
WASDI documentation center

May 13, 2024

GETTING STARTED

1	Getting Started with WASDI	3
1.1	Wasdi Web Platform access and basic usage	3
1.2	Wasdi Libraries Concepts	10
2	WASDI User Manual	17
2.1	Signing Up and Signing In	17
2.2	Workspace Management and Use	21
2.3	Searching for Products	43
2.4	Managing Subscriptions and Organizations	47
2.5	Other	53
3	WASDI Marketplace	59
3.1	Wasdi App Store	59
3.2	eDrift Tutorial	67
3.3	Wheat Locator	84
4	Add your App to WASDI	87
4.1	Python Tutorial	87
4.2	Jupyter Notebook Tutorial	127
4.3	Python Landsat Tutorial	133
4.4	Search and Import EO Images	148
4.5	Configuration tutorial	168
4.6	Working with Workspaces and Products	172
4.7	Synchronous and Asynchronous WASDI programming	175
4.8	C# Tutorial	181
4.9	Site Map	206
4.10	How to create a User Interface (UI)	209
4.11	Javascript Web Tutorial	240
4.12	Javascript Angular Tutorial	250
5	Reference center	257
5.1	C# WasdiLib	257
5.2	Java WasdiLib	290
5.3	Matlab WasdiLib	313
5.4	Octave WasdiLib	337
5.5	Python WasdiLib	341
5.6	Javascript WasdiLib	364
5.7	Create a config.json file	368
5.8	Python Application Skeleton	369
5.9	Read Parameters	370

5.10	Search Sentinel-1 Images	371
5.11	Search Sentinel-2 Images	374
5.12	Search Sentinel-3 Images	376
5.13	Search Sentinel-5p products	378
5.14	Search Copernicus Marine products	379
5.15	Search ECOSTRESS products	382
5.16	Search ERA5 products	384
5.17	Import Images after a Search	386
5.18	Import And Pre-Process	388
5.19	Run Snap Workflow	389
5.20	Run Another WASDI Application	391
5.21	Save Payload	393
5.22	Get list of S2 tiles in an area of interest	394
5.23	Use Library as client	395
5.24	Change HTTP request timeouts	396
5.25	Add a Data Provider to WASDI	397
5.26	Add Application User Interface controls	406
6	Terms and Conditions	415
6.1	EULA	415
6.2	Privacy policy	422
	Python Module Index	429
	Index	431

WASDI implements a unique, simple and intuitive interface to foster the exploitation of the asset concerning EO data and satellite products, for satisfying requirements of users' communities and, in particular:

- experts/researchers in the field of Earth Sciences
- managers of services and public administrations (i.e. civil protection decision makers)
- private companies (i.e. insurance, agriculture)

WASDI allows researchers to gather satellite data, in particular the Sentinel ones, display them on-line, run algorithms, displaying and evaluating the results, and allows to share these projects among different users.

The results of the calculations will then be available for download, allowing local further processing, or published directly through the Web.

GETTING STARTED WITH WASDI

WASDI web platform is the best starting point for your journey on Earth Observations (EO) resources !

This basic tutorial will help to acquire the main concepts and use WASDI for your EO research.

If you're acquired the basic concepts of WASDI and you're interested in how processors can be launched, take a look at this tutorial. This will highlights Synchronous and Asynchronous WASDI programming.

<https://youtu.be/6LHIwdyh45U>

1.1 Wasdi Web Platform access and basic usage

The Home page of WASDI is reachable at the address:

<https://www.wasdi.net/>



1.1.1 WASDI Login

To create a WASDI account, just click on the **New User? Register here!** link

To register the user has to input:

- A valid eMail Address (that will be used as UserId)
- A password
- A Name and Surname

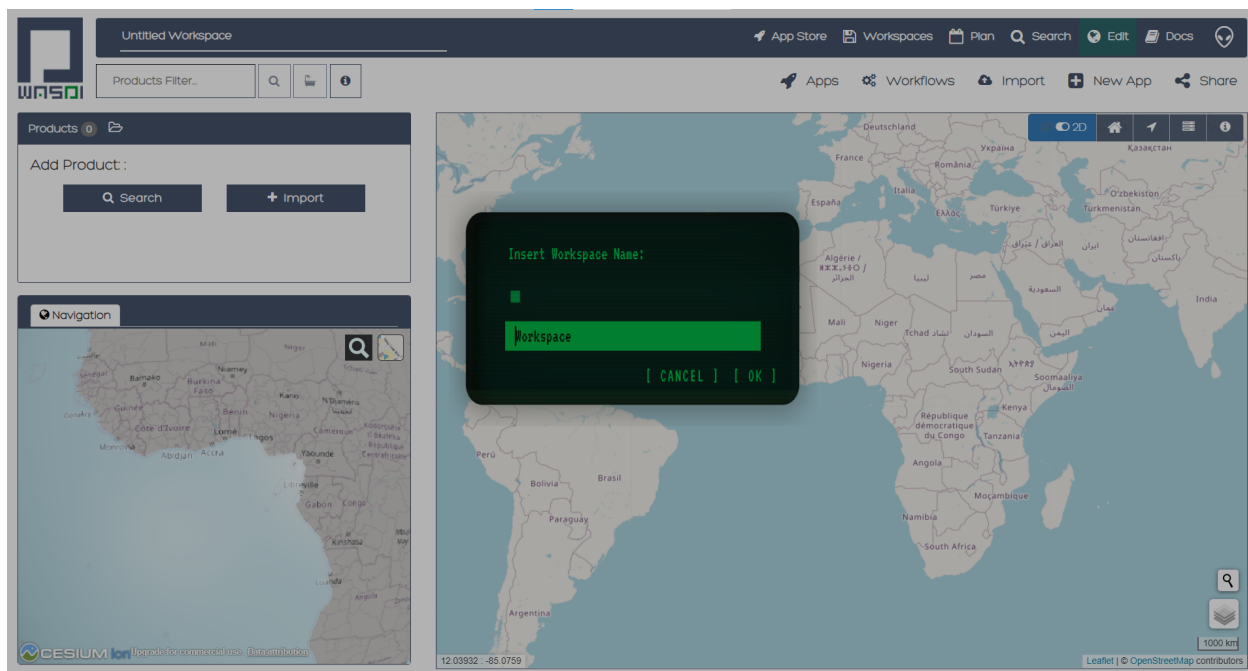
The user will receive a confirmation e-mail and then, after the confirmation, will be enabled to use WASDI.

1.1.2 Workspace Management

Each user in WASDI can work in one or more Workspaces. A Workspace is a set of files (original EO data or elaborated by some processor) that are grouped in the same “project”.

The workspace concept is the same of several other development tools or languages: the same concept can be either be named workspace or project and any other name designed to identify to a specific set of files.

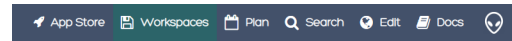
Once logged for the first time, the user is asked to create a new workspace.



To work with the Editor a Workspace is needed: just click on the text suggestion or on the New Workspace Button on the top right of the screen.

1.1.3 Wasdi Sections

The WASDI Sections are listed in the top blue bar:

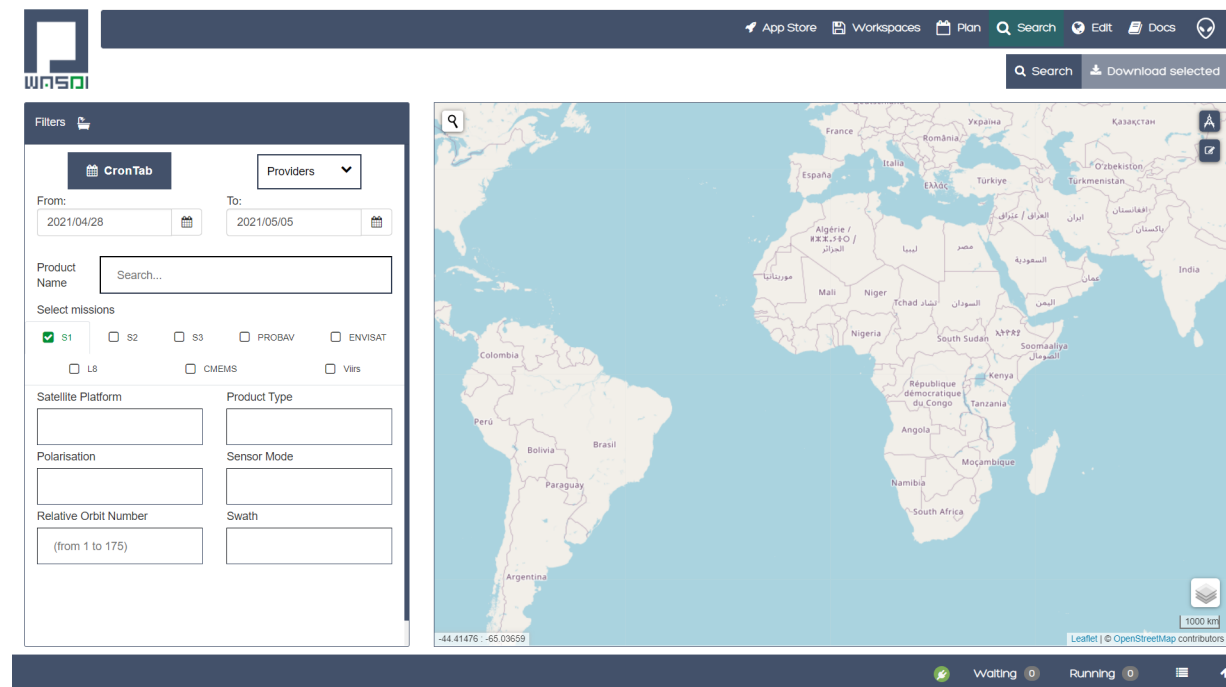


- **App Store:** Space Marketplace: explore and run available applications;
- **Workspaces:** workspace management: create, open or delete your workspaces;
- **Plan:** explore new acquisition plan for different Satellite platforms;
- **Search:** search for Satellite Images from all the supported Data Providers;
- **Edit:** edit data in a workspace;
- **User:** link to the documentation;
- **User:** user info and properties

App Store, Plan, and User are out of the scope of this Tutorial.

Search

The search section has many features; in this guide we will make a basic introduction just to let the user start working with EO Images.



The user can set the filters:

From: To:

Product Name:

Select missions

☒ S1 ☐ S2 ☐ S3 ☐ PROBAV ☐ ENVISAT

☐ L8 ☐ CMEMS ☐ VIIRS

Satellite Platform:

Product Type:

Polarisation:

Sensor Mode:

Relative Orbit Number:

Swath:

To enable one mission specific filter, first select the tab and the Checkbox of the mission. If no checkbox is selected, the system will search all the available missions, otherwise, selected it will search only for the selected ones.

WASDI has a Multi Provider search system: this means that the same query is sent to many data providers. The user can switch providers on/off:

← □ ✓

ONDA 2 SENTINEL 2 LSA 4 SOBLOO 0 EODC 0

CREODIAS 4 PROBAV 0 VIIRS 0

Products per page: 10 ▾

< Page: 1 of 1 >

May 1, 2021 6:01:47 PM

✓ + 🔍 i

IW SAR-C

1.65 GB

Name: **S1A_IW_GRDH_1SDV_20210501T052717_20210501T052717_000000000000_000000000000_000000000000**

Polarisation: **IW**

Relative Orbit: **168**

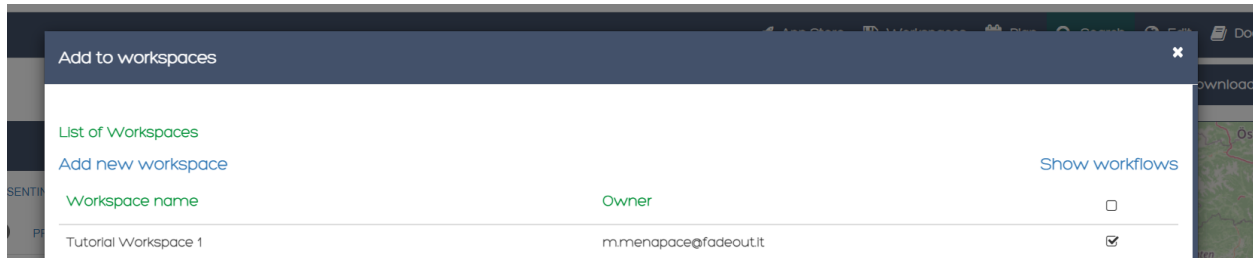
Provider: **ONDA**

Platform name: **Sentinel-1**

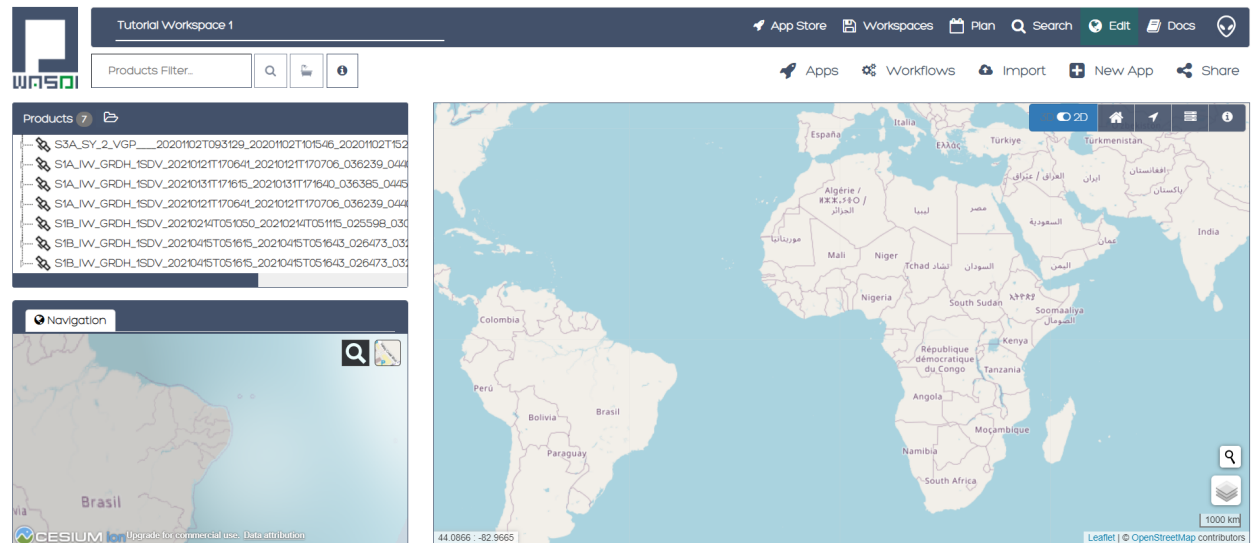
Sensor Operational mode: **IW**

The SERCO ONDA DIAS is the **priority one** Provider because data is stored in the Cloud where WASDI is installed so this is, usually, the fastest provider available.

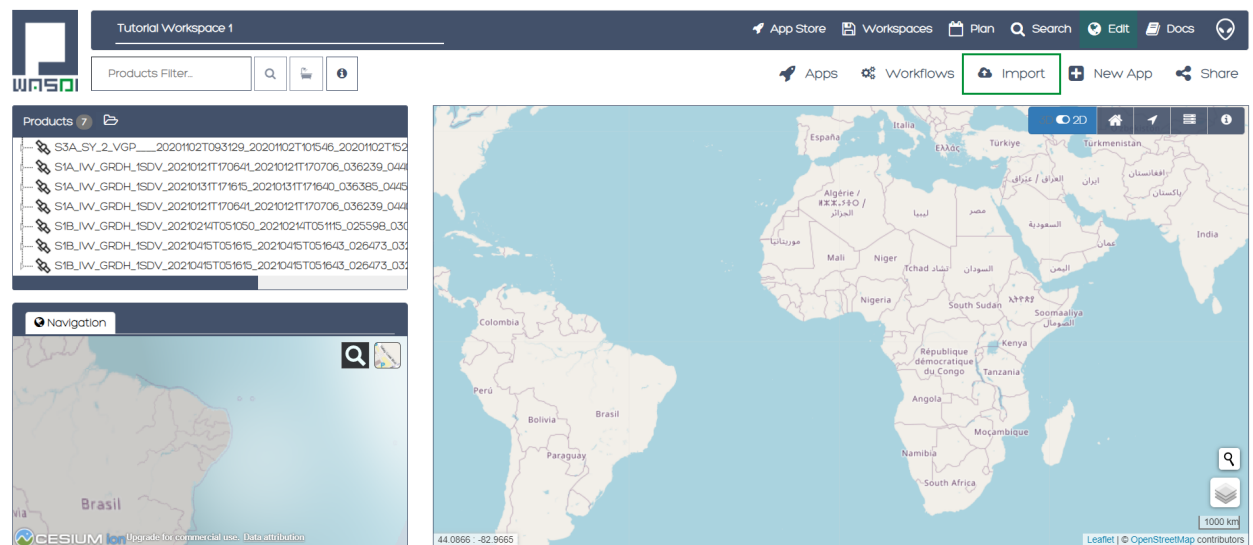
To start a query click on the SEARCH button on the top right of the screen:



The results are shown in different tabs for different providers:

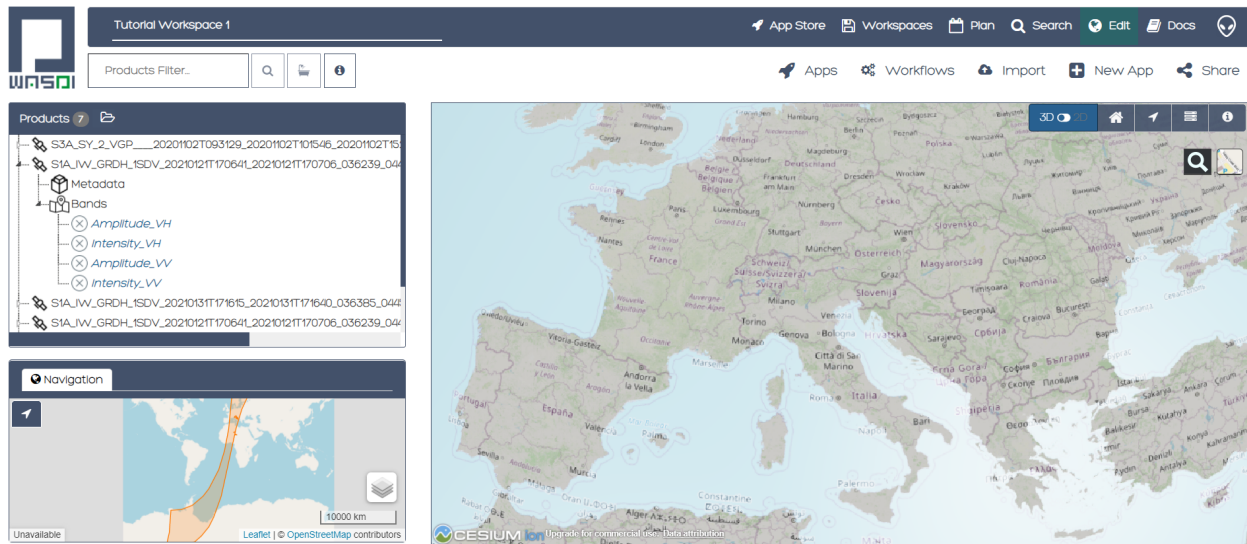


To add a image to WASDI, click on the “+” icon near to the name of the image. The system will ask for the workspace to use and then will start the import of the image.



Editor

Editor is where the user can interact with the EO Images.



On the left the tree of the products in the workspace is shown. Each product has its own metadata and band subfolder.

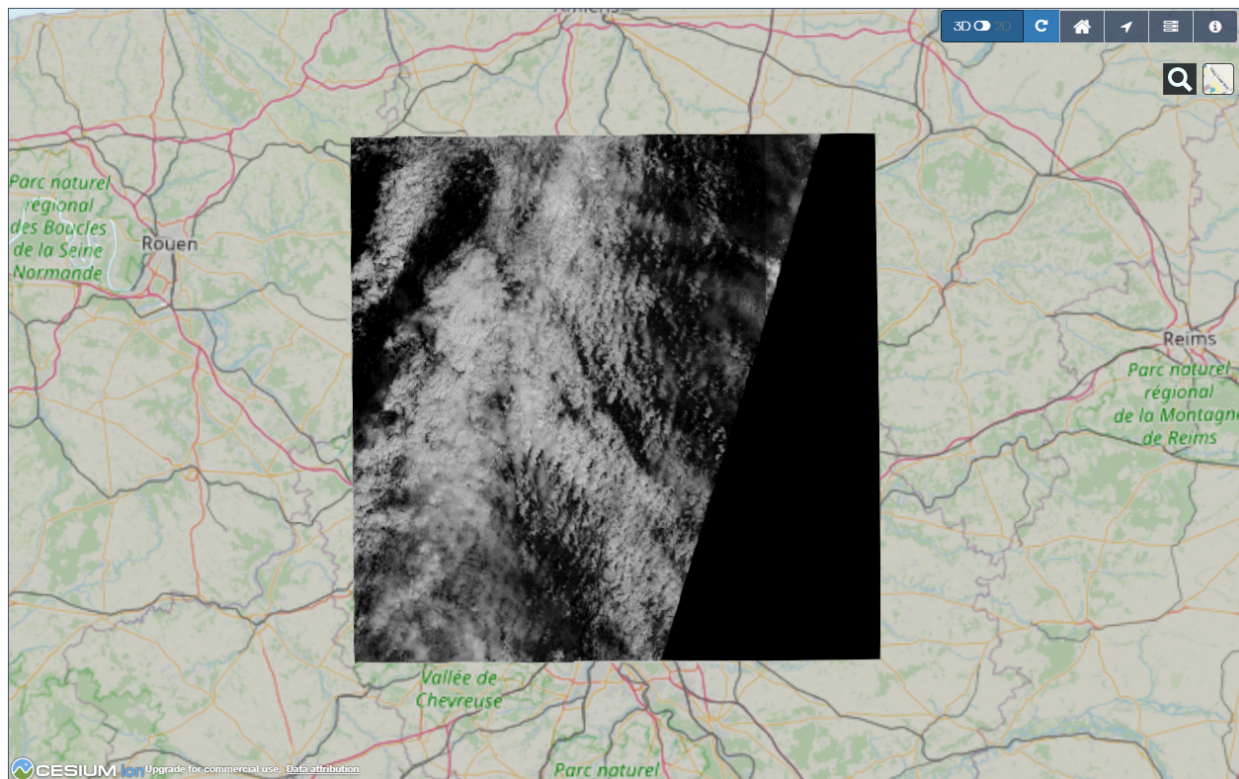


The user can browse the bands of the image. Just click on a band to see the band image.



