

---

# Isru Documentation

*Release 0.6.2*

**loicdtx**

**Aug 04, 2020**



<b>1</b>	<b>User Guide</b>	<b>3</b>
<b>2</b>	<b>API reference</b>	<b>5</b>
<b>3</b>	<b>Basic Usage</b>	<b>15</b>
<b>4</b>	<b>Order data using a polygon</b>	<b>17</b>
<b>5</b>	<b>Indices and tables</b>	<b>21</b>
	<b>Index</b>	<b>23</b>



*Query, order and download Landsat surface reflectance data programmatically*

lsru allows interaction with Usgs and Espa APIs programmatically from python. It has 3 main classes:

- `Usgs` is the interface to the USGS json API. It is mostly used to query the Landsat catalog for available scenes intersecting with a given spatio-temporal window.
- `Espa` is the interface to the ESPA API. Espa is a platform that proposes on demand pre-processing of Landsat data to surface reflectance. Orders can be placed directly from python using that class.
- `Order` is the interface to each individual orders placed to the espa platform; it allows retrieving order status and downloading corresponding scenes.

lsru also contains various utilities to smoothen workflows for various use cases of the module.



### 1.1 Installation

Activate a virtualenv (optional but preferable) and run:

```
pip install lsru
```

### 1.2 Setup

The package requires a configuration file in which usgs credentials are written. By default the file is called `~/.lsru` (this can be modified if you want to join this configuration with the configuration of another project) and has the following structure.

```
[usgs]
username=your_usgs_username
password=your_very_secure_password
```





## 2.1 Isru

Interfaces to Usgs and Espa APIs

<i>Usgs</i> ([version, conf])	Interface to the Usgs API
<i>Usgs.login</i> ()	Login to the Usgs api
<i>Usgs.search</i> (collection, bbox[, begin, end, ...])	Perform a spatio temporal query on Landsat catalog
<i>Usgs.get_collection_name</i> (num)	Get Earth Explorer Landsat collection names
<i>Espa</i> ([conf])	Interface to the Espa API
<i>Espa.order</i> (scene_list, products[, format, ...])	Place a pre-processing order to espa
<i>Espa.get_available_products</i> (scene_list)	Get the list of available products for each elements of a list of scene ids
<i>Order</i> (orderid[, conf])	Class to deal with espa orders
<i>Order.download_all_complete</i> (path[, unpack, ...])	Download all completed scenes of the order to a folder
<i>Order.cancel</i> ()	Cancel the order

### 2.1.1 Isru.Usgs

```
class Isru.Usgs (version='stable', conf='/home/docs/Isru')
```

Interface to the Usgs API

See documentation of the `search` method for basic usage

#### Parameters

- **version** (*str*) – API version to use, defaults to 'stable'
- **conf** (*str*) – Path of the configuration file containing usgs login credentials

#### USER

Usgs username

**Type** str

**PASSWORD**

Usgs password

**Type** str

**endpoint**

API endpoint

**Type** str

**key**

API key. Required to perform a search and obtained by running the `login()` method

**Type** str

**key\_dt**

Time at which the key was generated

**Type** datetime.datetime

**\_\_init\_\_** (*version='stable', conf='/home/docs/lsru'*)

Initialize self. See `help(type(self))` for accurate signature.

## Methods

<code>__init__</code> ( <i>version, conf</i> )	Initialize self.
<code>get_collection_name</code> ( <i>num</i> )	Get Earth Explorer Landsat collection names
<code>login</code> ()	Login to the Usgs api
<code>search</code> ( <i>collection, bbox[, begin, end, ...]</i> )	Perform a spatio temporal query on Landsat catalog

## Attributes

<code>key_age</code>	Determines the age of API key
----------------------	-------------------------------

### 2.1.2 Isru.Usgs.login

`Usgs.login()`

Login to the Usgs api

This method is necessary to obtain an API key (automatically saved in the `key` attribute), and send other queries to the API

