fail to specify either a tile or a region, you will be attempting to retrieve all data available for the dataset and this may well be global. Requesting large datasets will take time and the resulting data may be too large for the recipient PC's memory: it is strongly advised to make requests for the minimum times/areas you need.

The start and end times must be given as Python datetime objects, see help. Typically one would use time +/- a metadata timestamp e.g.

```
gen_ken.get_data(start=gen_ken.last_timestep - dt.timedelta(days=10),
                 stop=gen_ken.last_timestep)
```

would return data for the most recent 10 days, for the kenya tile since this was specified in the dataset creation.

Or one may use specific dates e.g.

```
tamsat.get_data(start=dt.datetime(2018, 8, 1),
                stop=dt.datetime(2018, 8, 1),
                tile='gha_prise')
```

would return the data for one day for the ghana tile which we specify here since it was not done during dataset creation.

To reiterate: if a tile or region is not given at dataset creation or when fetching the data, the system will attempt to serve *all* available data.

To obtain zonally averaged data, firstly the Dataset object must be initialised to the correct dataset (obviously including the tile!), then the request for data MUST contain both a country over which to do the averaging and a resolution to ensure data is sampled to get at least one pixel in the smallest of areas. We recommend a value of 0.01. e.g.

```
import datetime as dt

gen_ken.get_data(dt.datetime(2018, 8, 1), dt.datetime(2018, 8, 5),
                 res=0.01, country='kenya')

print(gen_ken.data)
```

Following the '`get_data()`' method call, the data exists in the Dataset's '`data`' attribute.

Your keyfile must provide the permission to do this.

Store data

By mathematically manipulating existing datasets, or creating new datasets, you may wish to store the information in the Assimila DataCube. The datasets and applicable tiles need to be registered initally (see below) and then one may use the '`put()`' method.

After registration, create a Dataset object to connect to the target data

```
new_dataset = Dataset(product='mynewdata', subproduct='datasubproduct')
```

Then name it, load up the data into the object, and write to the Assimila DataCube.

```
new_data = <some_calculation_result>
```

```
new_data.name = 'datasubproduct'    # name can be anything but should agree
                                      with Dataset's subproduct argument
new_dataset.data = new_data.to_dataset()
new_dataset.put()
```

Your keyfile must provide the permission to do this.

2.4.3. Register

If creating a new dataset for storage in the Assimila DataCube, it must first be registered. If it is to cover an area not currently included, a tile definition will also be required. An example is given in the DQTools/GettingStarted folder.

The salient points are:

- The tile must be defined with a name and bounding box.
- A product must be defined with a unique name, and any sub-products it may contain must have unique names within the product.
- Both the tile and product are registered by passing each of the definition files to a new Register object. The tile must be registered first.

Your keyfile must provide the permission to do this.

2.5. DataCube Inventory

The DataCube inventory is a database which holds information on each product and its sub-product(s), known as metadata, including (but not limited to):

- **Product / sub-product name –** each product in the DataCube has a unique name to identify it. The sub-product names are unique for the parent product but may be the same across products. Thus a dataset is uniquely identified by its product/sub-product name combination.
- **Longname -** a more explanatory name for the product or sub-product
- **Description -** free text to explain the product or sub-product
- **Keywords** – words that describe each product or sub-product so that datasets can be filtered using a keyword such as 'temperature' to return all products or sub-products that contain some data associated with 'temperature'.
- **Link** - some products or sub-products may have a provider website in which case this is held.
- **Tiles -** the tiles for which the dataset is available
- **Timesteps available** - first, last and frequency
- **Last Gold** - this is the timestep at which the dataset is most up to date: it applies to those which are derived from others where substitutions may be made to get a best estimate initially, but subsequently precursor data is obtained and the calculations redone.

Note that there are other fields in the DataCube inventory that are not described here. This is because many of the fields are either self-explanatory or are only of interest to expert users.

3. Support

If you have questions about this or if you have difficulties understanding this guide, please contact the Assimila team via prise@assimila.eu

Annex A: Method documentation

```
class Dataset:
    """
    The class to interact with Assimila DataCube data.
    """

    def __init__(self, product, subproduct, region=None, tile=None, res=None,
                 key_file=None):
        """
        Connect to the datacube and extract metadata for this particular
        product/subproduct.

        Attributes passed from the caller are recorded in self:
        self.product: name of the product
        self.subproduct: name of subproduct
        self.region [optional]: name of region required
        self.tile [optional]: name of tile required


        NOTE: If a region/tile is defined, then metadata pertains only to
        that region or tile. If no region/tile is defined then metadata is
        returned for the entire subproduct extent.

        Empty attributes created for
        - self.data: The xarray DataSet
        - self.timesteps: The timesteps of data available

        :param product:     product name (str)

        :param subproduct:  sub product name (str)

        :param region:      optional - the name of a region for
                            data/metadata, as defined in the regions
                            directory (NOTE: writing data for regions is not
                            possible, unless the bounds exactly match a tile...
                            in which case just use tile to define our spatial
                            extent!)

        :param tile:        optional - the tile to extract data/metadata for
                            (must match datacube record)

        :param res:         optional - the resolution of the output data
                            required. This will ultimately execute a GDAL Warp
                            inside the datacube to give you the required
                            resolution within the bounds defined in either tile
                            or region.

        :param key_file:    optional - Assimila DQ key file required to access
                            theHTTP server. Allows keyfile to be in a different
                            location as used by the QGIS Plugin.

        """


    def get_data(self, start, stop,
                 region=None, tile=None, res=None,
                 country=None):
        """
        Extract data from the datacube to the specification supplied.

        :param start:   Start datetime for dataset

        :param stop:    Stop datetime for dataset

        :param region:  optional - geographic region, do not use tile too
```

```
:param tile:    optional - specific tile, do not use region too
                Tile or region are only needed here if not already
                given when creating the Dataset object.

:param res:     optional - resolution required
                If providing a country and therefore expecting zonal
                averaging, it is recommended to set this value to 0.01
                to super-sample the data and ensure each county has
                at least one pixel.

:param country: optional - if country name is supplied, the returned
                dataset will have been zonally averaged according to
                county definitions within that country. The country
                name is case insensitive but must be one for which
                the system has a shapefile defining its counties.

:return: xarray of data
"""
```