WASDI will publish on the fly in OGC-WMS standard the selected bands.

Once published the band will be shown on the map:



From the editor is possible to run SNAP Workflows and all the available user-supplied processors.

1.2 Wasdi Libraries Concepts

This tutorial has to goal to introduce the main WASDI App development concepts, that can be used in all the supported programming language to get the best from the WASDI libs.

1.2.1 Introduction

To create a WASDI account, just click on the **New User? Register here!** link

The main goal of the WASDI libraries is to let you develop in your onw enviroment. Do you prefer Windows, Apple or Linux? Do you use Python, IDL, Java,C#, Javascprit or Matlab? Do you use an integrated IDE, or you code using VIM? For WASDI is the same.

The world is full of different possibilities and environments and we do not want to introduce a new one. We think that every one should concentrate to obtain the result not to learn a new tool. So WASDI libraries are designed to be intalled and used in every environment.

In `python` you can use:

```
pip install wasdi
```

In `IDL` or `Matlab/Octave`, just donwload the lib files and save on your computer.

For `Java`, just download and link our jar to your project.

For `C#`, install our nuget package.

For `JavaScript`, install our Node package, or link our cdn lib `JavaScript Library`.

All the `WASDI` libraries are `Open Source`.

1.2.2 Main Goals

The main goals on the libraries are:

- **Authenticate** in WASDI: this is based on a config file that you must create on your computer, and keep for you on your computer: here you will declare your WASDI credentials that will be used by the lib to interoperate with the system.
- Abstract the **access to the files**: this is the real main concept of WASDI, the only thing we require to our devel-opers: **ask the paths to WASDI!**

The file access abstraction is very simple to use: make your code in a way that, when must access one file, it can ask the path to wasdi:

```
# Python Code
fileFullPath = wasdi.getPath("S2B_MSIL1C_20210124T141049_N0209_R110_T20KNV_
↪20210124T160035.zip")
```

```
; IDL Code
fileFullPath = WASDIGETPATH("S2B_MSIL1C_20210124T141049_N0209_R110_T20KNV_
↪20210124T160035.zip")
```

```
//Java: we created an object WasdiLib oWasdiLib = new WasdiLib();
String sFileFullPath = oWasdiLib.getPath("S2B_MSIL1C_20210124T141049_N0209_R110_T20KNV_
↪20210124T160035.zip");
```



```
% We obtained a lib object calling Wasdi = startWasdi(config_path)
fileFullPath = wGetPath("S2B_MSIL1C_20210124T141049_N0209_R110_T20KNV_20210124T160035.zip
↪")
```

All the lib functions, take in input just the file names and not the paths, and you should work in the same way.

Thank to this little rule, WASDI will be able to optimize the access to files: when you work locally, WASDI will keep the file to the cloud until is not really necessary to open it locally: in that moment, if not present yet, WASDI will download automatically the file for you. The same when you create your result and wants to add it to the cloud: WASDI will keep the local file until is possible, and will automatically upload it if and when requested.

Files are exchanged only once, if and when needed: all is completely transparent to you, in all the supported languages.

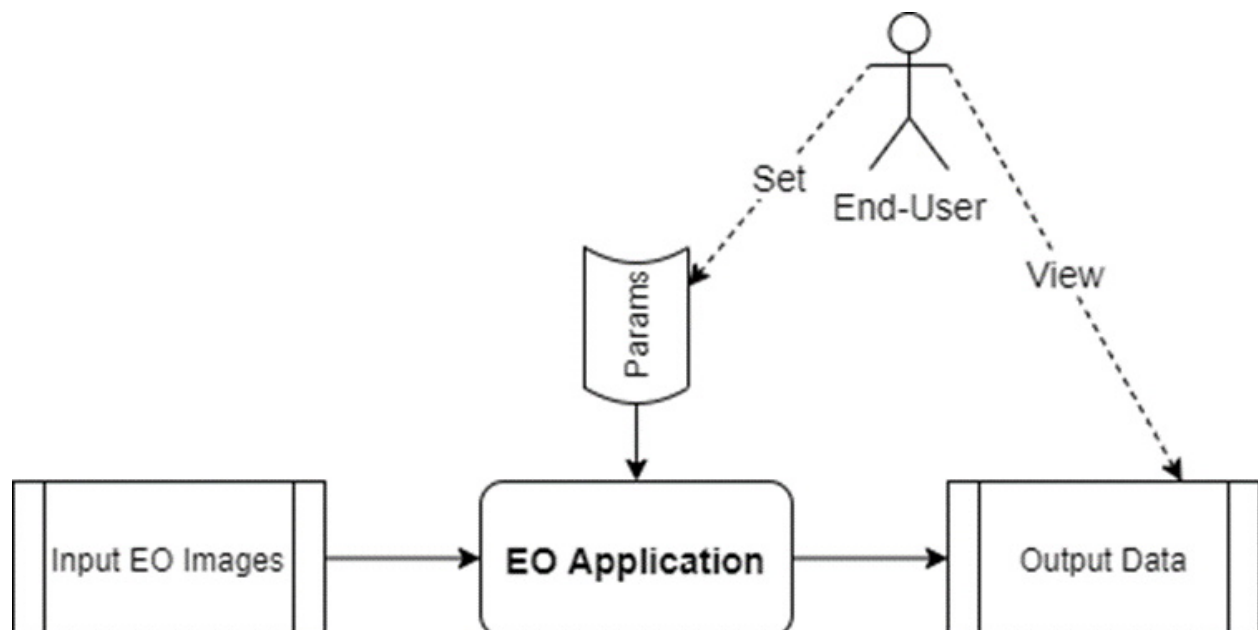
1.2.3 Main Entities

To work with WASDI there are only few basic entities to consider:

- **Product:** in general, any file in WASDI. Products are satellite images taken from a data provider, files imported by you, files generated by an application or a workflow.
- **Workspace:** the workspace is a container of products. Is where you can download images, run applications, run workflows. Each workspace has an owner and can be shared with users.
- **Workflow:** [SNAP](#) graphs. SNAP is the ESA Open Source Tool to handle Sentinels and Other missions images. A SNAP graph can be uploaded in WASDI with a drag and drop and executed on the cloud
- **App:** a WASDI app is a processor developed by some WASDI user, in some language, that can be used. An app can be your own code for instance. Each WASDI Application can be private, can be shared with selected users or can be public. Each Application can be shown in the marketplace or not. Can be free or not. The developer can decide any detail of his own application.

1.2.4 Applications Model

The typical WASDI Application can be represented by this schema:



EO Application is the real processor deployed with WASDI. When the End-User starts an application, he must set the input parameters. These parameters can be a date interval, a bounding box, a sensitivity index and any other specific option.

Usually EO Applications use these parameters to fetch EO Images that has to be elaborated and create the output added-value data. The End-User, once the processor is finished, can view the generated output usually in a web GIS Environment.

All the WASDI EO-Applications accept a key-value dictionary as Parameters.

In python and C#, it is a JSON File.

In Java, Matlab/Octave, IDL, it is a standard properties file in the format:

KEY=VALUE.

Parameters are your own inputs. Since WASDI is desinged to make your application running in the cloud, we ask you to make “pure code” that does not care how to get inputs from the user, but just USE the inputs given by the user. This is done in the parameters file: there you decide the inputs you need and there you can put and change your inputs. WASDI will let you read your inputs using a simple line of code:

```
#python
myParam = wasdi.getParamter("StartDate", new Date())
```

```
;IDL
myParam = WASDIGETPARAMETER("StartDate")
```

```
//Java: we created an object WasdiLib oWasdiLib = new WasdiLib();
String sMyParam = oWasdiLib.getParameter("StartDate");
```

```
% We obtained a lib object calling Wasdi = startWasdi(config_path)
myParam = wGetParameter("StartDate")
```

1.2.5 Configuration

All the libraries uses a config file to be initialized. For Java, Matlab/Octave, IDL config files are standard properties file in the format:

KEY=VALUE

For python and C# it is a JSON file.

The basic configuration, that can be used in almost all cases, is:

```
USER=your.email@domain.sample
PASSWORD=yourpassword
WORKSPACE=NameOfYourWorkspace
PARAMETERSFILEPATH=./params.txt
```

```
{
    "USER": "your.email@domain.sample",
    "PASSWORD": "yourpassword",
    "WORKSPACE": "NameOfYourWorkspace",
    "PARAMETERSFILEPATH": "./params.json"
}
```

Basic Parameters are:

- **User** is your valid WASDI UserId, ie the mail you used to register.
- **Password** is your valid WASDI password.
- **Workspace**, is the workspace where you want to run the code you are writing.
- **ParametersFilePath** File Path is the path where you have the parameters file for the code you are running.

Advanced configuration can be controlled adding these entries to the config file:

- **BASEPATH**=c:/local/path/ - This is the local base path used by wasdi to read and save the data you are using in your application.
- **DOWNLOADACTIVE**=1 - If 1, WASDI will automatically download the data you need in your code when requested
- **UPLOADACTIVE**=1 - If 1, WASDI will automatically upload your data to the cloud when requested.
- **BASEURL**=https://www.wasdi.net/wasdiwebserver/rest - base url of the WASDI APIs
- **WORKSPACEID**=364c24ff-4891-4d0a-83bd-2772d292f918 - Id of the workspace, can be used as an alternative to the **WORKSPACE** name option
- **VERBOSE**=1 1 to make local console verbose log of the lib, 0 to deactivate
- **REQUESTSTIMEOUT**=5 - seconds of timeout for the lib http calls

1.2.6 Local File System

As it been stated before, libraries make an automatic optimized download and upload of files from your local PC to the cloud when and only when is needed. This functionality is smart and is activated only when you work on your computer; when you will deploy your app to the cloud WASDI will directly access the files.

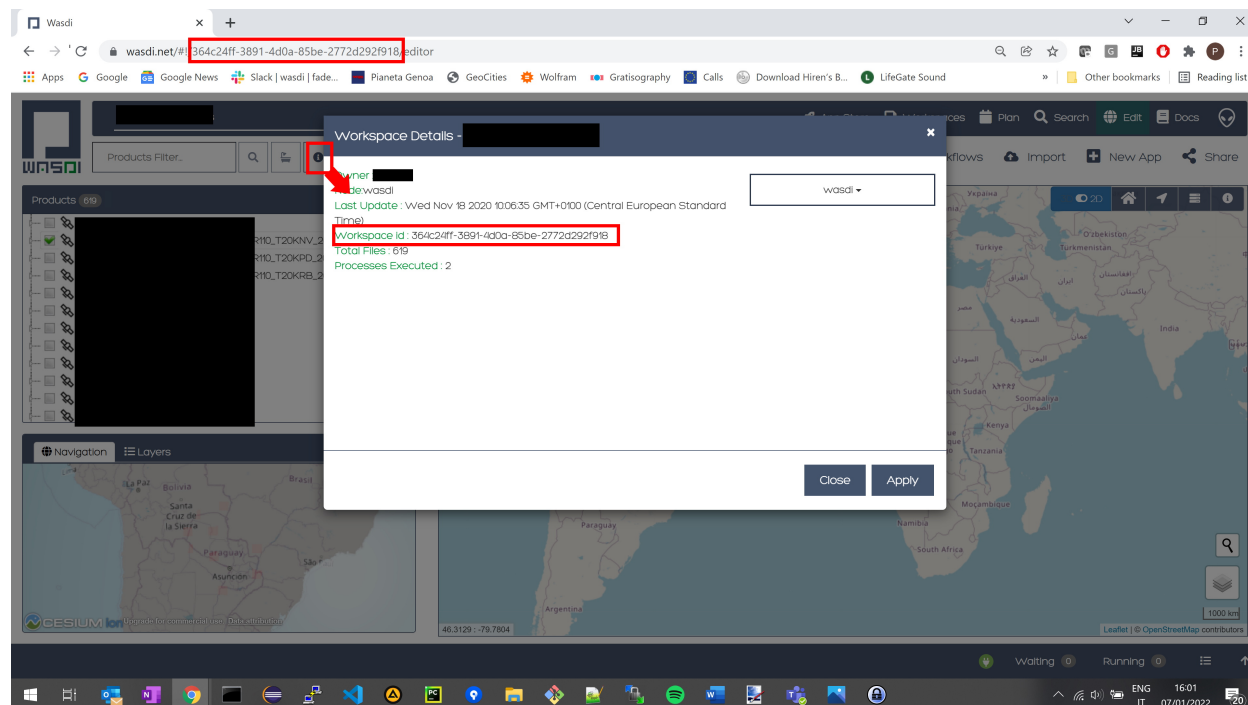
This means that you will have your files on your computer, and this can be useful to double check your results, open the files with other tools like ENVI or QGIS, copy the files to other locations and whatever you may need.

By default, for all the languages, WASDI use as base folder the home folder of your computer user and adds a .wasdi folder. Can be:

- Linux: /home/[your user]/.wasdi
- Windows: C:\Users\[your user]\.wasdi

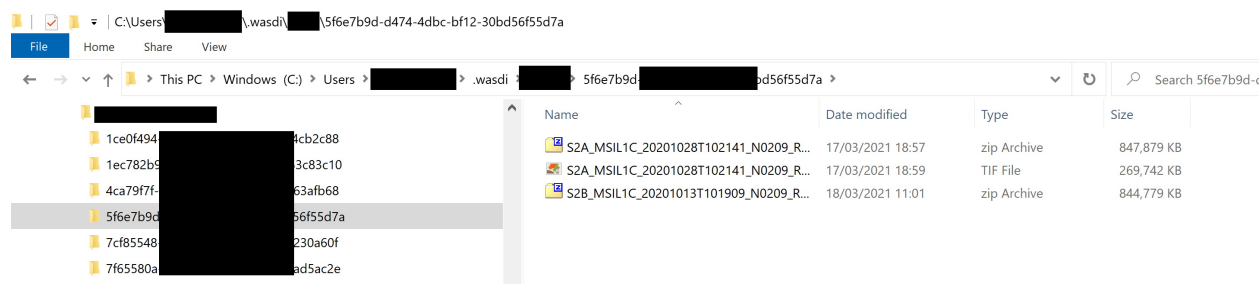
In the .wasdi folder WASDI will create a subfolder for each user and for each workspace. Is very important to remember: this will be done only if and when is needed!! if there is no need to access the file locally, all we be delegated to the cloud and no folder will be created. If a file is accessed locally, the folder will be created and the file downloaded.

Each workspace folder will be named as the workspaceId. The Workspace Id is a guid. You can find the workspace Id from the web application in two ways:



It is in the address bar, when you are in the Editor section. You can click on the info button and read from the property window the Workspace Id.

The folder structure will be something similar:



- Linux: /home/[your user]/.wasdi/[WASDI_User]/[WorkspaceId]/
- Windows: C:\Users\[your user]\.wasdi\[WASDI_User]\[WorkspaceId]\

These are your folders, you can do what you want of that folders. Again: only and when needed, WASDI will search there for the needed files and, if not available, will download it.

1.2.7 Basic Functionalities

The basic functionalities are:

- Access users' workspaces and files
- Search EO Images
- Import EO Images in the workspace
- Execute SNAP Workflows
- Execute other WASDI Applications

- Execute basic GIS Operations (mosaic, multisubset)
- Run Sen2Core

1.2.8 Advanced Functionalities

The advanced functionalities are:

- Send log directly to the web user interface
- Update the progress of the processing
- Save a payload associated to each run of the app
- Search and retrieve the execution of other processors and the relative payloads

WASDI USER MANUAL

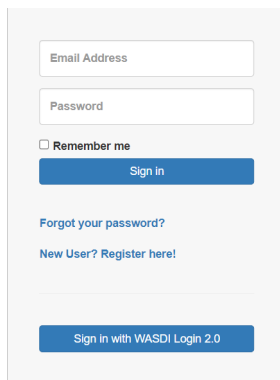
WASDI has created a comprehensive user manual to explain and simplify all operations in WASDI. If you require explanation for any concepts in WASDI, please see the corresponding section in the manual. A good starting point to search for and executing applications is the tutorial on the Space Marketplace

2.1 Signing Up and Signing In

To create an account on <https://www.wasdi.net/> , navigate to the homepage and click “Sign Up”.

2.1.1 WASDI Login

To create a WASDI account, either click **New User? Register here!** link.

A screenshot of the WASDI login form. It features two input fields: 'Email Address' and 'Password'. Below the password field is a checkbox labeled 'Remember me'. A blue 'Sign in' button is positioned below the checkbox. Underneath the button are two links: 'Forgot your password?' and 'New User? Register here!'. At the bottom of the form is a blue button labeled 'Sign in with WASDI Login 2.0'.

The image shows a web registration form for WASDI. The header features the WASDI logo in white and green on a dark blue background. The form itself is white and centered, with the title 'Register' at the top. It contains five input fields: 'First name' with the value 'Betty', 'Last name' with 'Spurgeon', 'Email' with 'betty.spurgeon713@gmail.com', 'Password' with masked characters, and 'Confirm password' also with masked characters. Below the fields is a green link '« Back to Login'. At the bottom is a large green button labeled 'Register'.

WASDI

Register

First name

Last name

Email

Password

Confirm password

[« Back to Login](#)

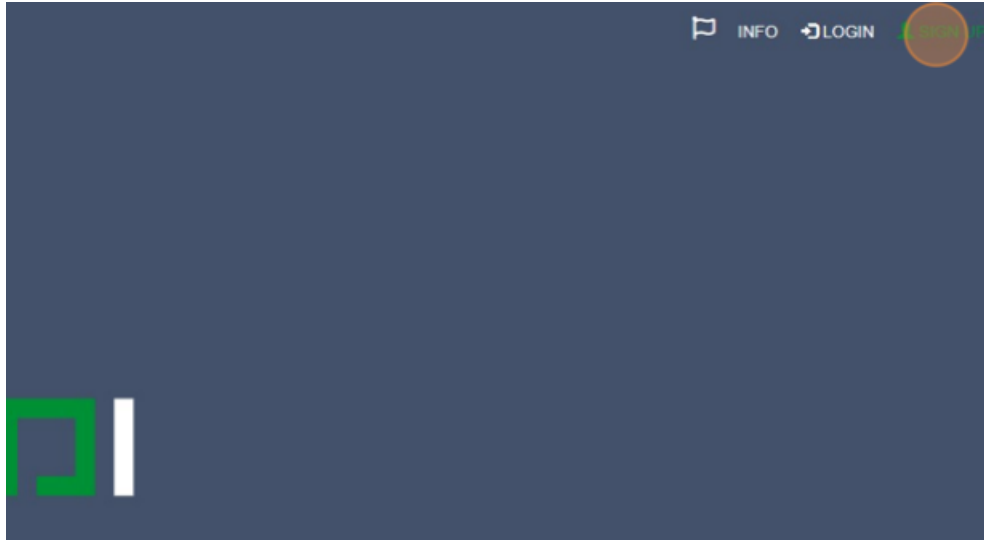
Register

To register, the user must input:

- A valid email address (this will be your User Id in WASDI).
- A password
- A first name and last name

2.1.2 Keycloak Login

You are also able to create a new account via WASDI's authentication partner Keycloak. To create an account with Keycloak, either click **Sign Up** in the top right-hand corner or click **Login**, then **Sign in with WASDI Login 2.0**, and finally click **New User? Register**.



The image shows a registration form for WASDI. The header features the WASDI logo in a stylized, blocky font, with the 'S' and 'D' in green and the 'W', 'A', 'I', and 'I' in white. Below the logo, the word "Register" is centered. The form contains five input fields: "First name" with the value "Betty", "Last name" with the value "Spurgeon", "Email" with the value "betty.spurgeon713@gmail.com", "Password" with masked characters "*****", and "Confirm password" with masked characters "*****". A green button labeled "Register" is at the bottom. A link "« Back to Login" is positioned above the button.

Register

First name

Last name

Email

Password

Confirm password

[« Back to Login](#)

Register

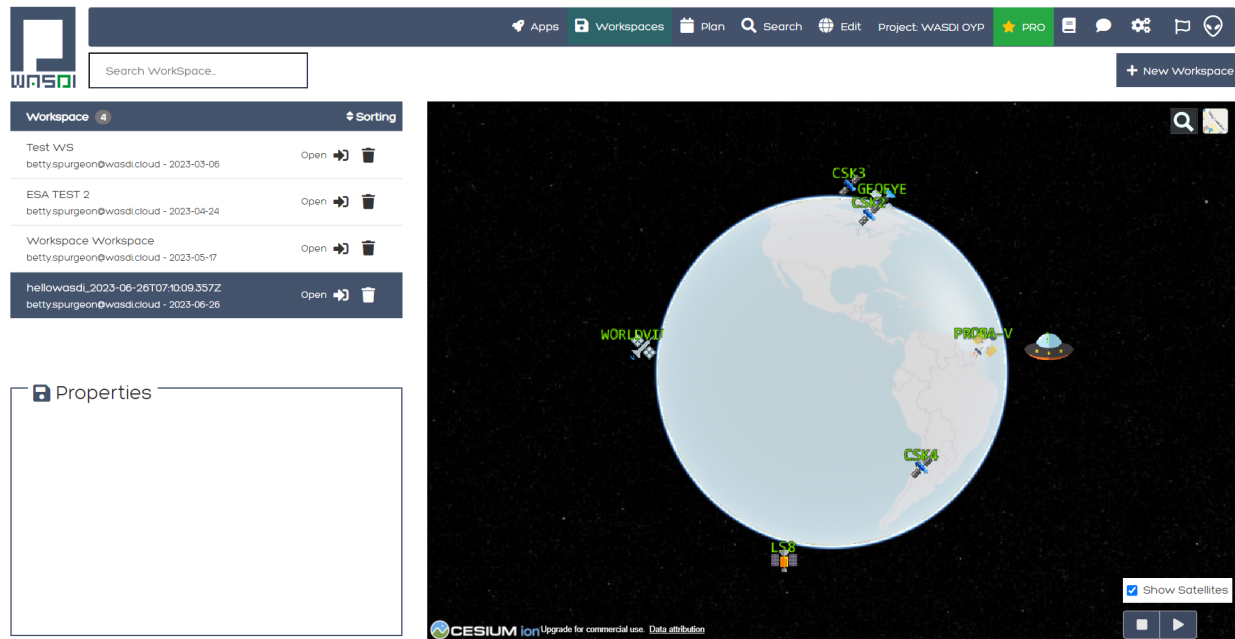
After you've successfully created an account via either method, you will receive a confirmation email. Once you have followed any prompts in this email, you will be able to use WASDI!