**Returns** True if query was successful, False otherwise

**Return type** bool

### 2.1.3 Isru.Usgs.search

`Usgs.search`(*collection, bbox, begin=None, end=None, max\_cloud\_cover=100, months=None, starting\_number=1, max\_results=50000*)

Perform a spatio temporal query on Landsat catalog

**Parameters**

- **collection** (*str*) – Landsat collection to query. Use LANDSAT\_8\_C1, LANDSAT\_ETM\_C1 and LANDSAT\_TM\_C1 for OLI, ETM+, and TM respectively
- **bbox** (*tuple*) – A bounding box in the form of a tuple (left, bottom, right, top)
- **begin** (*datetime.datetime*) – Optional begin date
- **end** (*datetime.datetime*) – Optional end date
- **max\_cloud\_cover** (*int*) – Cloud cover threshold to use for the query
- **months** (*list*) – List of month indices (1,12) for only limiting the query to these months
- **max\_results** (*int*) – Maximum number of scenes to return
- **starting\_number** (*int*) – Used to determine the result number to start returning from. Is meant to be used when the total number of hits is higher than `max_results`, to return results in a paginated fashion

### Example

```
>>> from lsru import Usgs
>>> import datetime
>>> usgs = Usgs()
>>> usgs.login()
>>> scene_list = usgs.search(collection='LANDSAT_8_C1',
>>>                          bbox=(3.5, 43.4, 4, 44),
>>>                          begin=datetime.datetime(2012,1,1),
>>>                          end=datetime.datetime(2016,1,1))
>>> print(scene_list)
```

**Returns** List of scenes with complete metadata

**Return type** list

### 2.1.4 Isru.Usgs.get\_collection\_name

**static** `Usgs.get_collection_name` (*num*)  
Get Earth Explorer Landsat collection names

**Parameters** `num` (*int*) – Landsat spacecraft number (4, 5, 7 or 8)

**Returns**

Earth Explorer collection name formatted to i.e. pass to the search method

**Return type** str

### 2.1.5 Isru.Espa

**class** `lsru.Espa` (*conf*='~/home/docs/lsru')  
Interface to the Espa API

Espe is a platform providing on demand pre-processing of Landsat surface data. This class uses the API of the espe platform to query and place orders programatically

**USER**

Usgs username

**Type** str

**PASSWORD**

Usgs password

**Type** str

**host**

API host url

**Type** str

**Parameters** **conf** (*str*) – Path of the config file containing usgs credentials

`__init__` (*conf*='~/home/docs/Isru')

Initialize self. See help(type(self)) for accurate signature.

**Methods**

<code>__init__</code> ( <i>conf</i> )	Initialize self.
<code>get_available_products</code> ( <i>scene_list</i> )	Get the list of available products for each elements of a list of scene ids
<code>order</code> ( <i>scene_list</i> , <i>products</i> [, <i>format</i> , <i>note</i> , ...])	Place a pre-processing order to espa

**Attributes**

<code>formats</code>	Get a list of file formats supported by the platform
<code>orders</code>	Get a list of current orders
<code>projections</code>	Get a dictionary of projections supported by the platform
<code>resampling_methods</code>	Get a list of resampling methods supported by the platform
<code>user</code>	Get Usgs user details

**2.1.6 Isru.Espa.order**

`Espa.order` (*scene\_list*, *products*, *format*='gtiff', *note*=None, *resampling*='nn', *resolution*=None, *projection*=None, *extent*=None, *extent\_units*='dd', *verbose*=False)

Place a pre-processing order to espa

**Parameters**

- **scene\_list** (*list*) – List of Landsat scene ids
- **products** (*list*) – List of products to order for pre-processing See `Espa.get_available_products()` to get information on available products
- **format** (*str*) – Pre-processing file format. See `Espa.formats` for information on available formats
- **note** (*str*) – Optional human readable message to pass to the order
- **resampling** (*str*) – Resampling method to be used when reprojecting or resizing ordered images. See `Espa.resampling_methods` for valid values.