2.2 Workspace Management and Use

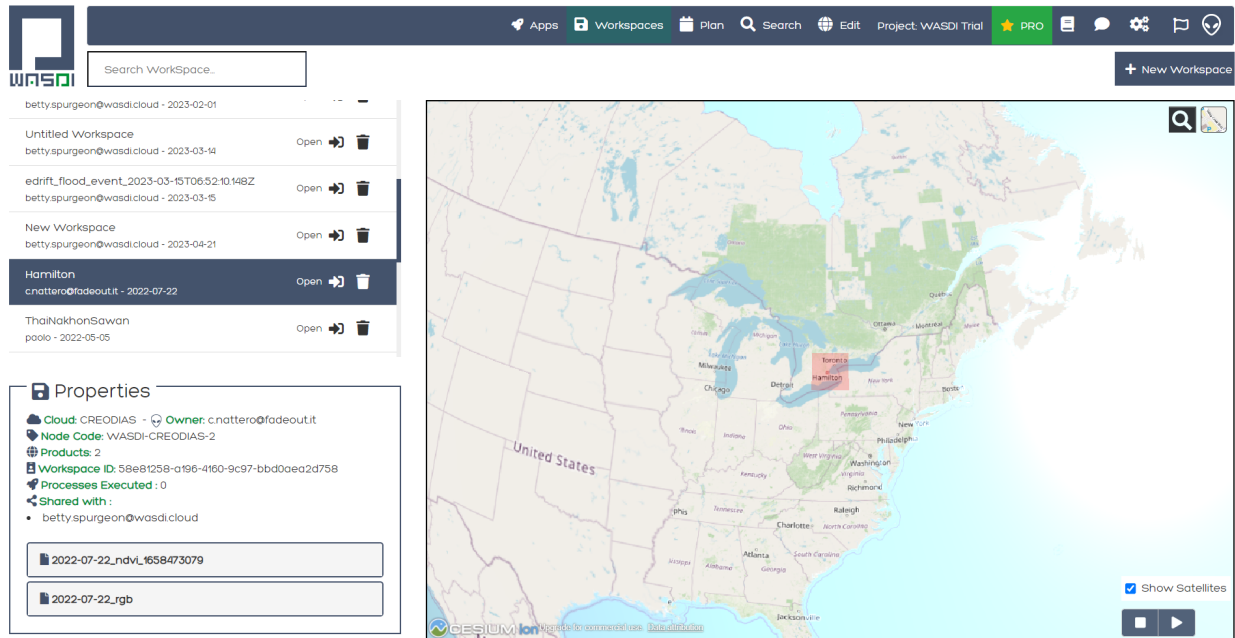
2.2.1 The Workspaces Tab

Each WASDI User can work in one or more Workspace. A Workspace is a set of files (original EO data or elaborated by some processor) that are grouped in the same “project”.

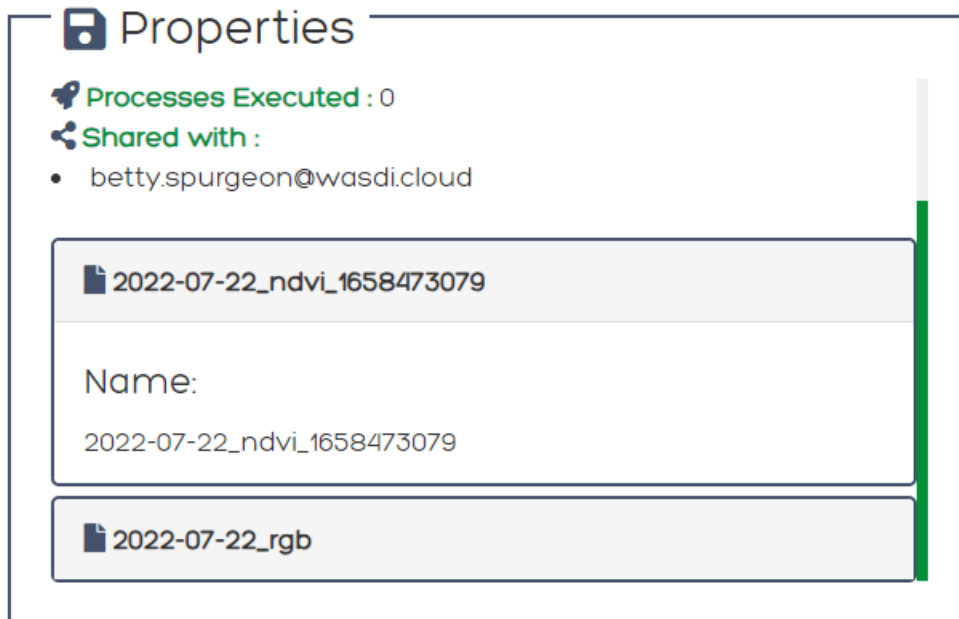
The Workspaces page is where all of your workspaces are displayed. On this page you are able to manage (e.g., create and delete) and view the properties of each workspace.



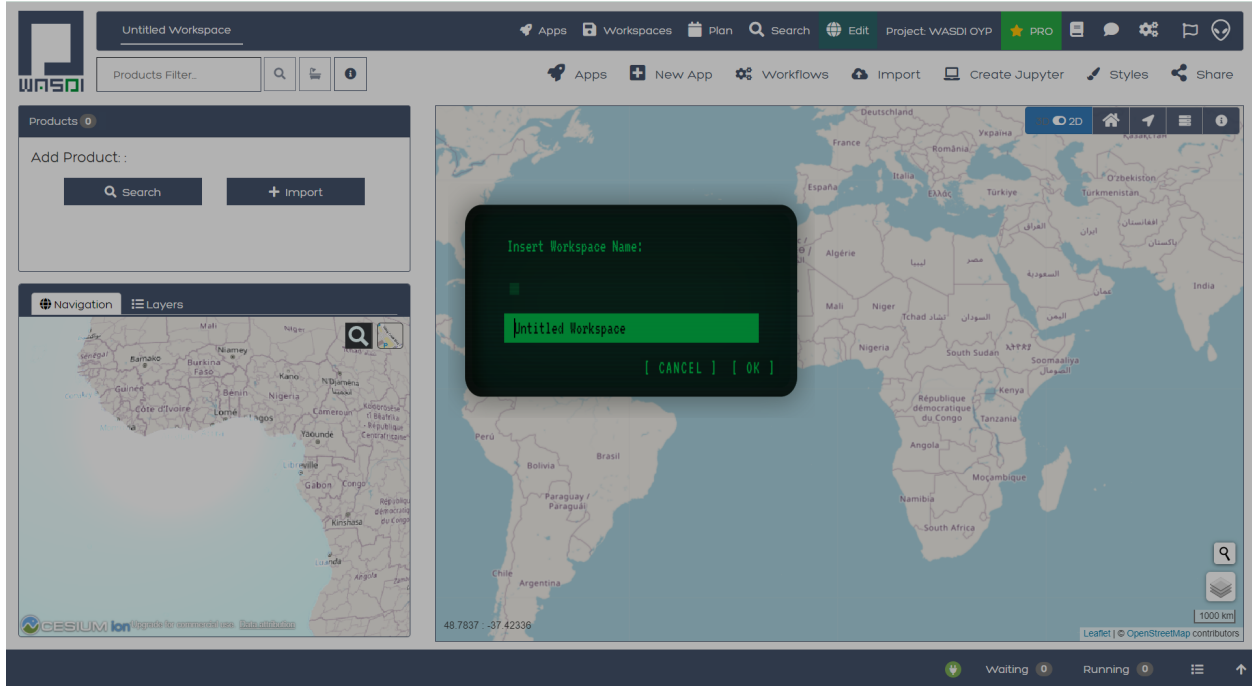
When you select a workspace, that workspace’s properties will be displayed and the globe will navigate to the associated geographic location.



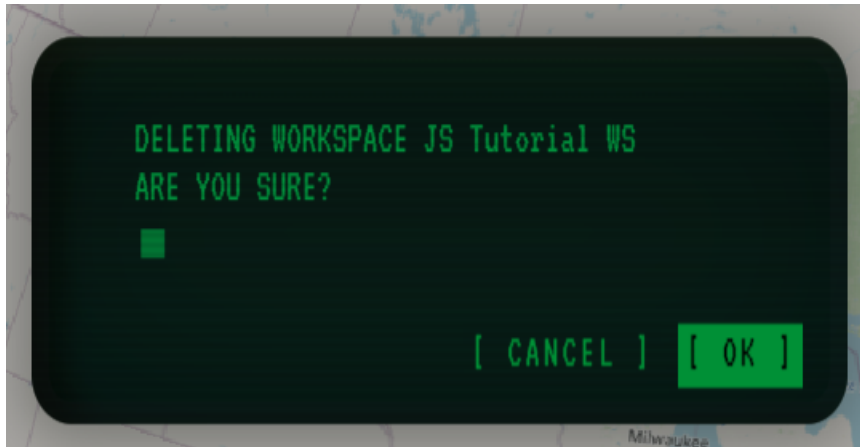
Inside the “Properties” box, you’ll be able to view the products inside the workspace along with that workspace’s products.



If you click “New Workspace”, you will be automatically re-directed to a new open workspace. You will be able to change the name of this new workspace once inside. When the user logs into for the first time, they will be asked to create a new workspace.



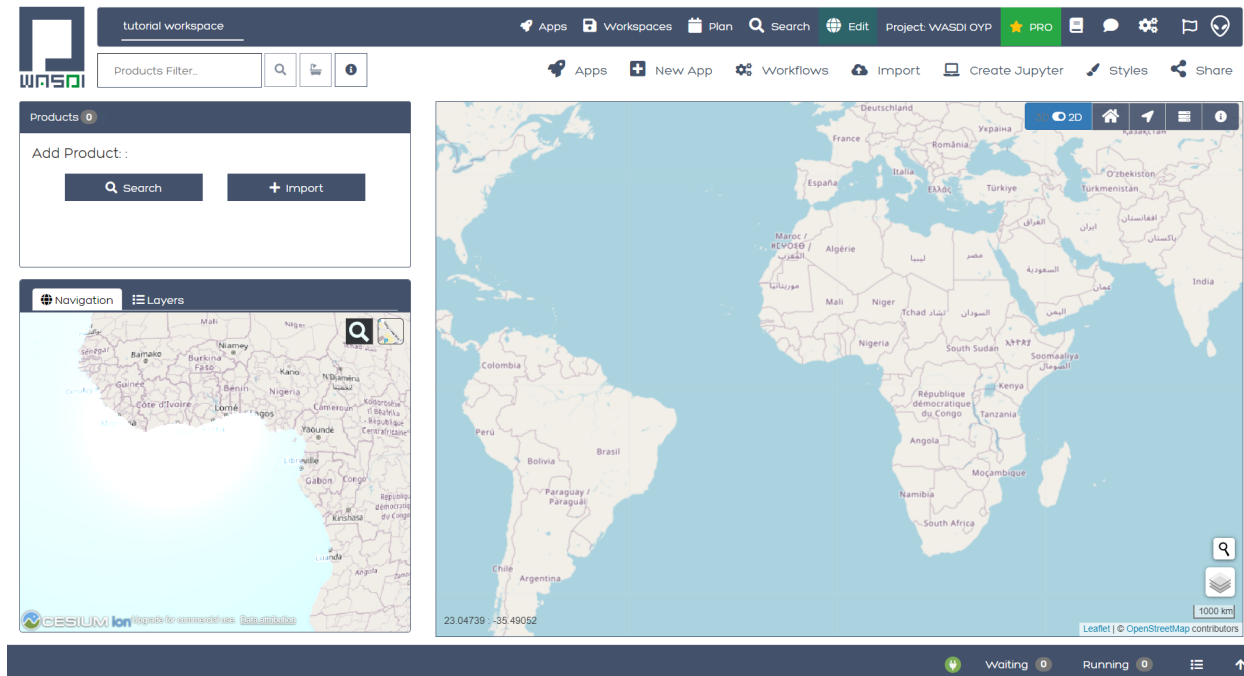
If you are not the owner of a workspace, but you wish to remove a workspace from your account, click the trashcan icon beside that workspace. This will remove your sharing permissions, not delete the workspace.



However, if you are the owner, then clicking the trashcan and confirming will permanently delete that workspace.

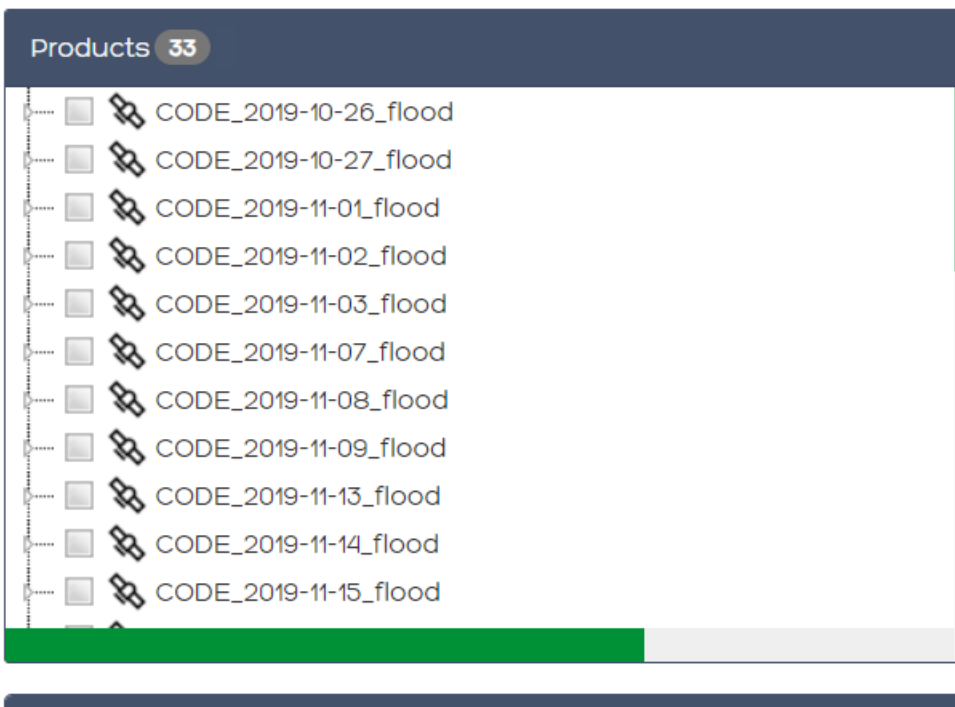
Just like when executing an application through the marketplace, if you click the “open” button beside a workspace name, you will be directed to that workspace.

If you decide to navigate away from the newly opened workspace, it will remain open in the Edit tab. As long as you keep WASDI open, this workspace will remain open until you open a different workspace.

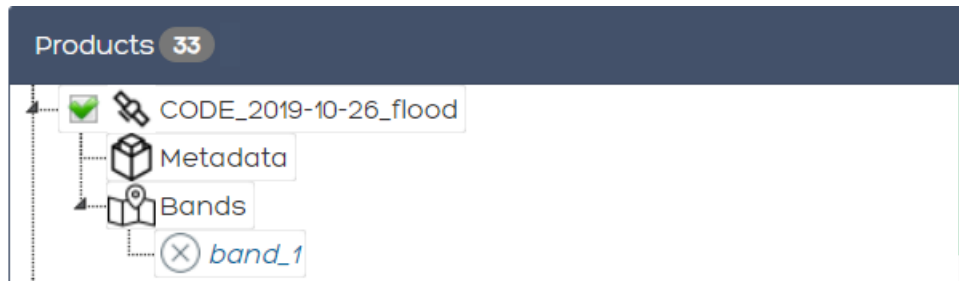


2.2.2 Interacting With Products

Once you have loaded products into your workspace, the products will be housed in the Products Box. The number beside the title “Products” represents how many products there are in your workspace.



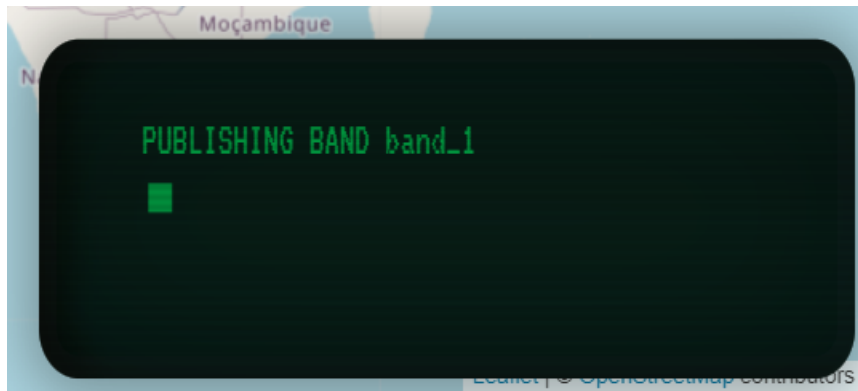
When you click on the arrow beside a product, you will open the Metadata and Bands nodes. The Bands node also have arrows and when clicked, the Bands will be shown (if there is any).



2.2.3 Publishing and Interacting with Bands

To publish a band, simply open the Bands node for the desired product and then click on the band you wish to publish. Depending on the band type, the amount of time it takes to publish the band will vary.

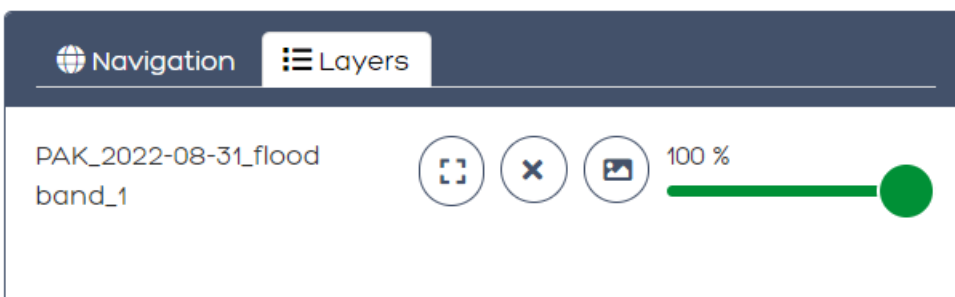
You will receive a notification in the bottom right corner that indicates that your band is being published.



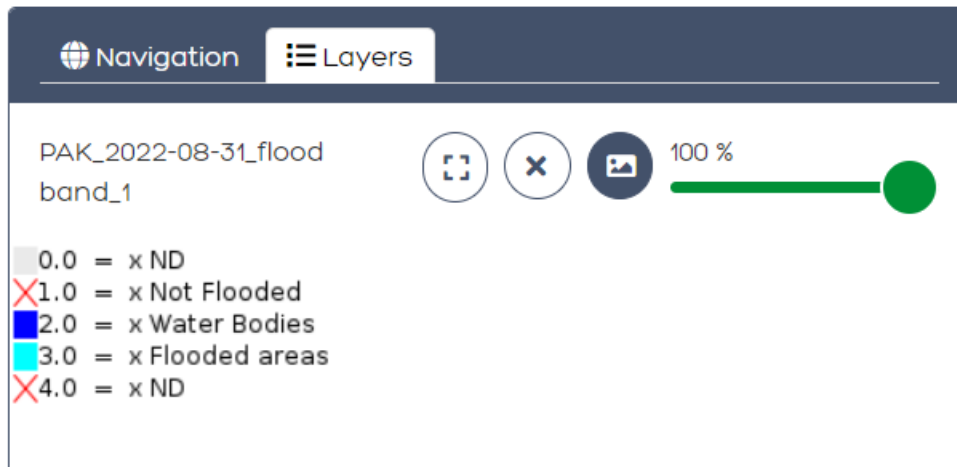
Once the publishing is complete, you will see that the band is on the 2D or 3D Map in the style applied to that product - note the workspace opens with your map component in 2D by default.

- For more information about Styles, please view the section on styles.

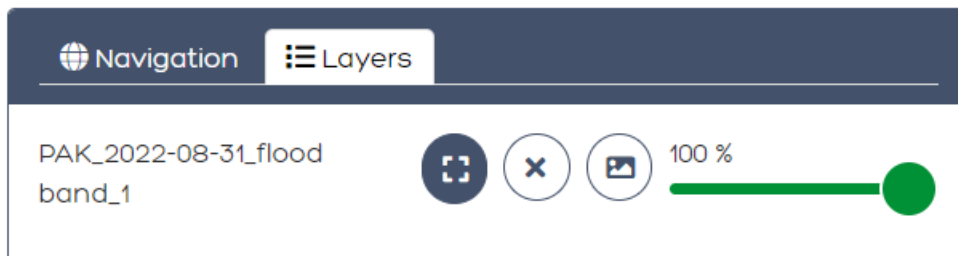
Once a band is published, you can set the opacity of the band by using the slider displayed next to that band's name in the "Layers" tab.



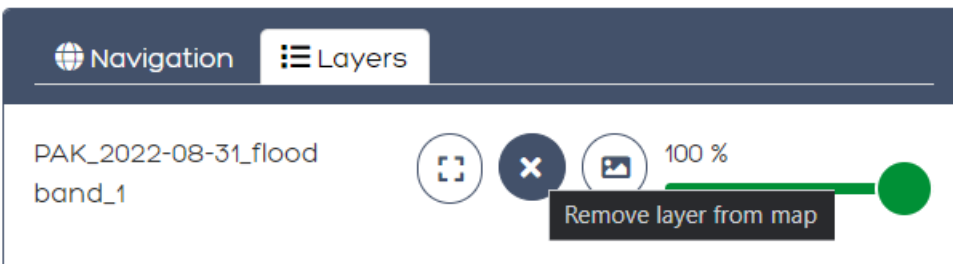
To view the legend for a band, click the "Legend" button to reveal the legend.



If you move away from a displayed band and wish to refocus on it quickly, click the “Navigate To” Button.

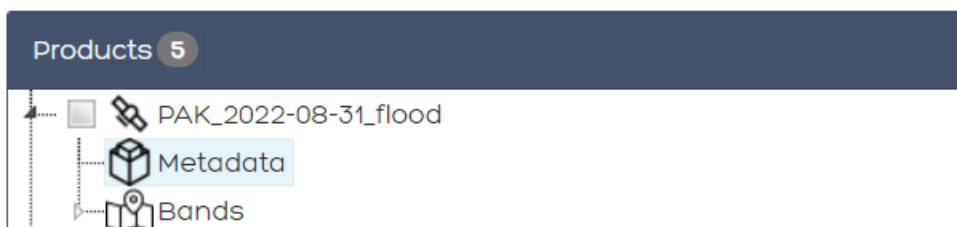


To remove the band from your map, click the “Remove layer from map” Button.



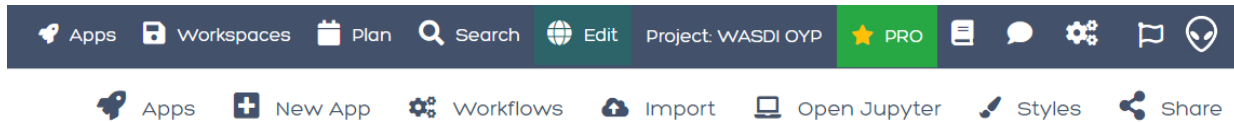
2.2.4 Reading Metadata

To read the Metadata of a Product, simply click “Metadata” and WASDI will begin fetching the Metadata.



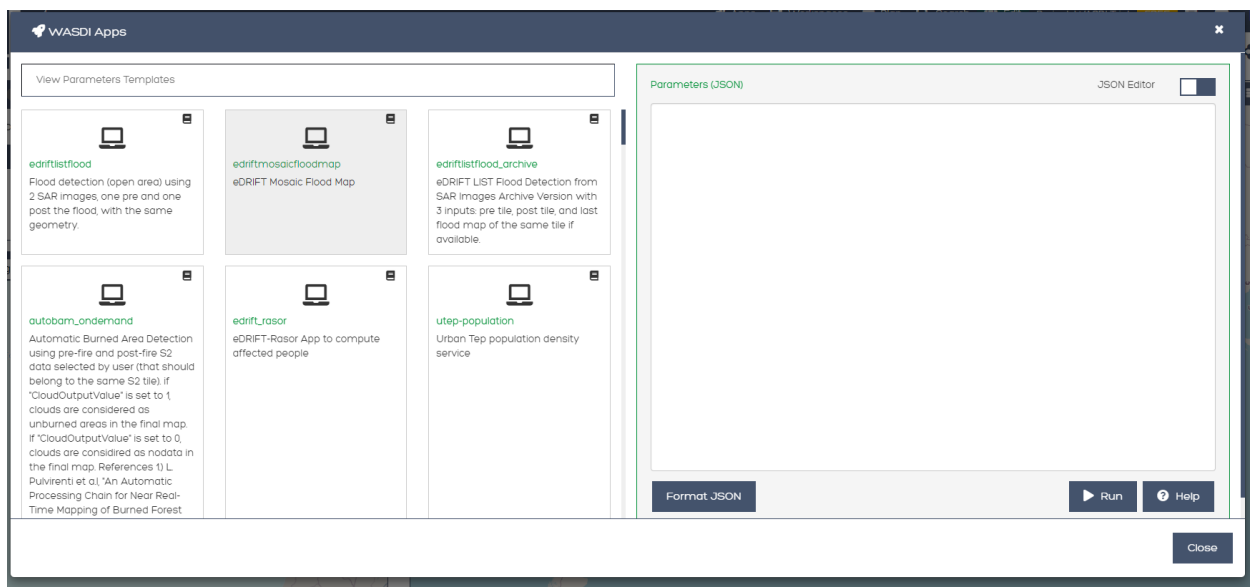
2.2.5 The Editor Toolbar

In the top corner of the workspace below the WASDI navbar, you will find the workspace Toolbar. Here you can execute different actions in order to work with your workspace.



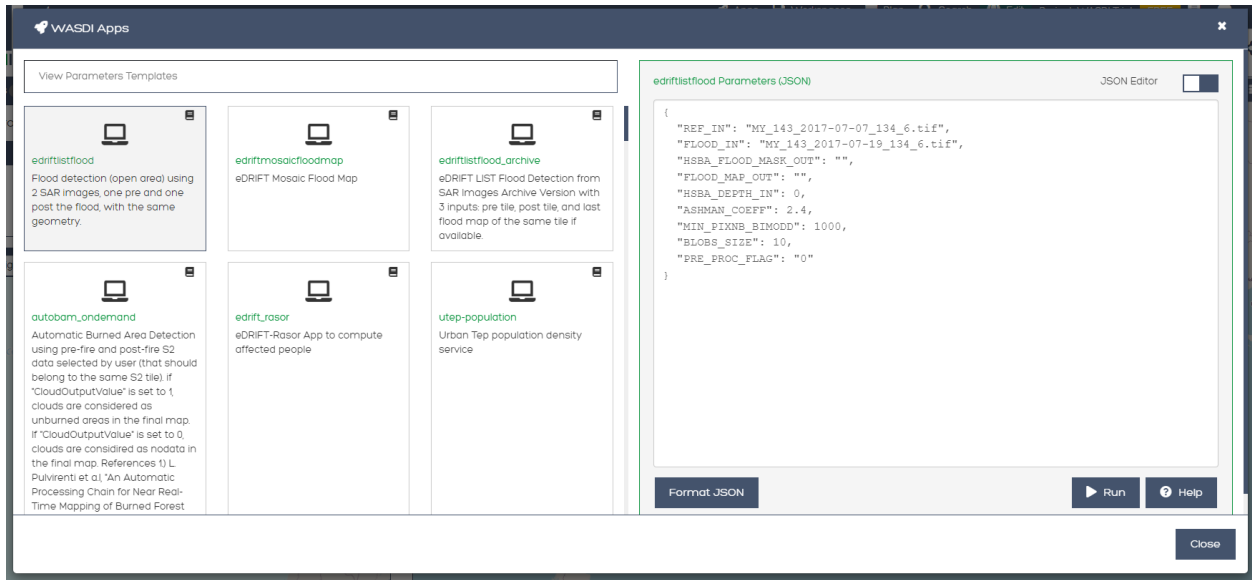
2.2.6 Apps

Clicking “Apps” will open a dialog box containing all WASDI applications. Applications will be displayed with different names, additionally, there may be apps here that are not in the marketplace.

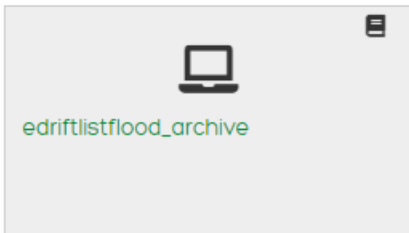


On the right-hand side of the dialog box, the processor parameters of the first application are displayed by default. When you select an application this will be updated automatically.

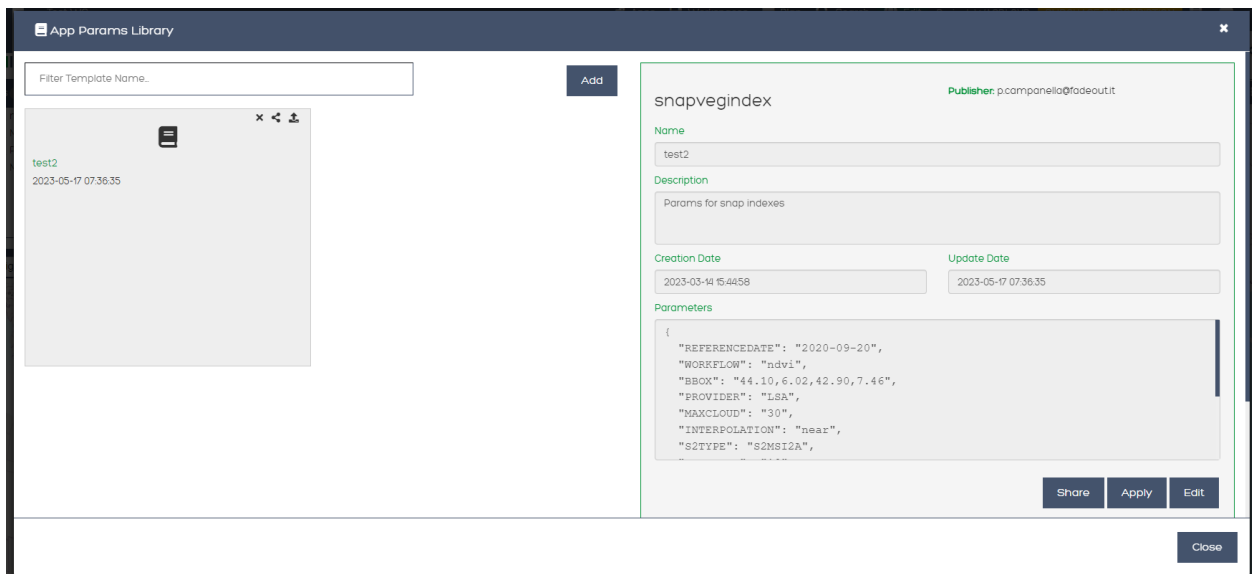
- At first the default parameters provided by the developer are displayed, but you can update them by changing any of the values in the JSON file and then executing the processor with those parameters by clicking “RUN”.



If you are not the owner of an application or an application has not been shared with you, you are still able to use that application and create processor parameters for it by clicking the book icon.



Processor Parameters are parameters you've created and saved to execute in that application.



You edit and share processor parameters as with any other WASDI elements.

Once you click “Apply” either on the processor Parameters card or in the information box, that parameters template will be automatically applied to the application so when you “Run” that application, the parameters you selected will

execute.

hellowasdi

Publisher: p.campanella@fadeout.it

Name

12345

Description

12345

Creation Date

2023-05-04 09:17:07

Update Date

2023-05-04 10:14:04

Parameters

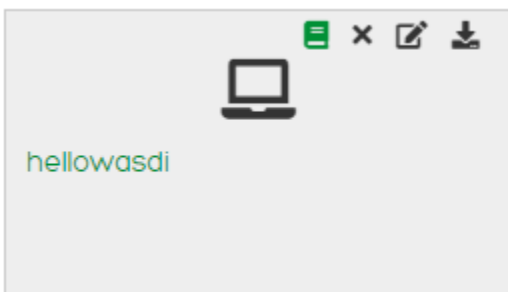
```
{  
  "NAME": "1234567"  
}
```

Share

Apply

Edit

If you are the owner or a processor has been shared with you, your toolbar inside the processor card will be different.



In regards to deleting a processor: A processor can only be deleted by the person that uploaded it. If the processor has been shared with you, by clicking the delete button, you are simply removing your permissions to access the processor - not the processor itself

- If you've removed your permissions by accident, contact the processor's developer or another WASDI user with permission to share the processor to have them grant you access once again.

You are also able to download a processor.

If you are the owner of a particular application or it has been shared with you, you will be able to edit it by clicking the page with pen button in the processor card.

The editable elements are:

- The Processor Information: i.e., the Short Description, the Programming Language, the Timeout in minutes, and the JSON Sample. From here you can also access WASDI's package manager for applications, which will be explained in depth in the PACKAGE MANAGER section.
- The Processor Store Information: i.e., how the processor appears in the app store (if at all) the Processor's Friendly Name (the name it can be searched by in the app marketplace), the application's information link if applicable), the developer's or association's email , the price of the application either as a description or on demand use price, a long description, and the categories.
- The Processor Media: i.e., the logo for the application or association, and any other associated imagery - up to seven (7) images.
- The Processor's Sharing Settings: i.e., WASDI users with whom the processor has been shared.
- The Processor UI: i.e., the input fields required to run the processor. If the processor is accessed through the marketplace, these UI fields will show as form inputs. Users will be able to update them manually through the JSON file as well.

2.2.7 New App

To upload a new application to WASDI, you can click the “NEW APP” button inside the workspace. This will open a new dialog box that resembles the edit application button in the “EDIT PROCESSOR” section. The difference here being that you cannot update the Store Information, Media. or Sharing options.

WASDI APP

Processor Store Media Share UI

Name

Type

Select Your Programming Language

Drag and Drop here the .zip file with your WASDI App or click to browse.

Short Description

JSON Sample

TimeOut [min]

Close Apply

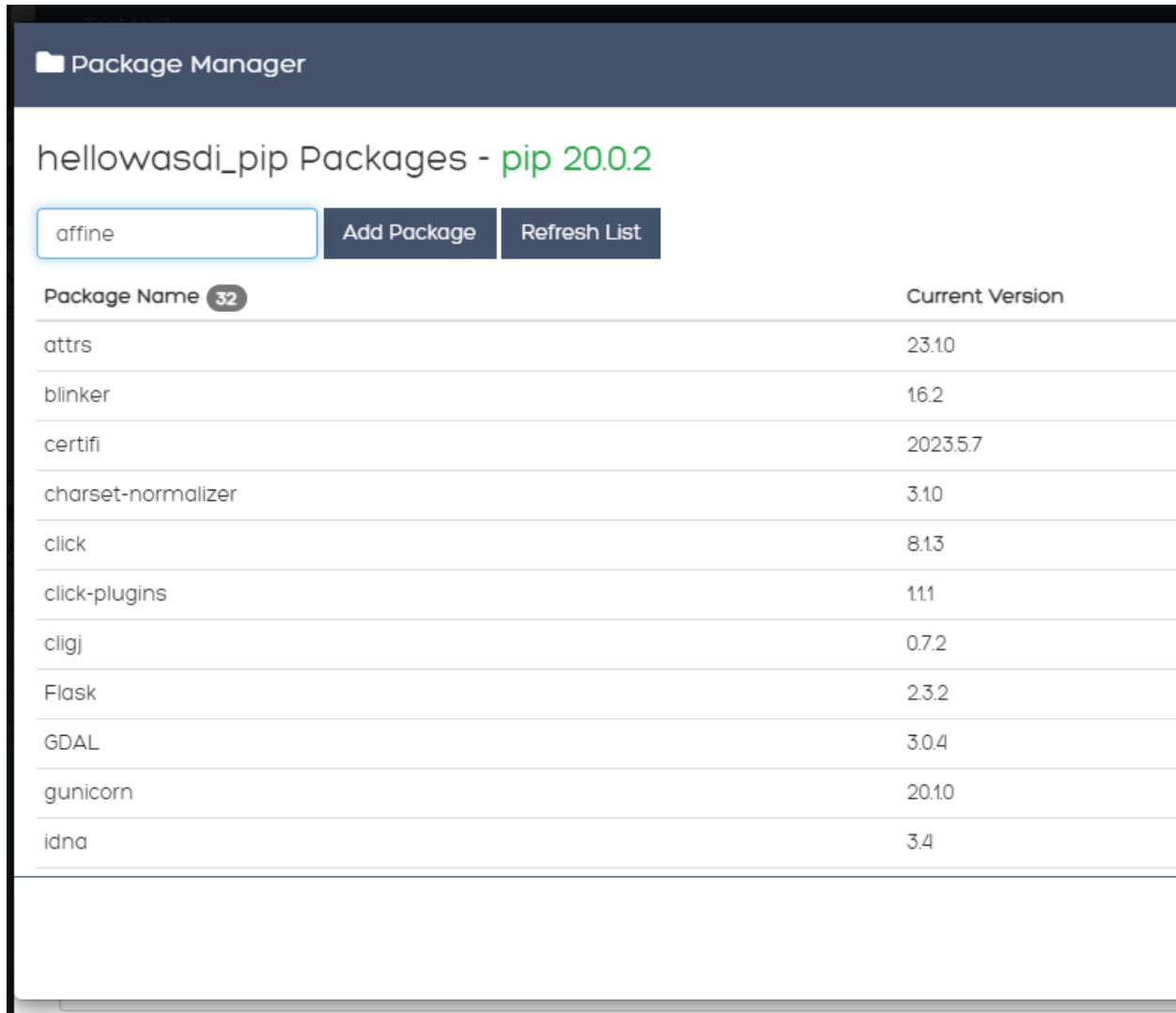
For more information on these options please see the section on Editable Elements in the APPS section.

2.2.8 Package Manager

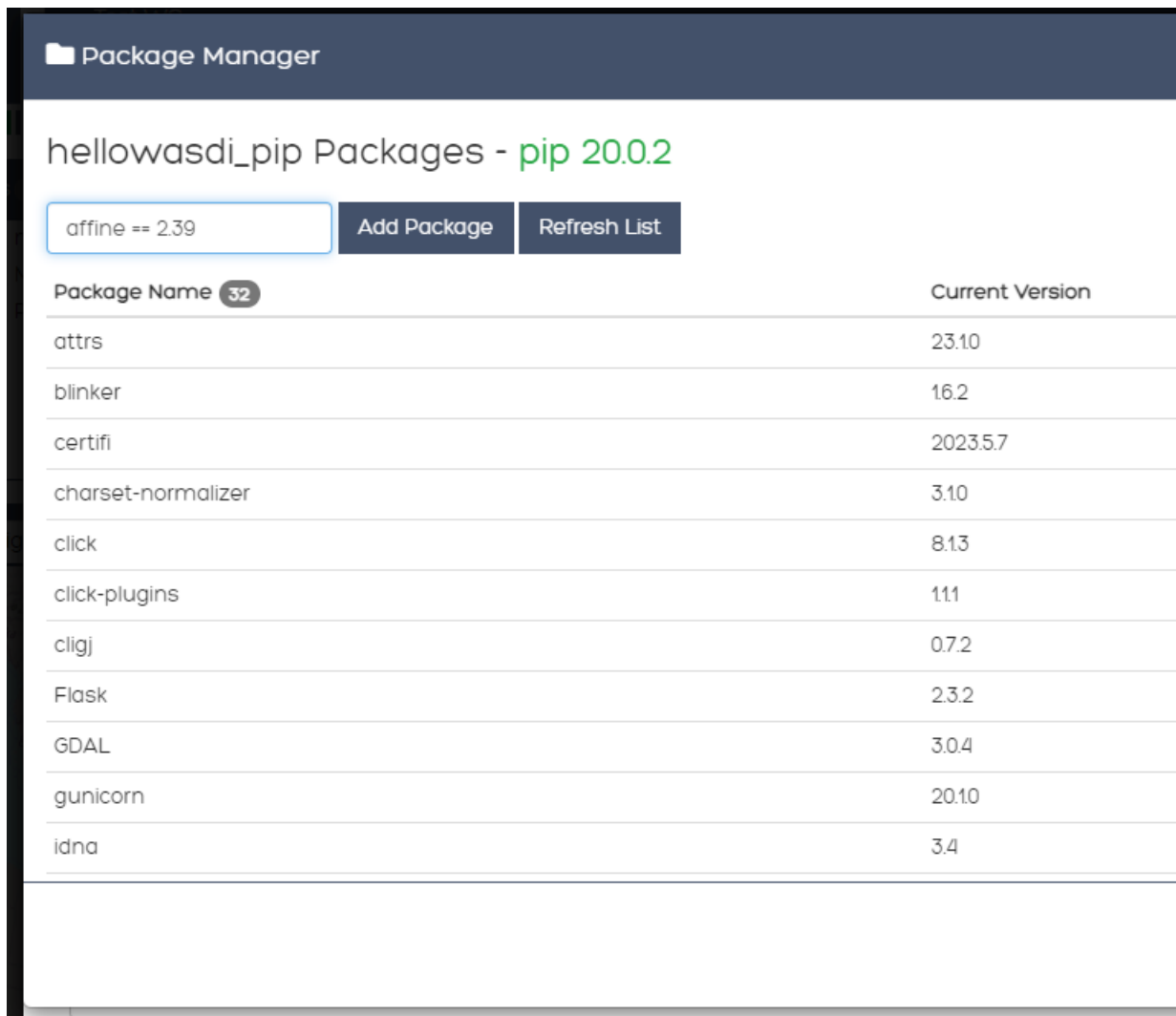
Once you've opened the Package Manager, you will be able to view all the packages the selected application is dependent on.

To add a package:

- Add the package by name (e.g., affine) - ensure it is spelled correctly, or the package will not be added.
- This method will add the package in its most current version. (e.g., affine 2.40 will be added)



To add a specific version of the package use the following construction - affine == 2.39



The screenshot shows the 'Package Manager' interface. At the top, it says 'hellowasdi_pip Packages - pip 20.0.2'. Below this is a search bar containing 'affine == 2.39', an 'Add Package' button, and a 'Refresh List' button. A table lists the installed packages with their names and current versions. The table has two columns: 'Package Name' and 'Current Version'. The packages listed are: attrs (23.10), blinker (16.2), certifi (2023.5.7), charset-normalizer (3.10), click (8.13), click-plugins (1.1.1), cligj (0.7.2), Flask (2.3.2), GDAL (3.0.4), gunicorn (20.10), and idna (3.4).

Package Name	Current Version
attrs	23.10
blinker	16.2
certifi	2023.5.7
charset-normalizer	3.10
click	8.13
click-plugins	1.1.1
cligj	0.7.2
Flask	2.3.2
GDAL	3.0.4
gunicorn	20.10
idna	3.4

To remove a Package, simply click the remove package button (the Trashcan icon) and confirm.

- When updating the Package Manager in any way, it may take some time to communicate the changes to the WASDI servers. This is normal and if you close the package manager and re-open it before the action is complete, it may not be immediately reflected in your dashboard. To check if the action was completed, click “Refresh List”.

To automatically update a Package, click the Update Package button (the Upwards facing Arrow). This action updates the selected package to the most recent version.

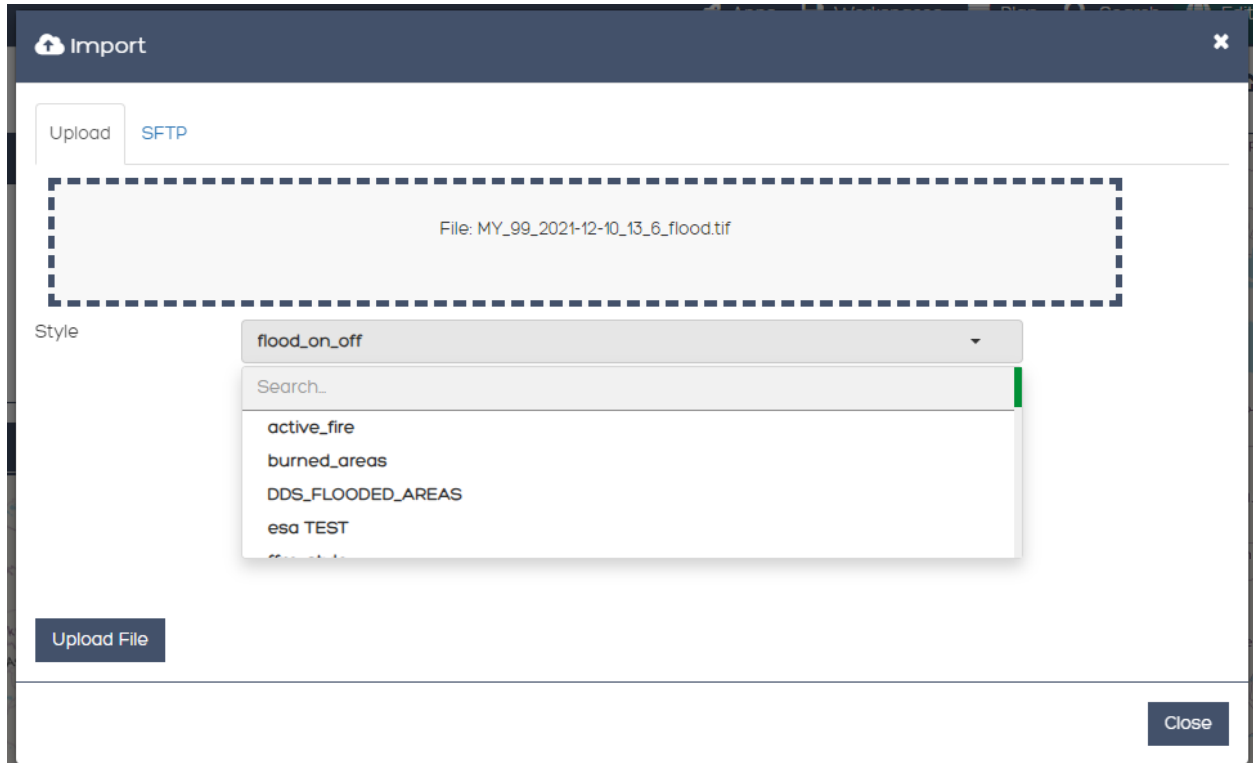
You can search for packages by name in the ‘Search Packages’ input bar. The search is NOT case-sensitive.