- **resolution** (*float*) – Output resolution (optional). If specified, the pre-processing order will be resized to the specified resolution. If set to None (default), no resizing is performed and products are processed at their original resolution (usually 30m).
- **projection** (*dict*) – Optional dictionary with projection name and projection parameter values. Ordered products are re-projected to the specified projection when set. See `Espa.projections` for list and format of supported projections
- **extent** (*tuple*) – Bounding box to use to crop the pre-processed products bounding box is in the form of a (left, bottom, right, top) tuple. This is optional and requires a projection to be set.
- **extent\_units** (*str*) – Units of the provided extent. 'dd' (decimal degrees) is the default. If 'meters' bounds are specified according to the coordinate reference system space.
- **verbose** (*bool*) – Prints the json body being sent. Useful for debugging purposes

### Example

```
>>> from lsru import Espa, Usgs
>>> import datetime
>>> espa = Espa()
>>> usgs = Usgs()
>>> usgs.login()
>>> scene_list = usgs.search(collection='LANDSAT_8_C1',
...                          bbox=(3.5, 43.4, 4, 44),
...                          begin=datetime.datetime(2014,1,1),
...                          end=datetime.datetime(2018,1,1))
>>> scene_list = [x['displayId'] for x in scene_list]
>>> order = espa.order(scene_list, products=['sr', 'pixel_qa'])
```

**Returns** The method is mostly used for its side effect of placing a pre-processing order on the espa platform. It also returns a the `lsru.Order` instance corresponding to the order

**Return type** *lsru.Order*

## 2.1.7 lsru.Espa.get\_available\_products

`Espa.get_available_products` (*scene\_list*)

Get the list of available products for each elements of a list of scene ids

**Parameters** `scene_list` (*list*) – List of scene ids

### Example

```
>>> from lsru import Espa
>>> espa = Espa()
>>> print(espa.get_available_products([
...     'LE07_L1TP_029030_20170221_20170319_01_T1'
... ]))
```

**Returns**

**Information on products available for each element of the input** list provided

Return type dict

## 2.1.8 Isru.Order

**class** `Isru.Order` (*orderid*, *conf*='/home/docs/Isru')

Class to deal with espa orders

**orderid**

Espa order ID

Type str

**Parameters**

- **orderid** (*str*) – Espa order ID
- **conf** (*str*) – Path to file containing usgs credentials

**\_\_init\_\_** (*orderid*, *conf*='/home/docs/Isru')

Initialize self. See help(type(self)) for accurate signature.

**Methods**

<code>__init__(orderid[, conf])</code>	Initialize self.
<code>cancel()</code>	Cancel the order
<code>download_all_complete(path[, unpack, ...])</code>	Download all completed scenes of the order to a folder

**Attributes**

<code>is_complete</code>	Check if order has status complete
<code>items_status</code>	
<code>status</code>	Get the current status of the order
<code>urls_completed</code>	Get list of item's url whose status is complete

## 2.1.9 Isru.Order.download\_all\_complete

`Order.download_all_complete` (*path*, *unpack*=False, *overwrite*=False, *check\_complete*=True)

Download all completed scenes of the order to a folder

**Parameters**

- **path** (*str*) – Directory where data are to be downloaded
- **unpack** (*bool*) – Unpack downloaded archives on the fly
- **overwrite** (*bool*) – Force overwriting existing files even when they already exist? Defaults to False
- **check\_complete** (*bool*) – When local files exist and `overwrite` is set to `False`, check whether local and remote files sizes match? File is re-downloaded when sizes are different. Only makes sense if `overwrite` is set to `False`. Defaults to `True`. Also note that checking file size takes time (a few milliseconds probably), so that you'll save time setting this argument to `False` in case you're sure previous downloads are complete. Note that this option

does not work when `unpack` is set to `True`

**Returns** Used for its side effect of batch downloading data, no return

### 2.1.10 Isru.Order.cancel

`Order.cancel()`

Cancel the order

Orders are processed in the order they were placed. Cancelling an order may be useful when the order is blocking other orders