2.2.9 Workflows

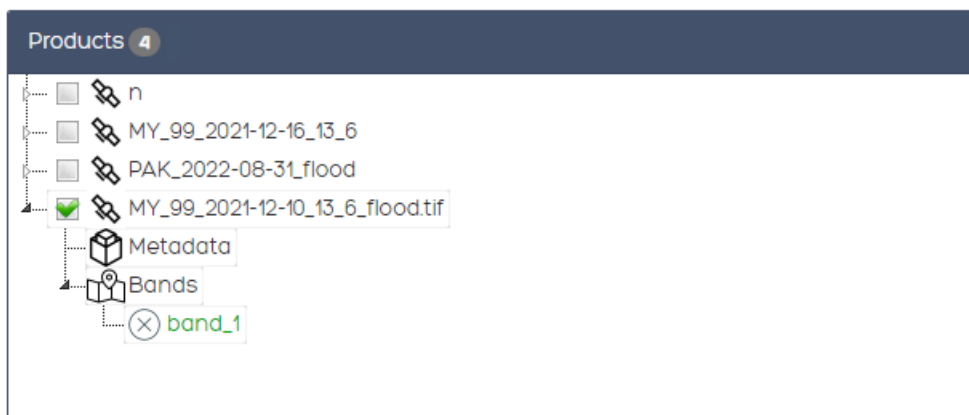
2.2.10 Import

If you have an image that you would like to use in an existing workspace, you can use the import dialog. You can drag and drop the file into the box or click the box to search for a file on your machine.

You may also select a Style to apply to this product from the Style dropdown menu.

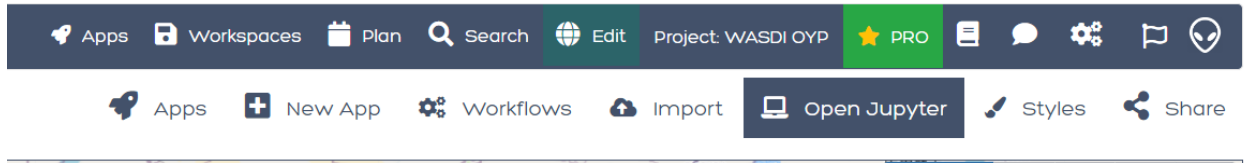


You will receive a notification once your product has been uploaded and then it will be added to the list of products in your workspace.

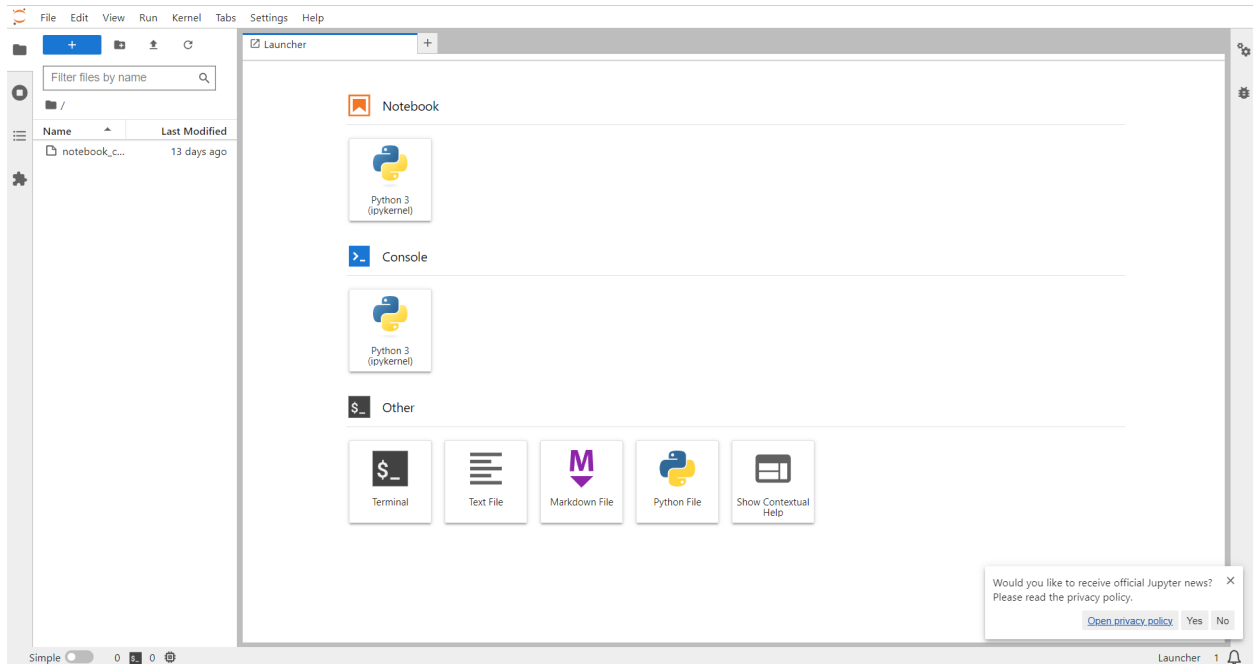


2.2.11 Open Jupyter

By clicking “Open Jupyter” WASDI begin preparing a Jupyter notebook workspace automatically. This progress on this process will show up in the Processes Progress Bar.



When the Jupyter Notebook workspace has been prepared, you will receive a notification.

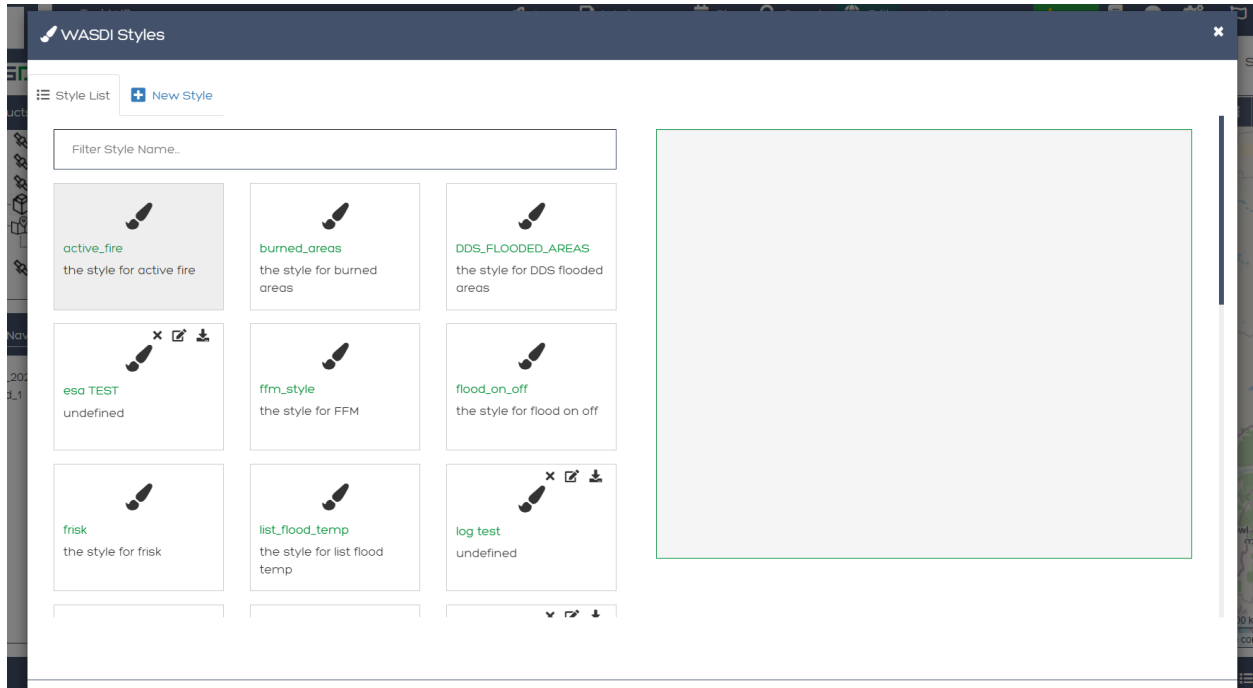


Note: Learn more in the ``tutorial about working with Jupyter notebooks in WASDI<../ProgrammingTutorials/JupyterNotebookTutorial.rst>`_`

2.2.12 Styles

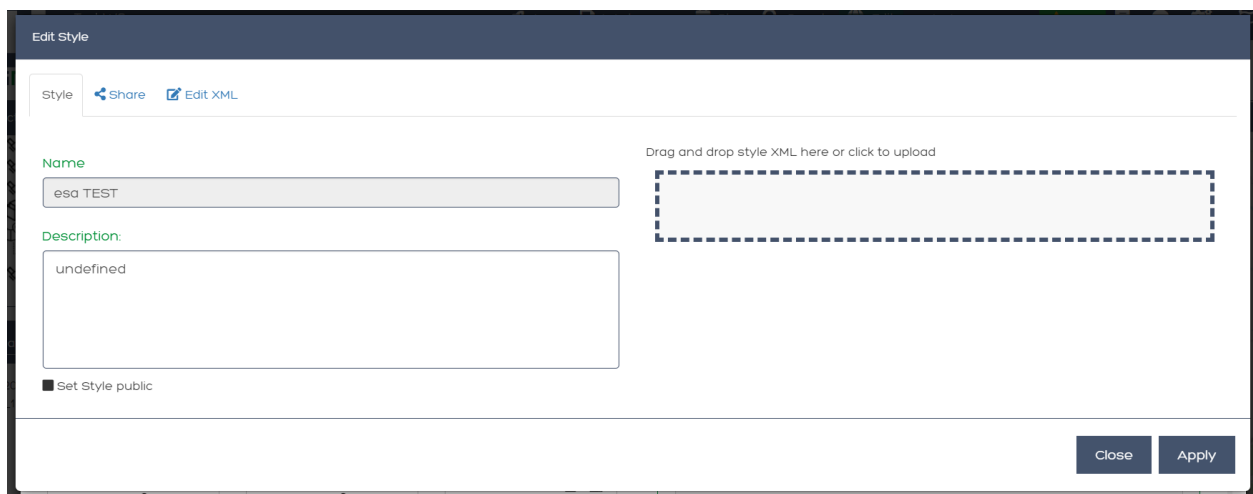
The Styles Button will open a dialog displaying all the styles available to WASDI users. These styles will allow you to change the appearance of published images.

- This is an information viewing dashboard. You cannot apply styles from this dialog. For information on how to apply styles to products, see the section on editing existing products.

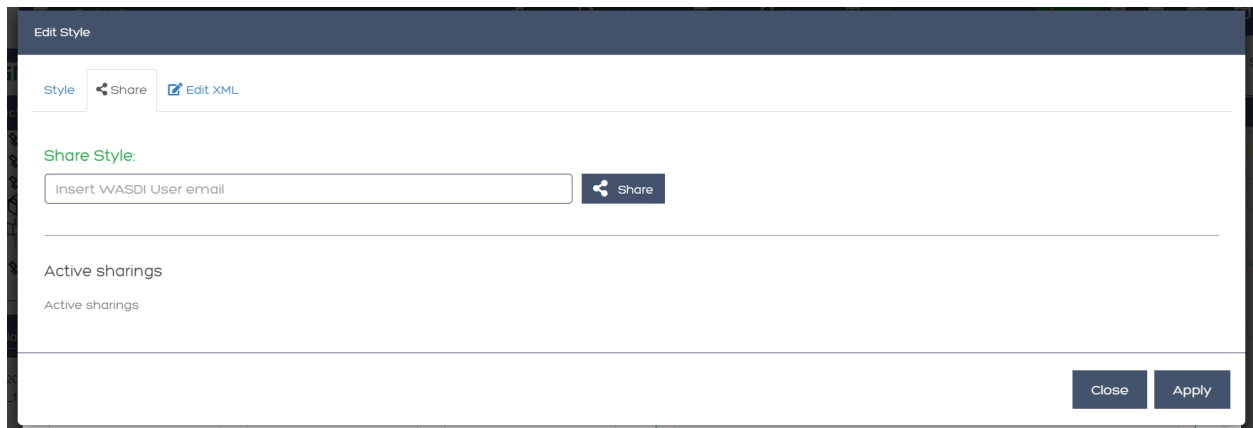


If you are the owner of an existing style or permissions to the style have been shared with you, you can make changes to a style by clicking the Edit Style button.

In the Edit Style Dialog you can edit the name, description, and XML file (by adding a new one), and set the Style to public so all WASDI users can access it.



From this same dialog you can manage the users with whom the style has been shared.

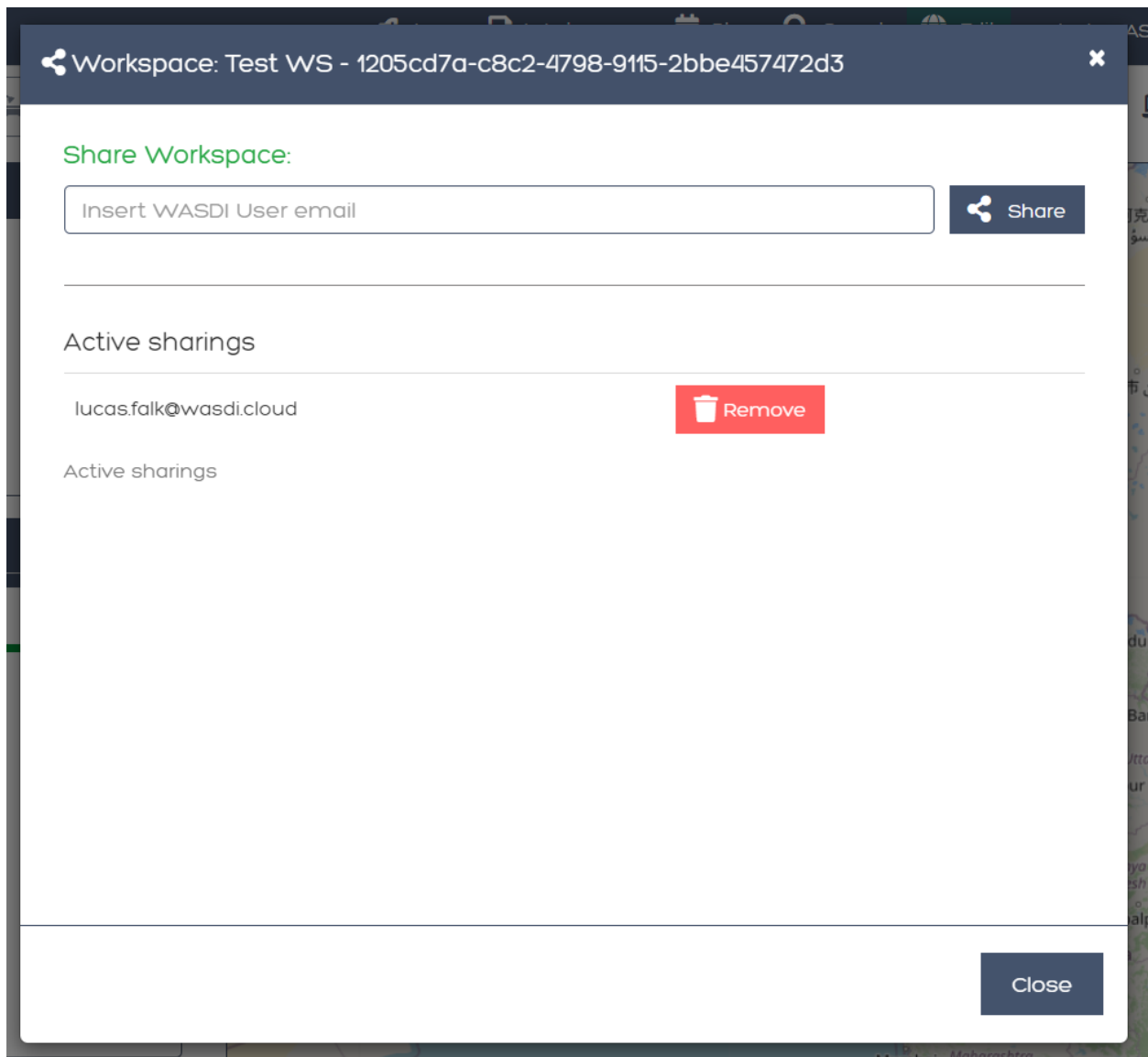


You can also edit the XML file by accessing the “Edit XML” Tab.



2.2.13 Share

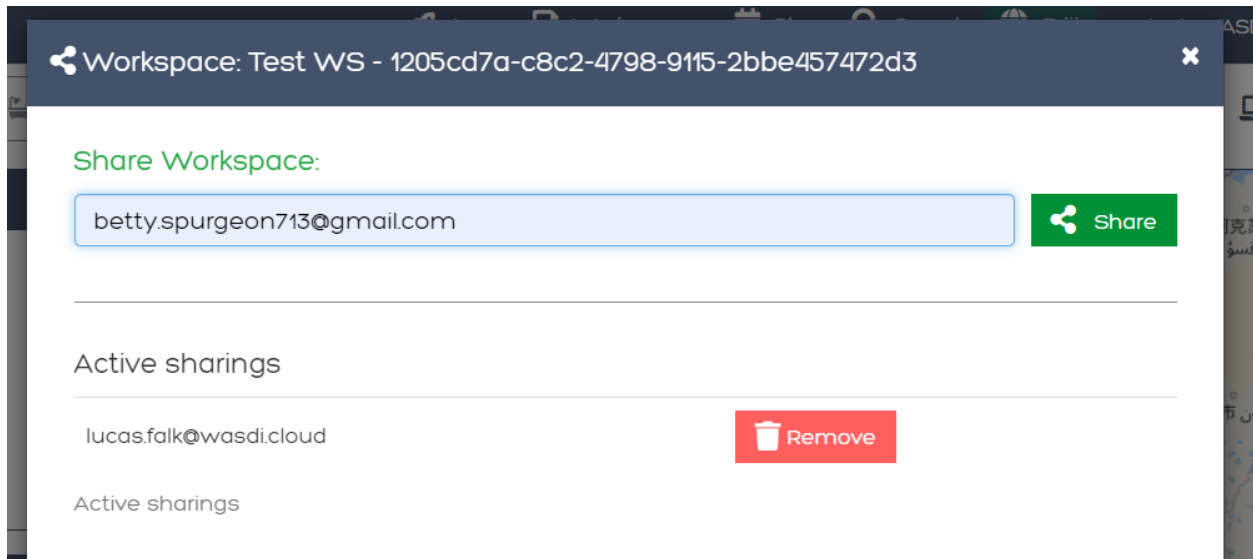
The Share button will open a dialog box displaying all the users (if any) that this workspace has been shared with.



Both the owner of the workspace and all the users they've shared it with can manage the shared users.

To search a user to share the workspace with, search for the user's email address associated with their WASDI account and click "Share".

- If the email address was correct then the sharing will be executed automatically.

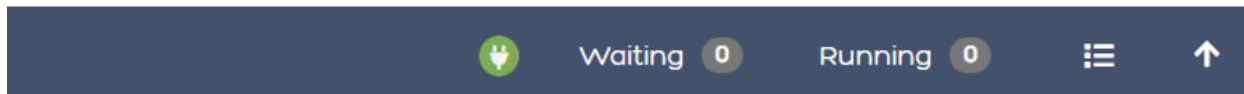


To remove a user from the workspace, simply click “Remove” and once you confirm that you wish to remove them, their permissions will be removed automatically. Note: you can give permissions back to a removed user at any time.

2.2.14 The Processes Bar

The processes bar is the most useful component in WASDI. This is where you can find information about actions executed in your workspace.

When closed, the processes bar will display the status of websocket (Green for connected, red for disconnected), the number of processes waiting, the number of processes running, a button to open your Workspaces Processes List dialog, and an arrow to open the processes bar.



When the processes bar is open, you will be able to view all to view the most recent 5 processes that were executed in your workspace. You can see how long it took for the processor to work (or if it is ongoing).

	Operation	Name	User	Size	Created	Started	Progress	Duration	
Log	READMETADATA	PAK_2022-08-31_flood.tif	betty.spurgeon@wasdi.cloud		2023-07-12 12:34:34	2023-07-12 12:34:36	100 %	00:00:00	
Log	APP	hellowasdi	betty.spurgeon@wasdi.cloud		2023-07-10 11:19:51	2023-07-10 11:19:53	100 %	00:00:05	
Log	APP	hellowasdi	betty.spurgeon@wasdi.cloud		2023-07-10 11:18:24	2023-07-10 11:18:27	100 %	00:00:05	
Log	APP	hellowasdi	betty.spurgeon@wasdi.cloud		2023-07-10 11:15:38	2023-07-10 11:15:41	100 %	00:00:16	
Log	PUBLISH	MY_99_2021-12-10_13_6_flood.tif	betty.spurgeon@wasdi.cloud	707.3 KB	2023-07-10 06:56:12	2023-07-10 06:56:14	100 %	00:00:03	

[Load more...](#)

In the right-most column, you will see either one or two icons. If you click the “Logs” icon you will open the logs for that operation.

Log	APP	hellowasdi	betty.spurgeon@wasdi.cloud		2023-07-10 11:19:51	2023-07-10 11:19:53	100 %	00:00:05	
---------------------	-----	------------	----------------------------	--	---------------------	---------------------	-------	----------	--

WASDI Processor Logs ID: ce6197ff-169c-4544-bb56-4f8cf541e640

Search by text:

Operation: APP Name: hellowasdi Status: DONE

Download

Date: ▼	Row
2023-07-10 11:19:54	wasdiexecuteProcessor Process finished. Forcing status to DONE
2023-07-10 11:19:54	wasdiexecuteProcessor Done
2023-07-10 11:19:54	Hello wasdi
2023-07-10 11:19:54	Start myProcessor Python 3.7
2023-07-10 11:19:54	wasdiexecuteProcessor RUN ce6197ff-169c-4544-bb56-4f8cf541e640
2023-07-10 11:19:54	wasdiProcessorServer Process Started with local pid 18
2023-07-10 11:19:54	wasdiProcessorServer RUN ce6197ff-169c-4544-bb56-4f8cf541e640

First

Previous

1

Next

Last

Refresh

Close

In the logs dialog, you have the option to download a record of the logs in a .txt file.

If you click the “Payload” icon, you will open a dialog where you can view the payload of that operation and copy it to your clipboard.

Log

PUBLISH

MY_99_2021-12-10_13_5_flood.tif

bettyspurgeon@wasdi.cloud

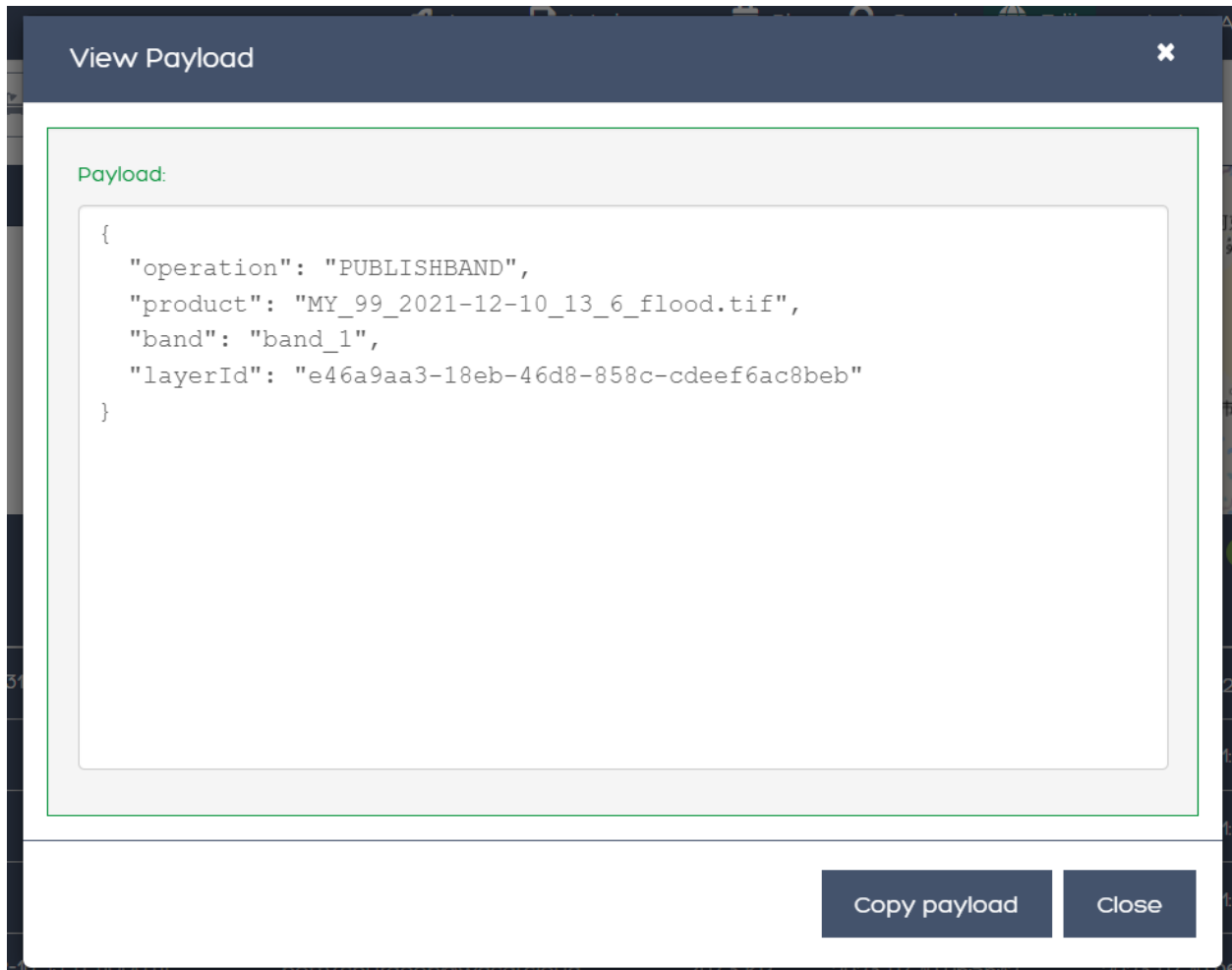
707.3 KB

2023-07-10 06:56:12

2023-07-10 06:56:14








100 %

00:00:03



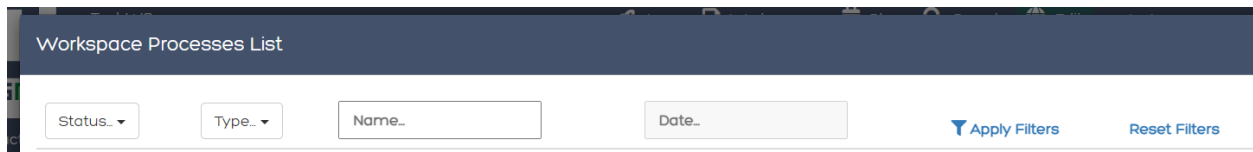
2.2.15 The Workspace Processes List

To open the workspace processes list, click either the “Load More” button at the bottom of the open processes bar or the “Open Processes List” button representing by the list icon.

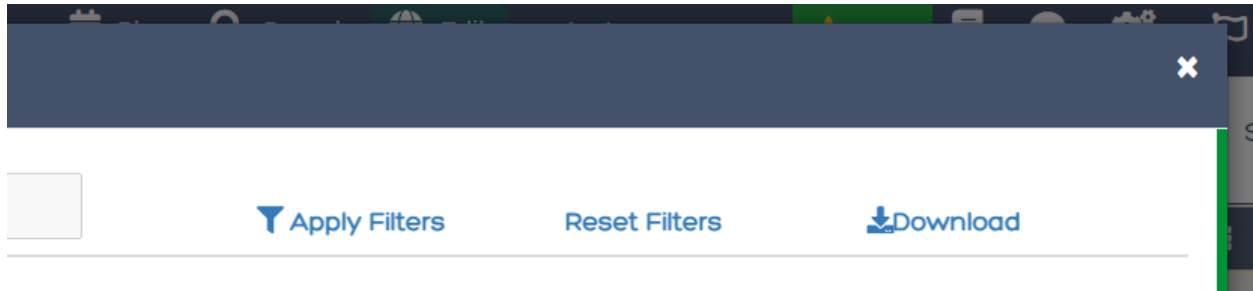
 Waiting 0 Running 0  			
Started	Progress	Duration	
2023-07-12 12:34:36	<div>100 %</div>	00:00:00	
2023-07-10 11:19:53	<div>100 %</div>	00:00:05	
2023-07-10 11:18:27	<div>100 %</div>	00:00:05	
2023-07-10 11:15:41	<div>100 %</div>	00:00:16	

Operation	Name	User	Size	Created	Started	Progress	Duration
Log READMETADATA	PAK_2022-08-31_flood.tif	betty.spurgeon@wasdi.cloud		2023-07-12 12:34:34	2023-07-12 12:34:36Z	100 %	00:00:00
Log APP	hellowasdi	betty.spurgeon@wasdi.cloud		2023-07-10 11:19:51	2023-07-10 11:19:53Z	100 %	00:00:05
Log APP	hellowasdi	betty.spurgeon@wasdi.cloud		2023-07-10 11:18:24	2023-07-10 11:18:27Z	100 %	00:00:05
Log APP	hellowasdi	betty.spurgeon@wasdi.cloud		2023-07-10 11:15:38	2023-07-10 11:15:41Z	100 %	00:00:16
Log PUBLISH	MY_99_2021-12-10_13_6_flood.tif	betty.spurgeon@wasdi.cloud	707.3 KB	2023-07-10 06:56:12	2023-07-10 06:56:14Z	100 %	00:00:03
Log INGEST	MY_99_2021-12-10_13_6_flood.tif	betty.spurgeon@wasdi.cloud	707.3 KB	2023-07-10 06:56:02	2023-07-10 06:56:04Z	100 %	00:00:01
Error WORKFLOW		betty.spurgeon@wasdi.cloud		2023-07-07 09:20:30	2023-07-07 09:20:32Z	100 %	00:00:01
Log APP	hellowasdi	betty.spurgeon@wasdi.cloud		2023-07-07 09:00:06	2023-07-07 09:00:08Z	100 %	00:00:04
Log KILLPROCESSTREE	ee256f6b-a6b8-40a5-8483-61ce13fae08d	betty.spurgeon@wasdi.cloud		2023-07-07 06:35:49	2023-07-07 06:35:52Z	100 %	00:00:00










In the processes list dialog you can search for specific processes by name or filter your processes based on their status (Any, Created, Running, Waiting, Ready, Done, Error, Stopped), type (Any, Run Processor, Run IDL, Run MatLab, Ingest, Download, Publish Band, Graph, Deploy Processor, Copy to SFTP, FTP Upload, Mosaic, Multi-subset), or the date. To apply the filters, set the filters you wish to use and then click “Apply Filters”. To remove filters you applied, simply click “Reset Filters”.



You can also download a copy of all the processes executed in this workspace by clicking “Download”. You will receive a .csv file.



Similarly to the processes bar, you can open the logs and payload dialog for any process that has them from the processes list.

betty.spurgeon@wasdi.cloud		2023-07-10 11:19:51	2023-07-10 11:19:53Z	100 %	00:00:05		
betty.spurgeon@wasdi.cloud		2023-07-10 11:18:24	2023-07-10 11:18:27Z	100 %	00:00:05		
betty.spurgeon@wasdi.cloud		2023-07-10 11:15:38	2023-07-10 11:15:41Z	100 %	00:00:16		
betty.spurgeon@wasdi.cloud	707.3 KB	2023-07-10 06:56:12	2023-07-10 06:56:14Z	100 %	00:00:03	 	
betty.spurgeon@wasdi.cloud	707.3 KB	2023-07-10 06:56:02	2023-07-10 06:56:04Z	100 %	00:00:01	 	
betty.spurgeon@wasdi.cloud		2023-07-07	2023-07-07	100 %	00:00:01		

2.2.16 The Map

The Map inside your workspaces features that can be found in the top right-hand corner of the map box.

To switch between the 2D leaflet map and the 3D Cesium globe, click the 3D/2D toggler.



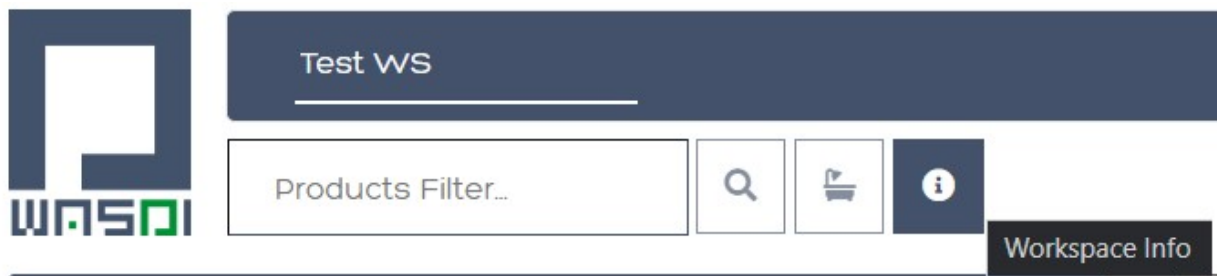
The dimension that is not currently active in the large map box will be shown in the Navigation tab of the Navigation/Layers box.

The home button is used to navigate back to the “home” bounding box for the workspace.

When the arrow button is clicked, the map in the main view will synchronise to that of the navigation tab (smaller map under your products).

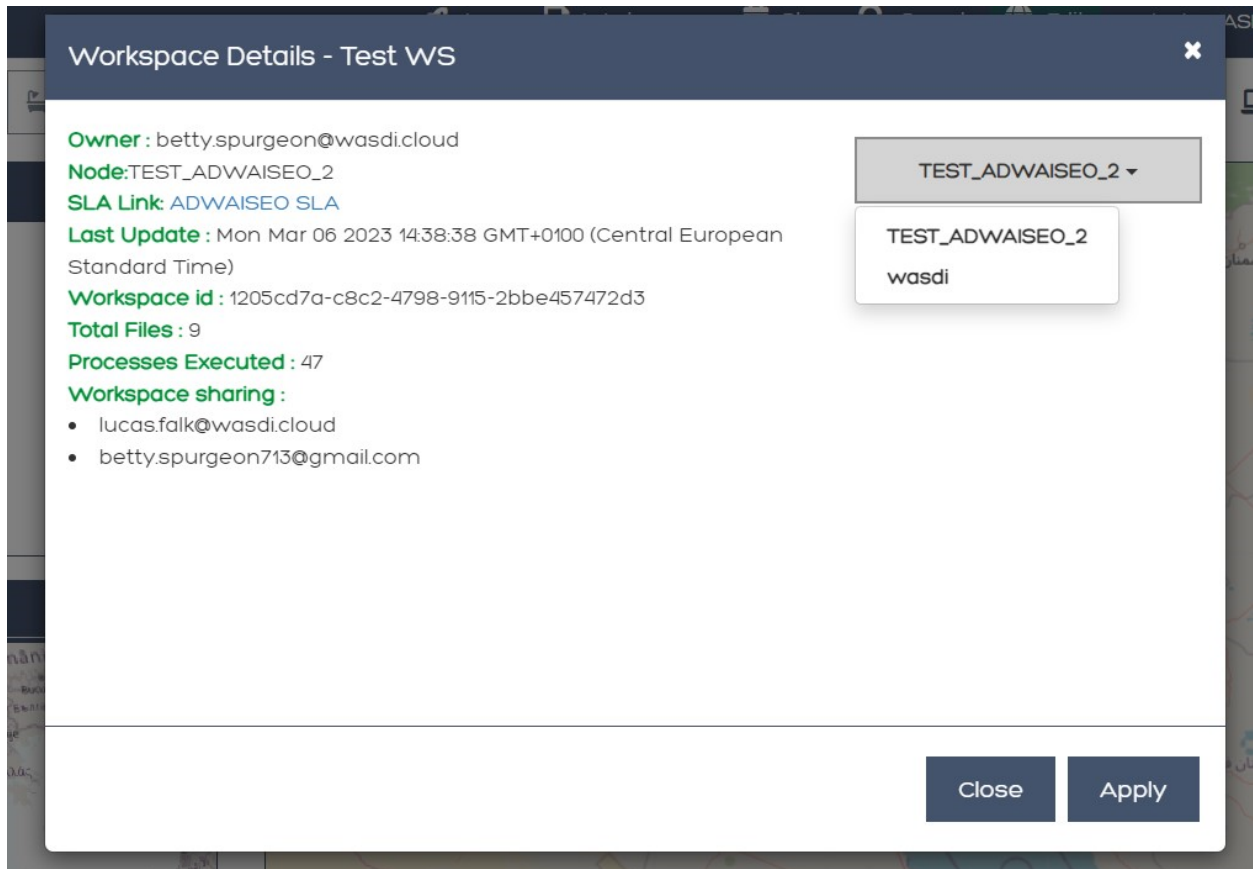
2.2.17 Workspace Details

In the top left-hand corner beside the Products filter, you will find the button to open your Workspace Details



Once open, you will find information about your workspace and the Node which houses your workspace.

You are able to change the node by selecting one from the dropdown menu

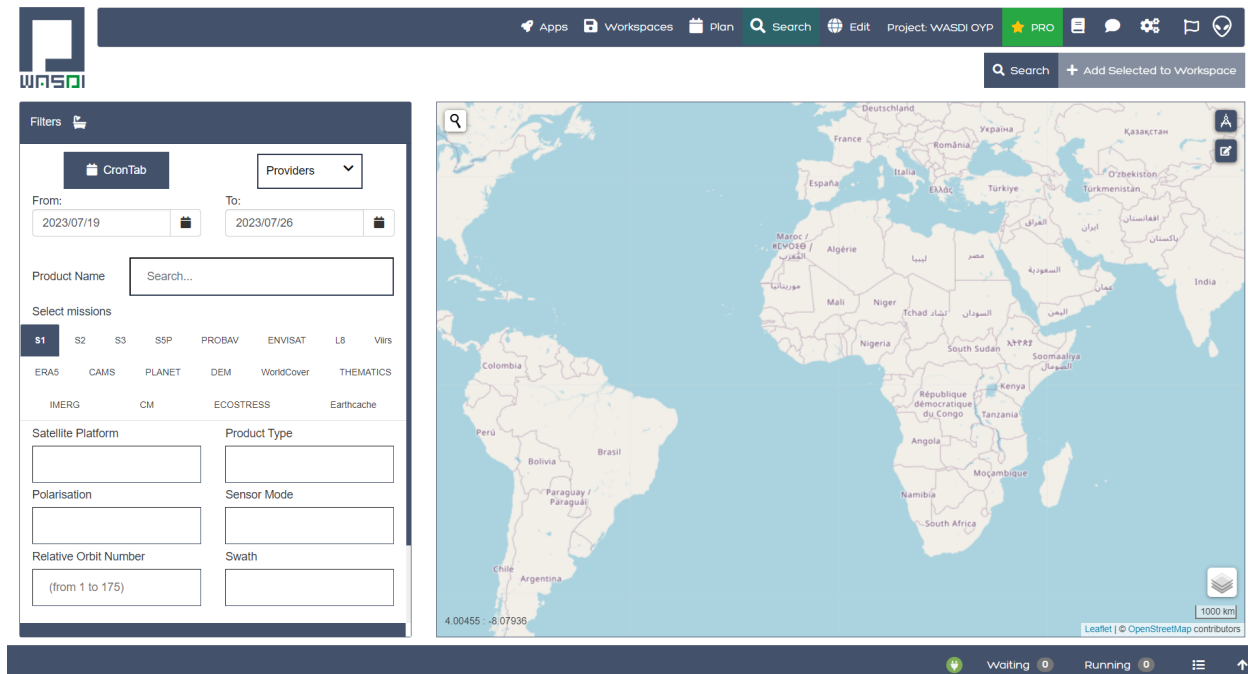


If you are using an ADWISEO node, you will also be able to view the ADWISEO SLA to review their terms of service.

2.3 Searching for Products

The search tab is used to search for images that have been recorded in the past. To execute a search:

- Select a date range for your search. You can either select a specific date range or you may use the CronTab to search more broadly for a season in a certain year (e.g., Spring 2022).



- You may enter a specific product name if you would prefer (e.g., IW_RAW__0S).
- To enable one mission specific filter, first select the tab and the Checkbox of the mission. If no checkbox is selected, the system will search all the available missions, otherwise, selected it will search only for the selected ones. (For example, with S1 - Sentinel-1, you can further narrow your search with Satellite Platform, Polarisation, Sensor Mode, Relative Orbit Number, and Swath - these inputs will likely change based on the mission you select).

From: To:

Product Name:

Select missions

☒ S1 ☐ S2 ☐ S3 ☐ PROBAV ☐ ENVISAT

☐ L8 ☐ CMEMS ☐ VIIRS

Satellite Platform:

Product Type:

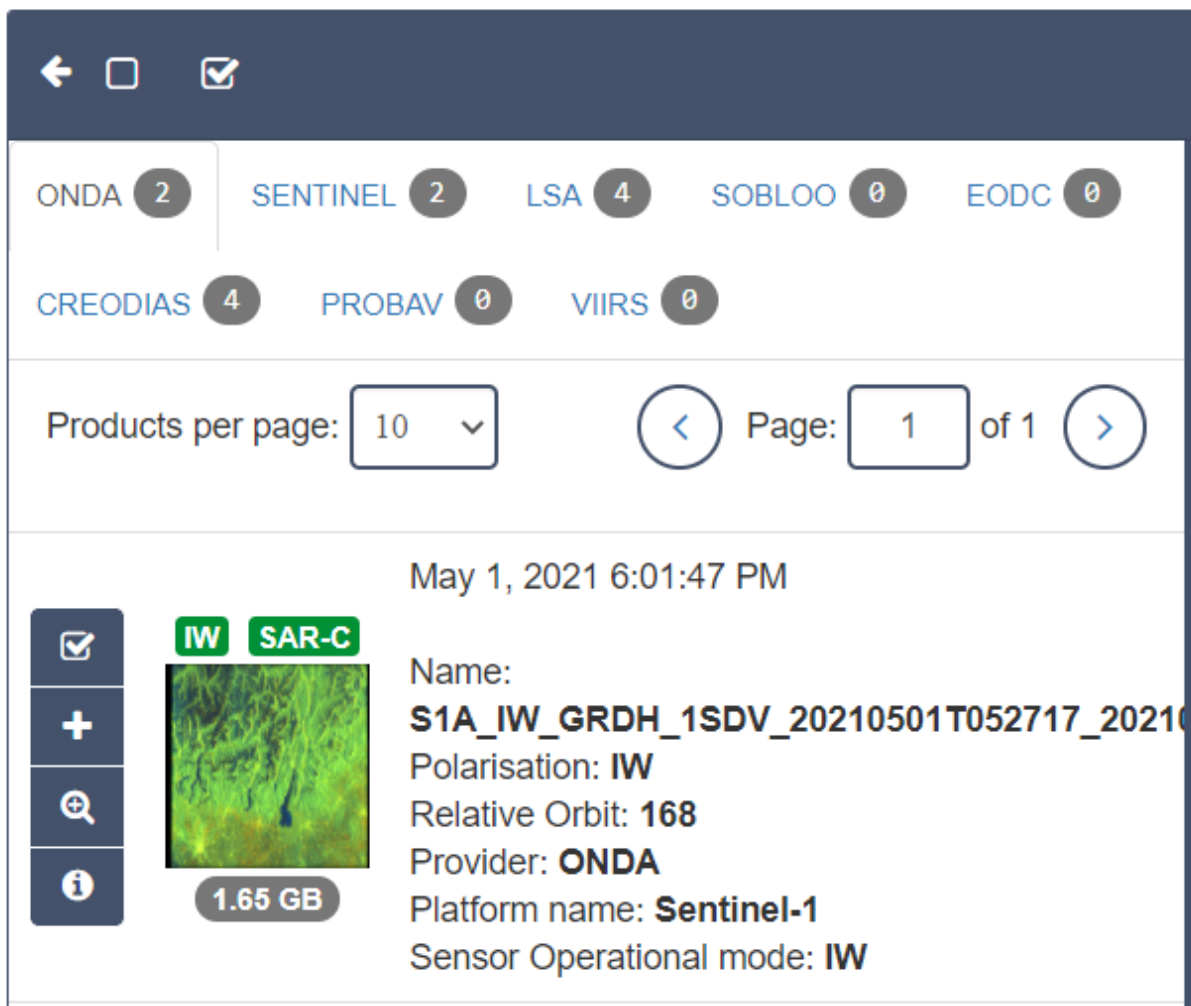
Polarisation:

Sensor Mode:

Relative Orbit Number:

Swath:

WASDI has a Multi Provider search system: this means that the same query is sent to many data providers. The user can switch providers on/off:

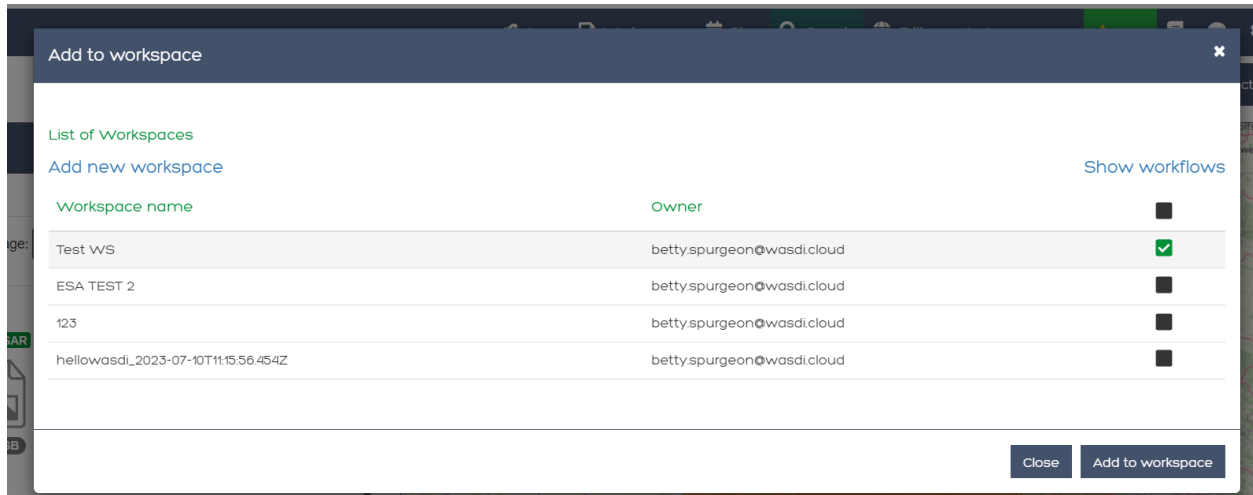


The SERCO ONDA DIAS is the **priority one** Provider because data is stored in the Cloud where WASDI is installed so this is, usually, the fastest provider available.