**Returns** The response of the API to the cancellation order

**Return type** dict

## 2.2 utils

<code>utils.bounds(geom)</code>	Return a bounding box from a geometry
<code>utils.geom_from_metadata(meta)</code>	Return a geometry from a Landsat scene metadata as returned by USGS api
<code>utils.is_valid(id)</code>	Landsat scene id validity checker
<code>utils.url_retrieve(url, filename[, ...])</code>	Generic file download function
<code>utils.url_retrieve_and_unpack(url, path[, ...])</code>	Generic function to combine download and unpacking of tar archives

### 2.2.1 Isru.utils.bounds

`lsru.utils.bounds(geom)`

Return a bounding box from a geometry

Adapted from <https://gis.stackexchange.com/a/90554/17409>

**Parameters** `geom` (*dict*) – Geojson like geometry

#### Example

```
>>> from lsru.utils import bounds
>>> geom = {'coordinates': [[ [3.34481, 43.96708],
...                           [6.17992, 45.23413],
...                           [5.55366, 43.56654],
...                           [3.34481, 43.96708]]],
...        'type': 'Polygon'}
>>> print(bounds(geom))
```

**Returns** Bounding box (left, bottom, right, top)

**Return type** Tuple

## 2.2.2 Isru.utils.geom\_from\_metadata

`lsru.utils.geom_from_metadata` (*meta*)

Return a geometry from a Landsat scene metadata as returned by USGS api

**Parameters** `meta` (*dict*) – Landsat scene metadata as returned by Usgs Api

### Example

```
>>> from lsru import Usgs
>>> from lsru.utils import geom_from_metadata
>>> import datetime
>>> from shapely.geometry import shape
>>> from pprint import pprint
```

```
>>> usgs = Usgs()
>>> usgs.login()
>>> scene_list = usgs.search(collection='LANDSAT_8_C1',
...                          bbox=(3.5, 43.4, 4, 44),
...                          begin=datetime.datetime(2012,1,1),
...                          end=datetime.datetime(2016,1,1))
>>> geom = geom_from_metadata(scene_list[0])
>>> s = shape(geom)
>>> pprint(geom)
>>> print(s.is_valid)
```

**Returns** GeoJson like geometry. CRS is always in longlat (EPSG 4326)

**Return type** dict

## 2.2.3 Isru.utils.is\_valid

`lsru.utils.is_valid` (*id*)

Landsat scene id validity checker

**Parameters** `id` (*str*) – Landsat scene id

### Example

```
>>> from lsru import Usgs
>>> from lsru.utils import is_valid
>>> import datetime
>>> import requests
>>> from pprint import pprint
>>> usgs = Usgs()
>>> usgs.login()
>>> scene_list = usgs.search(collection='LANDSAT_8_C1',
...                          bbox=(3.5, 43.4, 4, 44),
...                          begin=datetime.datetime(2012,1,1),
...                          end=datetime.datetime(2016,1,1))
>>> pprint(scene_list[2])
>>> print(is_valid(scene_list[2]['displayId']))
```

**Returns** Whether the provided scene id is valid or not



**Return type** bool

## 2.2.4 Isru.utils.url\_retrieve

`Isru.utils.url_retrieve(url, filename, overwrite=False, check_complete=True)`

Generic file download function

Similar to `url_retrieve` from standard library with additional checks for already existing files and incomplete downloads

### Parameters

- **url** (*str*) – Url pointing to file to retrieve
- **filename** (*str*) – Local file name under which the downloaded content is written
- **overwrite** (*bool*) – Force overwriting local file even when it already exists? Defaults to False
- **check\_complete** (*bool*) – When local file exists and `overwrite` is set to False, check whether local and remote file sizes match? File is re-downloaded when sizes are different. Only makes sense if `overwrite` is set to False. Defaults to True