To start a query click on the SEARCH button on the top right of the screen:



Finally, you must select a specific bounding box (geographic location) to search for images.

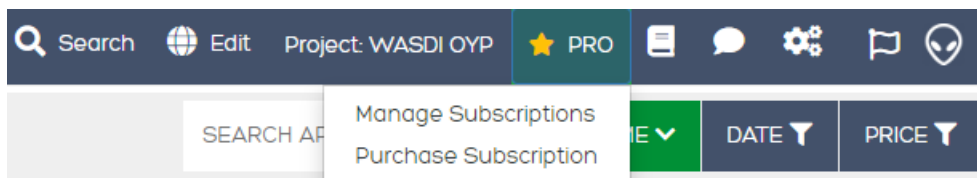


2.4 Managing Subscriptions and Organizations



















WASDI has recently implemented a subscription based usage model. To execute various actions in WASDI, the user must have an a valid subscription and an active project.

2.4.1 Managing Subscriptions

To review your subscriptions, navigate to the subscription dashboard by selecting your subscription type (in this example, we are using a PRO account) and click “Manage Subscriptions”.

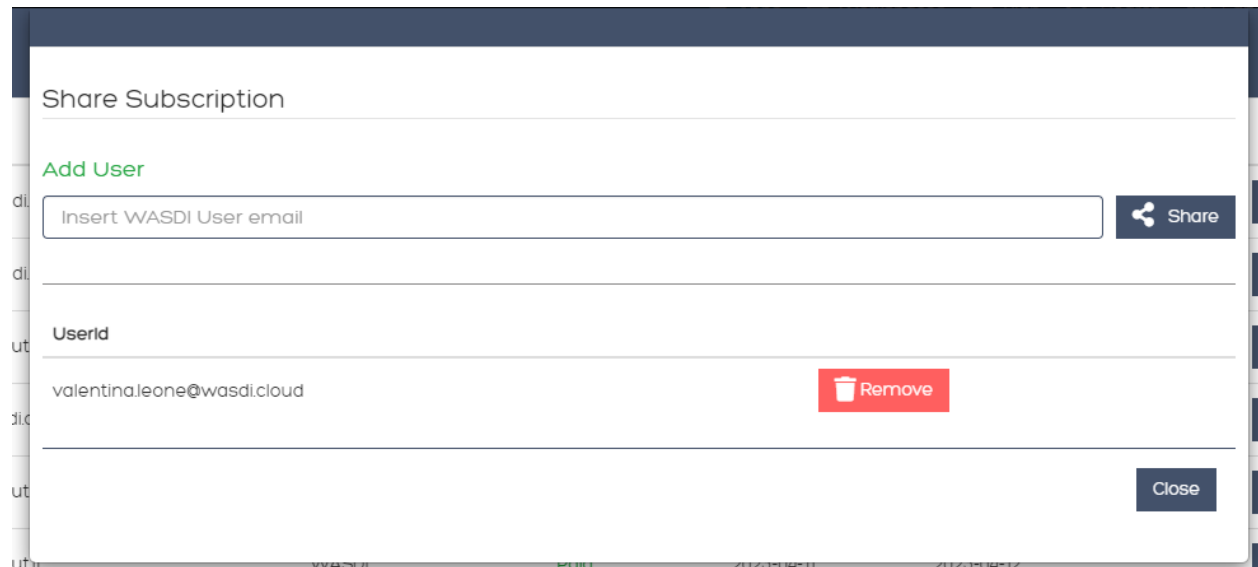


This action will open the Subscription Management Dialog where all your subscriptions (including expired subscriptions) are shown.

Name	Type	Owner ID	Organization	Payment Status	Start Date	End Date	Permission
One Year Standard	One Year Standard	betty.spurgeon@wasdi.cloud		Paid	2023-07-03	2024-07-03	owner   
One Day Standard(4)	One Day Standard	betty.spurgeon@wasdi.cloud		Pending	2023-07-04	2023-07-05	owner   
WASDI OYP	One Year Professional	p.campanella@fadeout.it	WASDI	Paid	2023-02-16	2024-02-16	shared by WASDI   
RIST OMP	One Month Professional	petru.petrescu@wasdi.cloud	RIST	Paid	2023-02-21	2023-03-21	shared by RIST   
uxtest	One Day Standard	p.campanella@fadeout.it	WASDI	Paid	2023-04-03	2023-04-04	shared by WASDI   
One Day Standard	One Day Standard	p.campanella@fadeout.it	WASDI	Paid	2023-04-11	2023-04-12	shared by WASDI   

In this dashboard you are able to:

View the users with whom the subscription is shared: by clicking the “Show Users” button, you will open a secondary dialog box that displays all the users who have access to this subscription. This sharing dialog behaves the same way that other sharing dialogs behave in WASDI.



The screenshot shows a 'Share Subscription' dialog box. At the top, it has the title 'Share Subscription'. Below the title is a green 'Add User' button. Underneath is a text input field with the placeholder 'Insert WASDI User email'. To the right of the input field is a blue 'Share' button with a share icon. Below the input field is a section titled 'Userid'. Inside this section, the email 'valentina.leone@wasdi.cloud' is listed. To the right of the email is a red 'Remove' button with a trash icon. At the bottom right of the dialog is a blue 'Close' button.

To view the Projects associated with the subscription, click the “Show Projects” button represented by a briefcase. This opens a secondary dialog where you can manage (create, edit, or delete) projects.

To Edit a subscription’s information, click the “Edit Subscription” button. This opens a secondary dialog where you can view read-only information (the acquisition date, start date, end date, and days remaining) and edit the subscription’s name, description, and attach an organisation.

Edit Subscription

One Year Standard

Name

One Year Standard

Description

Enter a Description (optional)

Acquisition Date

2023-07-03

Start Date

2023-07-03

End Date

2024-07-03

Days Remaining

359

Organization

No Organization

Cancel

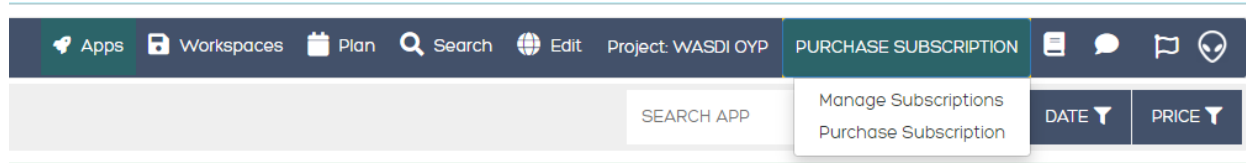
Save

You can also remove subscriptions, but be aware: you CANNOT remove subscriptions that are attached to organizations or have active projects.

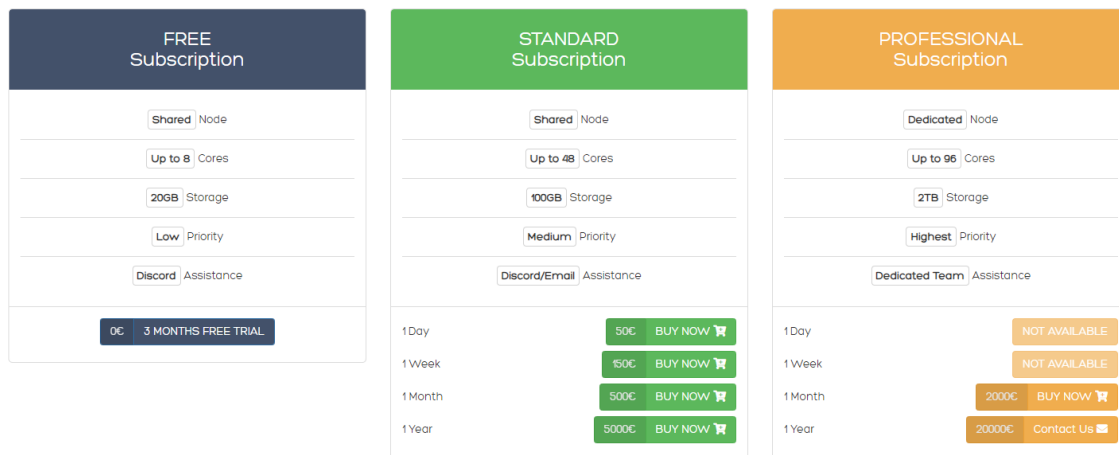
If you are not the owner of a subscription, by removing it, you are not deleting it - you are simply removing your permission to view it. Furthermore, if you are not the owner, information about the subscription is read-only and you can only edit or create projects for that subscription.

2.4.2 Purchasing Subscriptions

If you do not have a WASDI subscription and would like to purchase one - click the “PURCHASE SUBSCRIPTION” button in the navbar and select “Purchase Subscription” to be redirected to our purchasing page.



On the purchasing page you will be able to see the distinction between the various tiers of WASDI Subscriptions:



For this example, we will purchase a 1 Day Standard Subscription for 50 euros. By clicking “Buy Now” we open a dialog box where we can enter a name for this subscription. If we do not enter a name, a standard one will be provided for us (in this case the standard name will be “One Day Standard”).

Add Subscription

One Day Standard

Name


Description

Organization

No Organization ▾

Order Summary

Subtotal:	€ 50
Total:	€ 50

Cancel Checkout

When we click “Checkout” we will be re-directed our WASDI’s payment partner, Stripe. All payments are executed through Stripe. If you do not complete the payment right away, the Subscription information will be saved to your account and will be available through the “Manage Subscriptions” dialog and will be shown as “Pending...”

To complete your Stripe Payment enter your information to the Stripe payment channel.

Wasdi Sàrl

One Day Standard

€50.00

One Day Standard	€50.00
One day of standard usage of WASDI	
Subtotal	€50.00
VAT @	€6.90
Total due	€50.00

Powered by **stripe** | Legal





Contact information

Email

Payment method

☒ Card
 ☐ giropay
 ☐ EPS
 ☐ ID


Card information

1234 1234 1234 1234    

MM / YY CVC

Name on card

Billing address

Luxembourg 

Address line 1

Address line 2

Postal code City

☐ Securely save my information for 1-click checkout
 Pay faster on Wasdi Sàrl and everywhere Link is accepted.

☐ I'm purchasing as a business

☐ I agree to Wasdi Sàrl's [Terms of Service](#) and [Privacy Policy](#)

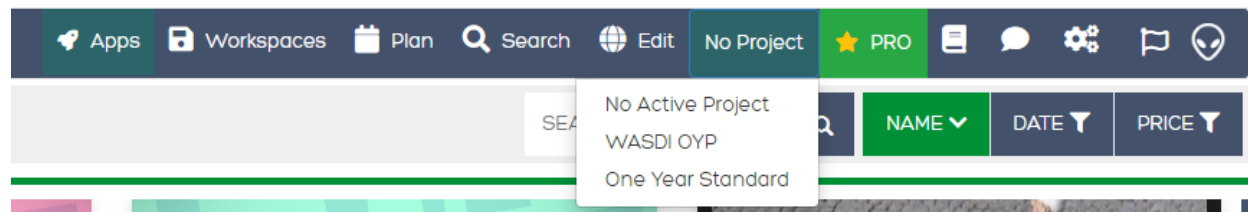
Pay

Once you have finalized your payment and it is confirmed by Stripe (this usually takes just a few moments) you will be re-directed to WASDI and if payment was successful, the “Payment Status” will read as “Paid”.

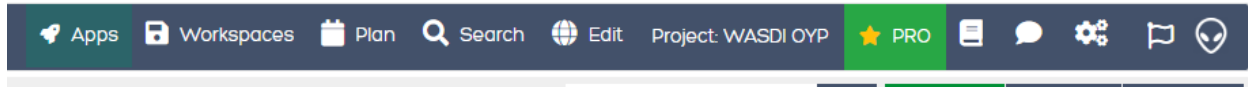
You are now set to use all features in WASDI!

2.4.3 Active Projects

To switch between projects in WASDI, use the projects tab in the navbar. When selected it will show a dropdown list of all the user's projects.



To switch between projects, simply select the project you wish to work with. If your selection is successful, that project's name will appear in your navbar and you will receive a notification that your selection was successful.



2.5 Other

2.5.1 Accessing Documentation

To access WASDI documentation directly from *www.WASDI.net* <<http://wasdi.net>> , select the documentation button in the navigation bar. Alternatively, you may also select the user profile dropdown (represented by the Alien head) and select “Docs” from the dropdown.


Both options will re-direct you to the WASDI documentation.

2.5.2 Sending Feedback

To send feedback or report bugs to the WASDI team, select the speech bubble icon in the navigation bar. This will open a dialog box where you will provide a title and message for the team.

Both input fields must be filled out in order to send the message. The message will be sent via email and the email address associated with you WASDI account will be included with the feedback.

Send feedback to WASDI as: Betty - betty.spurgeon@wasdi.cloud



Would you like to try our **live support**?
[Houston, we have a problem!](#)

Do you prefer to **write us**?

Title

Message:

Close

Send

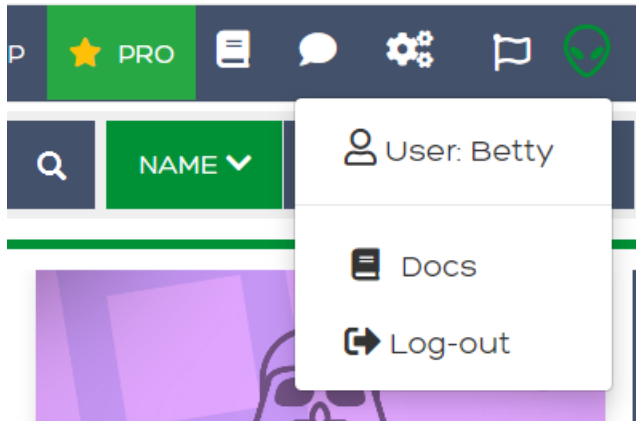
If your issue requires immediate assistance, you can click “Houston, we have a problem!” and this will open an invitation to WASDI’s discord server, where we offer live support.

To learn more about Discord, *please look to their documentation* <<https://discord.com/terms>>.

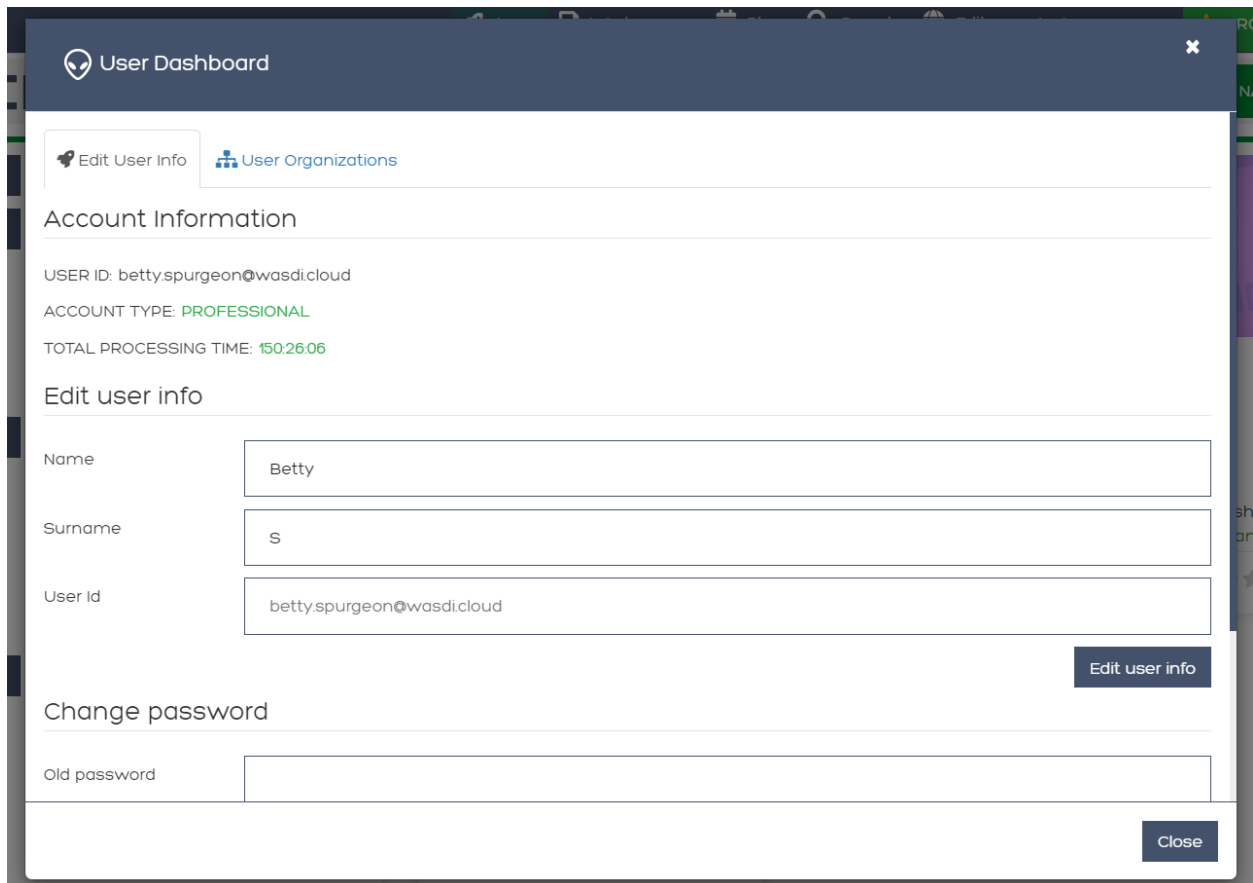
To participate in WASDI’s Discord Server, you must have a valid Discord account.

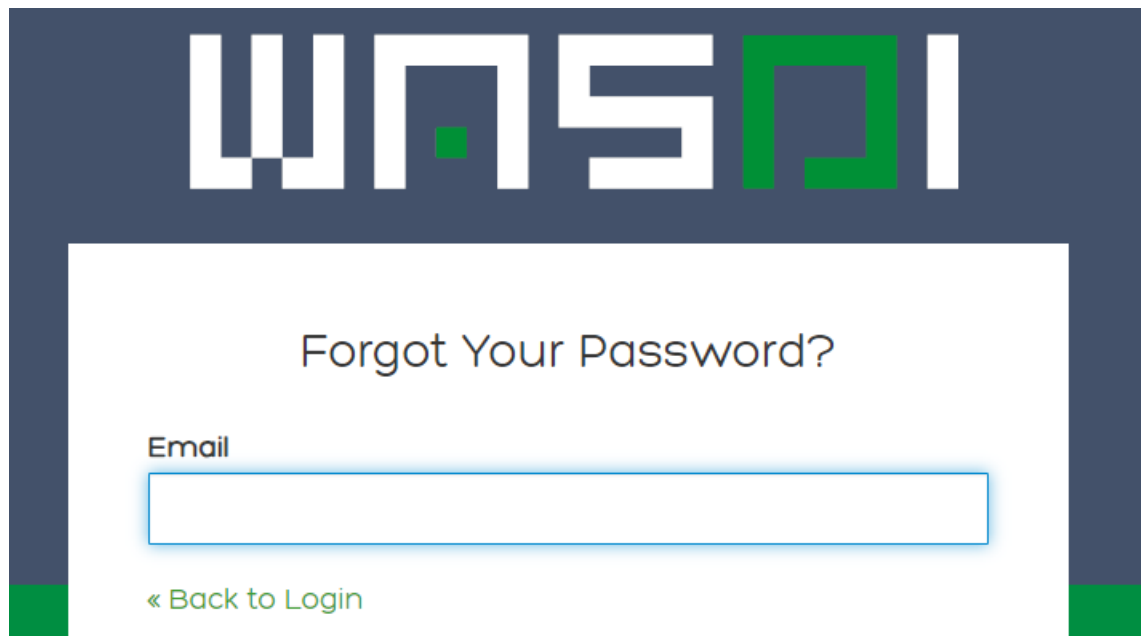
2.5.3 Account Management

To update your account settings, you can open your account management dashboard by selecting the alien icon and then your user profile (User: Name).