**Returns** The filename

**Return type** str

## 2.2.5 Isru.utils.url\_retrieve\_and\_unpack

`Isru.utils.url_retrieve_and_unpack(url, path, overwrite=False)`

Generic function to combine download and unpacking of tar archives

Downloads the tar archive as a memory object and extracts its content to a new directory. Directory name is the remote file name with stripped extension

### Parameters

- **url** (*str*) – Url pointing to tar file to retrieve
- **path** (*str*) – Path to directory under which a new directory containing the archive content will be created
- **overwrite** (*bool*) – Force overwriting local files even when the output directory already exist? Defaults to False

**Returns** The path containing extracted content

**Return type** str



This example shows how to query the data from a bounding box, order surface reflectance processing for full scenes without reprojection and download.

### 3.1 Define variables and query available scenes to Usgs

```
from lsru import Usgs
import datetime

# Define query extent
bbox = (3.5, 43.4, 4, 44)

# Instantiate Usgs class and login
usgs = Usgs()
usgs.login()

# Query the Usgs api to find scene intersecting with the spatio-temporal window
scene_list = usgs.search(collection='LANDSAT_8_C1',
                        bbox=bbox,
                        begin=datetime.datetime(2013,1,1),
                        end=datetime.datetime(2016,1,1),
                        max_results=10,
                        max_cloud_cover=40)

# Extract Landsat scene ids for each hit from the metadata
scene_list = [x['displayId'] for x in scene_list]
```

### 3.2 Place a processing order to Espa

The scene list can be used to send a processing order to Espa via the Espa API.

```
from lsru import Espa
from pprint import print

# Instantiate Espa class
espa = Espa()

# Place order (full scenes, no reprojection, sr and pixel_qa)
order = espa.order(scene_list=scene_list, products=['sr', 'pixel_qa'])
print(order.orderid)

# espa-loic.dutrieux@wur.nl-10212018-102816-245'
```

### 3.3 Check current orders status

```
for order in espa.orders:
    # Orders have their own class with attributes and methods
    print('%s: %s' % (order.orderid, order.status))

# espa-loic.dutrieux@wur.nl-10222018-062836-330: ordered
# espa-loic.dutrieux@wur.nl-10212018-174321-508: complete
# espa-loic.dutrieux@wur.nl-10212018-174430-792: complete
# espa-loic.dutrieux@wur.nl-10212018-102816-245: complete
# espa-loic.dutrieux@wur.nl-10182018-100137-786: complete
```

### 3.4 Download completed orders

When Espa finishes pre-processing an order, its status changes to `complete`, we can then download the processed scenes.

```
for order in espa.orders:
    if order.is_complete:
        order.download_all_complete('/media/landsat/download/dir')
```

---

## Order data using a polygon

---

The following example details the steps to place a pre-processing order of scenes intersecting with a polygon. Such task requires a little bit of boilerplate code since the Usgs API requires an extent and not a polygon, but thanks to `shapely` and some additional utilities provided by `lsru` it can be done with a few lines of code. The steps are roughly to:

- Read the feature (e.g. using `fiona` or directly from a geojson file with `json`)
- Compute the bounding box of the feature
- Query scenes intersecting with the bounding box to the Usgs API
- Filter out scenes that do not intersect with the geometry
- Place the pre-processing order to `Espa`

To run this script, we'll need `shapely`, and some utils to manipulate geojson geometries provided by `lsru`. You may need `fiona` as well in case you need to read a feature from a geospatial vector file (shapefile, geopackage, etc)

```
from datetime import datetime

from shapely.geometry import shape

from lsru import Usgs, Espa, Order
from lsru.utils import bounds, geom_from_metadata
```

For the present example, we'll use the following feature, which roughly corresponds to the contours of the state of Baja California Sur in Mexico.

```
feature = {'geometry': {'coordinates': [[[-114.192, 27.975],
                                         [-114.456, 27.8],
                                         [-115.093, 27.878],
                                         [-114.543, 27.313],
                                         [-113.928, 26.922],
                                         [-113.423, 26.844],
                                         [-112.72, 26.412],
                                         [-112.148, 25.741],
```

(continues on next page)

(continued from previous page)

```

        [-112.324, 24.827],
        [-111.687, 24.467],
        [-110.984, 24.147],
        [-110.72, 23.725],
        [-110.281, 23.504],
        [-110.061, 22.877],
        [-109.446, 23.06],
        [-109.424, 23.423],
        [-110.259, 24.347],
        [-110.347, 24.187],
        [-110.61, 24.327],
        [-110.588, 24.707],
        [-111.445, 26.215],
        [-112.654, 27.684],
        [-112.961, 28.053],
        [-114.192, 27.975]]],

    'type': 'Polygon'},
    'properties': {},
    'type': 'Feature'}

# Generate the bounding box of the geometry used for the spatial query
bbox = bounds(feature['geometry'])

# Instantiate Usgs class and login
usgs = Usgs()
usgs.login()
collection = usgs.get_collection_name(8)

# Query for records of Landsat 8 intersecting with the spatio-temporal window
meta_list = usgs.search(collection=collection, bbox=bbox,
                        begin=datetime(2018, 1, 1), end=datetime(2018, 2, 28),
                        max_cloud_cover=30)
print(len(meta_list))

# Printed:
# 52

```

Because we queried the data using the extent, it is highly probable that some scenes do not intersect with the initial geometry but only with its extent. We can therefore filter the list of scene metadata by testing for intersection between the scene bounds and the geometry. This is done using shapely

```

region_geom = shape(feature['geometry'])
meta_list = [x for x in meta_list if
             shape(geom_from_metadata(x)).intersects(region_geom)]

print(len(meta_list))

# printed:
# 27

```

The amount of element has reduced by half compared to the total API hits and we are now sure to have retained only scenes that actually intersect with the initial geometry.

We can now proceed to preparing the scene list for placing the order to espa

```

scene_list = [x['displayId'] for x in meta_list]
espa = Espa()

```

(continues on next page)

(continued from previous page)

```
order = espa.order(scene_list,  
                   products=['pixel_qa', 'sr_ndmi'])
```

We can then track the status of the order and eventually download it once processing is completed

```
print(order.status)  
  
# printed:  
# ordered
```





## CHAPTER 5

---

### Indices and tables

---

- `genindex`
- `modindex`
- `search`



---

## Symbols

`__init__()` (*lsru.Espa* method), 8  
`__init__()` (*lsru.Order* method), 10  
`__init__()` (*lsru.Usgs* method), 6

## B

`bounds()` (*in module lsru.utils*), 11

## C

`cancel()` (*lsru.Order* method), 11

## D

`download_all_complete()` (*lsru.Order* method),  
10

## E

`endpoint` (*lsru.Usgs* attribute), 6  
`Espa` (*class in lsru*), 7

## G

`geom_from_metadata()` (*in module lsru.utils*), 12  
`get_available_products()` (*lsru.Espa* method),  
9  
`get_collection_name()` (*lsru.Usgs* static  
method), 7

## H

`host` (*lsru.Espa* attribute), 8

## I

`is_valid()` (*in module lsru.utils*), 12

## K

`key` (*lsru.Usgs* attribute), 6  
`key_dt` (*lsru.Usgs* attribute), 6

## L

`login()` (*lsru.Usgs* method), 6

## O

`Order` (*class in lsru*), 10  
`order()` (*lsru.Espa* method), 8  
`orderid` (*lsru.Order* attribute), 10

## P

`PASSWORD` (*lsru.Espa* attribute), 8  
`PASSWORD` (*lsru.Usgs* attribute), 6

## S

`search()` (*lsru.Usgs* method), 6

## U

`url_retrieve()` (*in module lsru.utils*), 13  
`url_retrieve_and_unpack()` (*in module*  
*lsru.utils*), 13  
`USER` (*lsru.Espa* attribute), 7  
`USER` (*lsru.Usgs* attribute), 5  
`Usgs` (*class in lsru*), 5