At this current time, it is more efficient to change your password through WASDI's authentication page, Keycloak.



The image shows a web form for password recovery. At the top, the WASDI logo is displayed in white on a dark blue background. Below the logo, the text "Forgot Your Password?" is centered. Underneath, there is a label "Email" followed by a text input field. Below the input field is a link "« Back to Login" in green. The entire form is enclosed in a dark blue border with green accents at the bottom corners.

WASDI

Forgot Your Password?

Email

[« Back to Login](#)

Submit

Enter your username or email address and
we will send you instructions on how to
create a new password.

You can also manage your organizations from this dashboard. For more information about Subscriptions and Organisations, please see the section on “Managing Subscriptions and Organisations”

User Dashboard

Edit User Info

User Organizations

Add Organization

Organization Name	Organization Owner	Actions
WASDI	p.campanella@fadeout.it	<div><div></div><div></div></div>
RIST	petru.petrescu@wasdicloud	<div><div></div><div></div></div>

Close

WASDI MARKETPLACE

All the WASDI Applications are available for end users' with a simple and intuitive Interface. Choose your App, set your input data with a few clicks and enjoy the result.

A good starting point to applications is the App store overview

3.1 Wasdi App Store

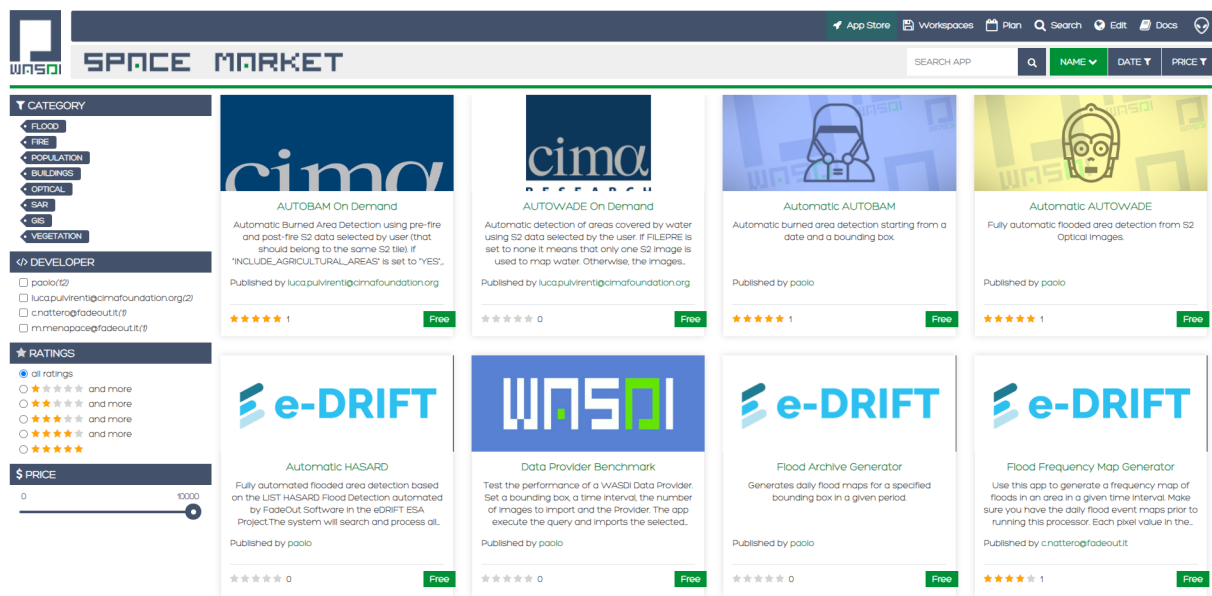
In this tutorial the WASDI app store will be introduced. The document will cover and highlight the main feature of this WASDI sections and also, will present how a WASDI application can be launched.

The app store concept is pretty common for mobile devices and our efforts was invested to develop the same user experience for WASDI. An user can upload, execute and share his own application directly on a web browser, with a fast and consistent user experience.

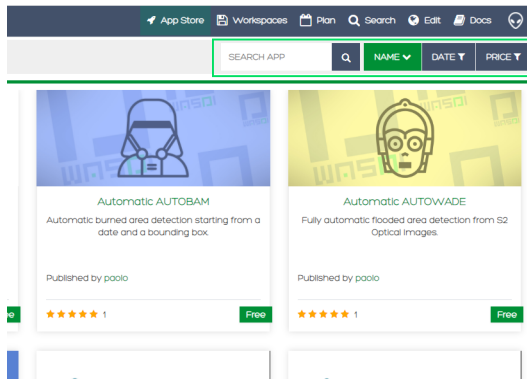
Also, a dedicated graphical interface(UI) can be added to the application, allowing to supply other users a well tailored experience.

3.1.1 Introduction

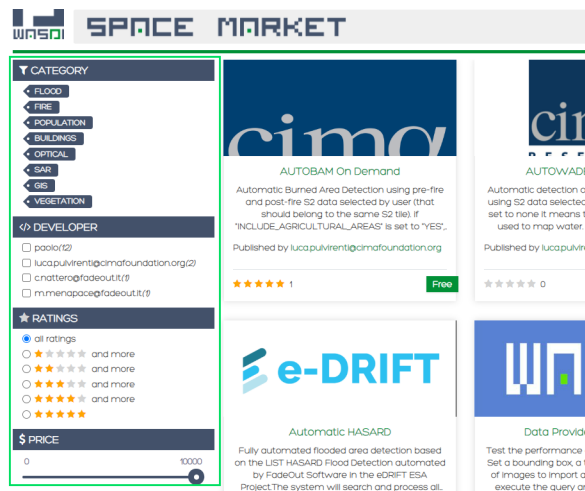
The main app store view consist of a list of entries reporting various WASDI applications:



It is possible to search the applications by using the following search field



Also, an user can enable some filters related to categories, developer, ratings and price

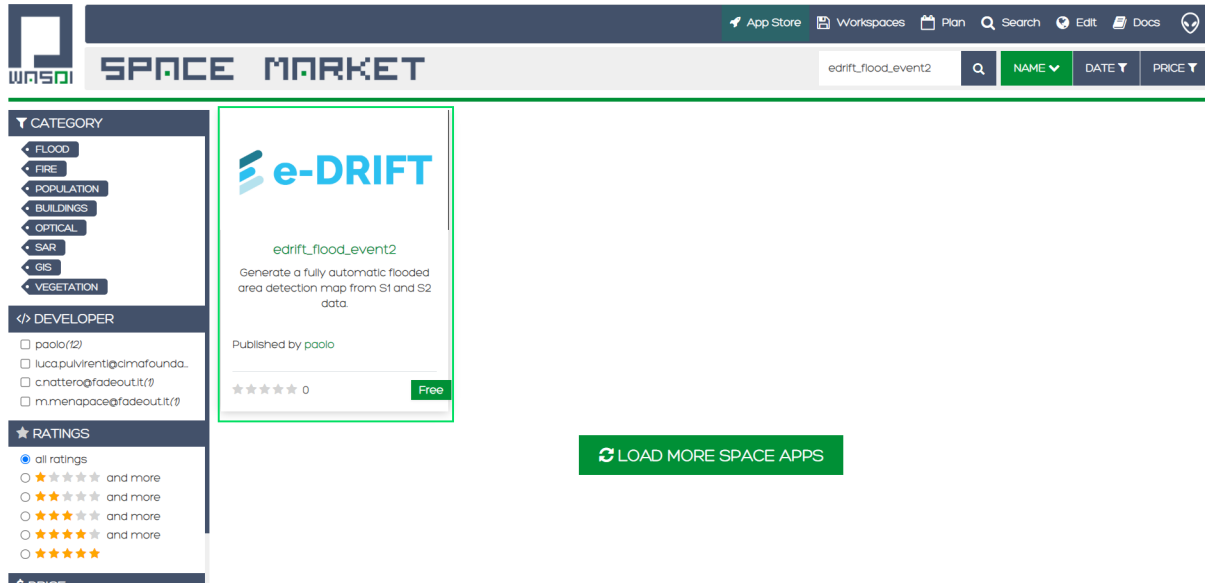


3.1.2 Launch an application

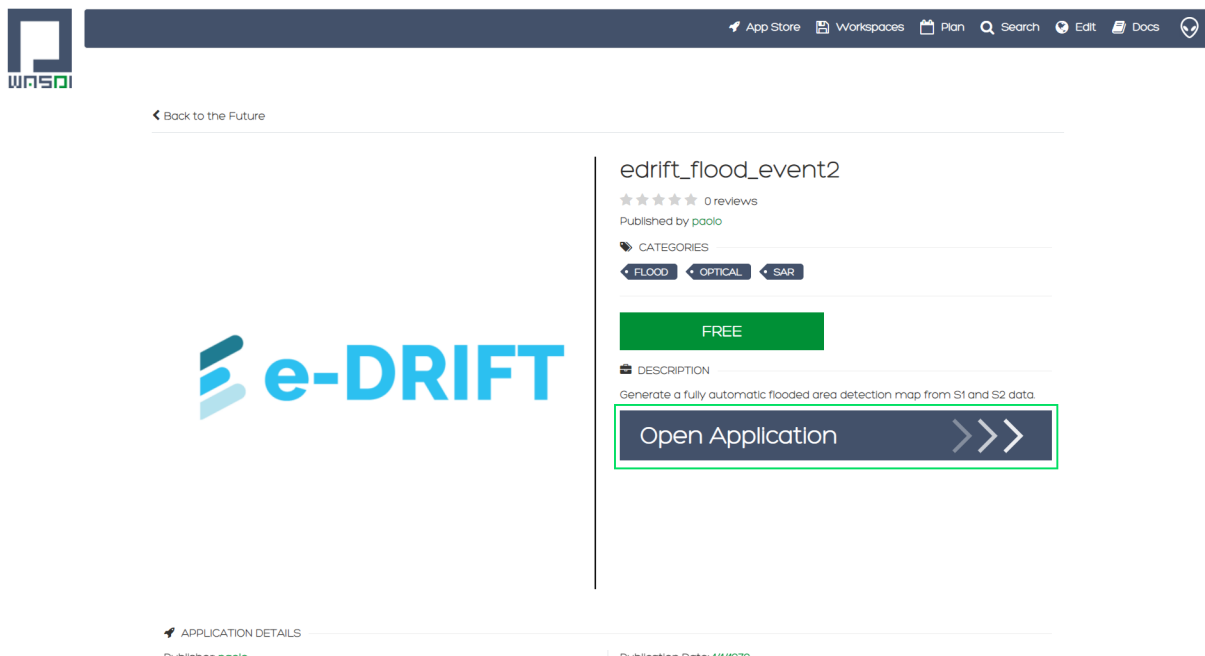
Let's try an application, in particular, search for **Automatic S1-S2 Floods**. This application can be used to identify flooded areas, using Sentinel-1 and Sentinel-2 products.

The application, at the lowest level of parameters specification, requires the date of the flood events and the area where the study must be done. WASDI, will then connect to dedicated servers (DIAS) to gather images and elaborates them to obtain the final product map.

Click on the **Automatic S1-S2 Floods** icon



and then on *Open Application*



The application UI it is shown. Several tabs are available to the user, allowing to customize the elaboration:

- **Advanced** allows to select the number of days before events to be evaluated, plus the selection of the Data Provider
- **SAR** give the possibility of a fine tuning of SAR(Syntethic Aperture Radar) parameters
- **Optical** tab deals with cloud coverage, setting a maximum percentage
- **GIS** allows to set parameters for the customization of final products
- **Help** reports a handle guide to this particular application
- **History** allow user to see previous run of the application

- **JSON** Show to the user the final JSON, which is a structured text format, that contains all parameters that will be used for the current app run.

For the sake of clarity only **Basic** tab will be used in this tutorial. More info on all the other parameters and a brief explanation of the app capabilities can be found in descriptions and in **Help tab**. This tutorial will analyze the outcome of a flood occurred on 11 May 2021 in Tajikistan and Afghanistan. In particular our analysis will consider imagery from the Khatlon Region of Tajikistan ([References](#))

From the first available field please select the date of the event and set it to 11/05/2021 (DD/MM/YYYY format). A text field value is used as default name for the output images.

As reported in Help section:

Event Code (BASENAME): Base Name of the output file. DO NOT USE “_” or “ ” or other special chars in the Base Name

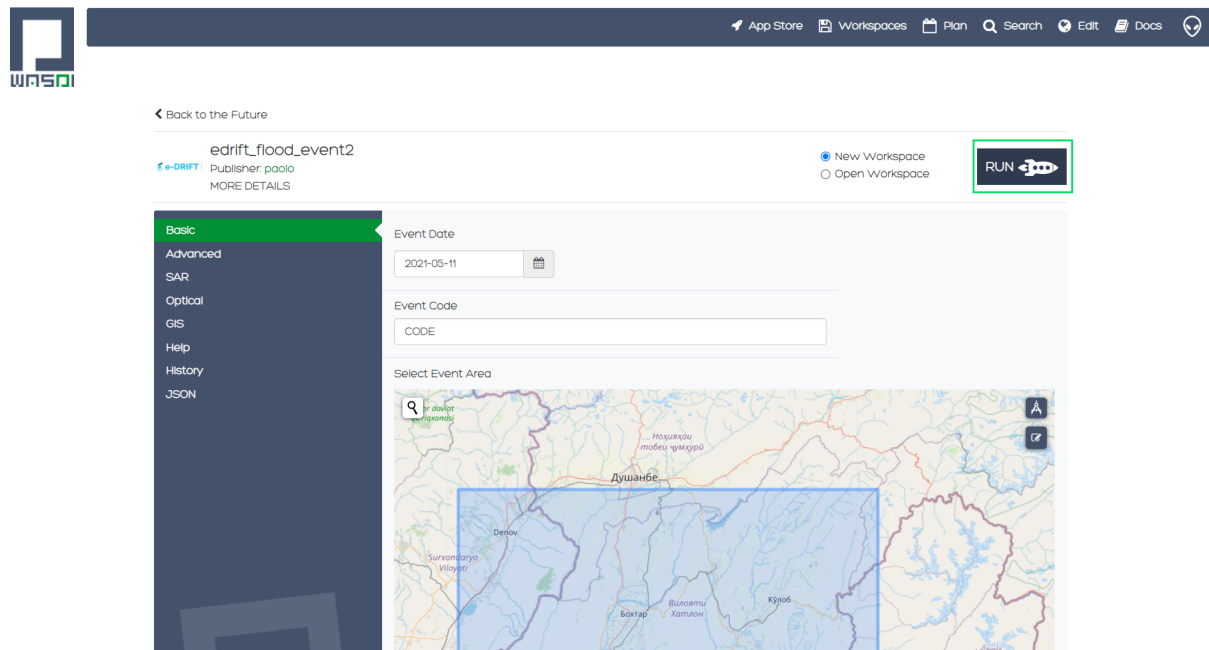
The output of the WASDI application will be several geoTiff images with the following naming convention:

- One layer for each day with a valid SAR Map, called [CODE]_[DATE]_flood
- SAR flood composite, sum of all days, called [CODE]_[DATE]_sar_flood_sum_days[TOT]
- Optical flood composite, sum of all tiles, called [CODE]_[DATE]_s2_flood
- Composite of SAR and Optical Flood, called [CODE]_[DATE]_flood_sum_days[TOT]

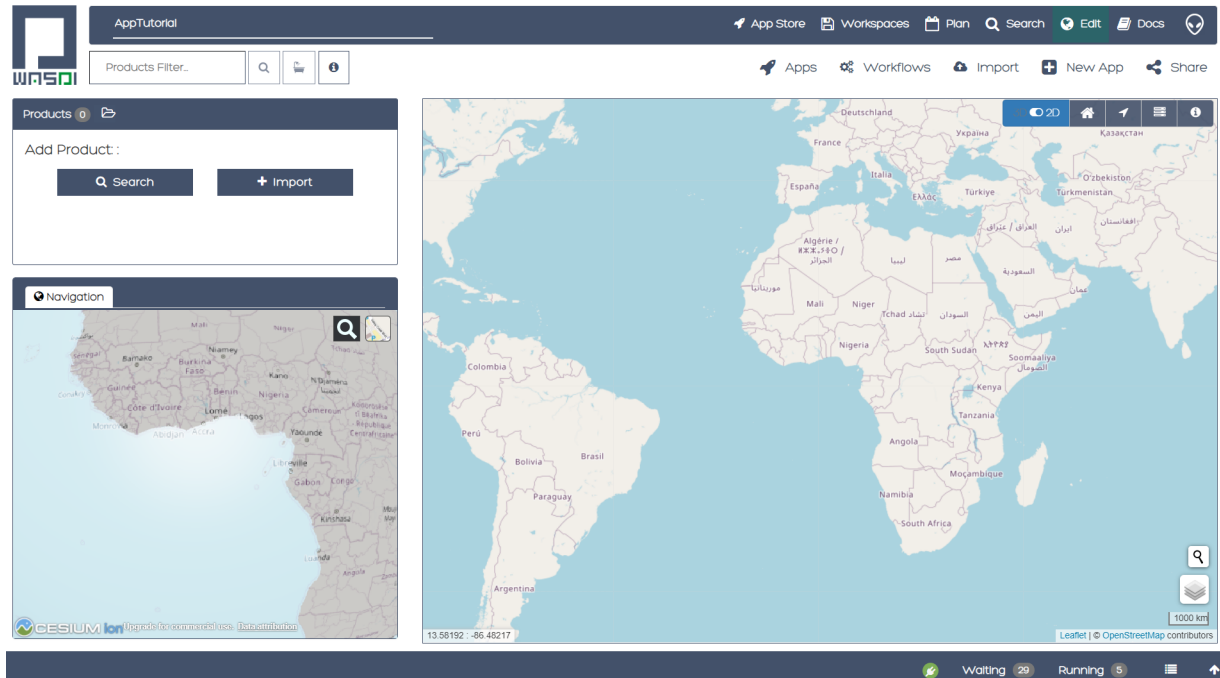
The single pixels in the result images uses the following value legend:

- **0** - No Data
- **1** - No Flood
- **2** - Permanent Water
- **3** - Flooded Areas

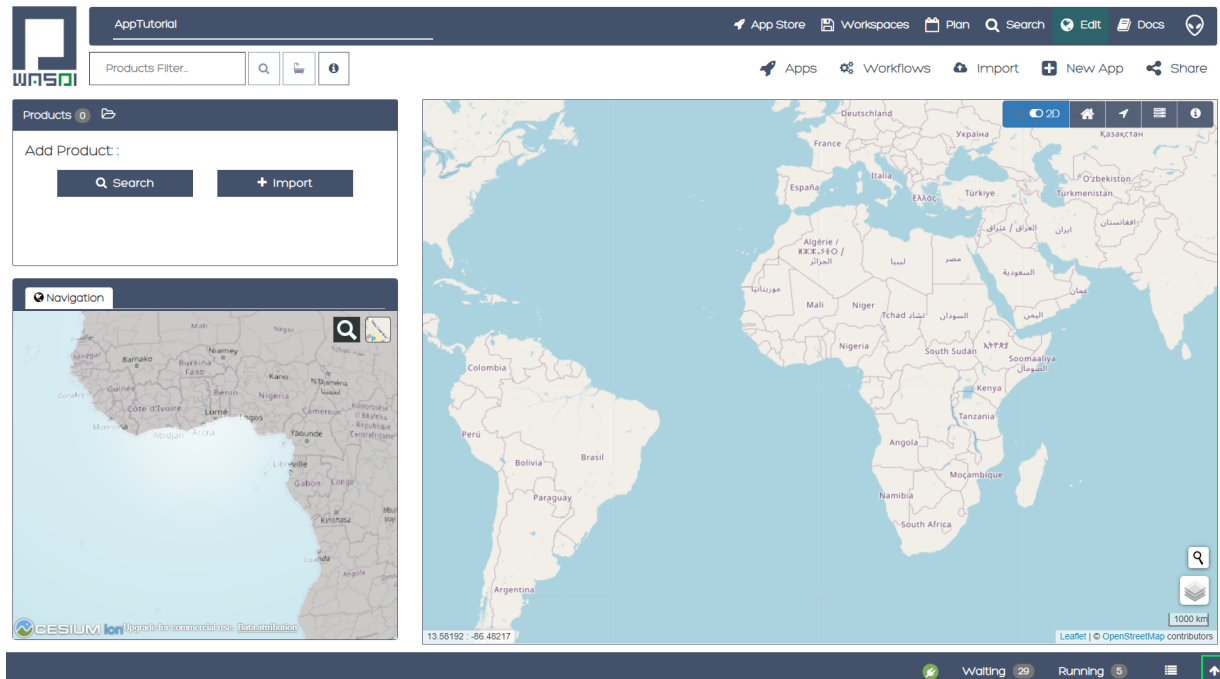
Select the area of the event and then click on “run” to start the processing.



After launching the elaboration the user will be redirected to the **edit view** of the newly generated workspace.



Here, the user can open the lower information bar, by clicking on the arrow in the lower right angle of the current perspective.



A panel with all the sub-operations is reported, showing the process status and the percentage of each single operation. In the following image there is reported the very early phases of the elaboration.

The screenshot shows the WASDI AppTutorial interface. At the top, there's a navigation bar with 'AppTutorial', 'App Store', 'Workspaces', 'Plan', 'Search', 'Edit', 'Docs', and a user profile icon. Below this, there's a 'Products Filter...' section with a search icon and a list icon. The main area is divided into two parts: on the left, a 'Products' section with an 'Add Product:' button and a '+ Import' button; on the right, a map of the Middle East region. Below the map, there's a table of operations.

Operation	Name	User	Size	Created	Started	Progress	Duration
App	mosaic_tile	m.menapace@fadeout.it		2021-05-24 12:09:59 Z	2021-05-24 12:10:03Z	<div></div>	00:00:04
App	optical_flood	m.menapace@fadeout.it		2021-05-24 12:09:51 Z	2021-05-24 12:09:57Z	<div></div>	00:00:10
App	edrift_archive_generator	m.menapace@fadeout.it		2021-05-24 12:09:50 Z	2021-05-24 12:09:55Z	<div></div>	00:00:12
App	edrift_flood_event2	m.menapace@fadeout.it		2021-05-24 12:09:35 Z	2021-05-24 12:09:39Z	<div></div>	00:00:28

At the bottom right of the table, there's a 'Load more...' link.

After the initial setup WASDI starts to fetch the required images. Each image is then added to the current workspace.

The screenshot shows the WASDI AppTutorial interface. At the top, there's a navigation bar with 'AppTutorial', 'Products Filter...', 'Search', 'List', and 'Info' icons. The main area is divided into two parts: on the left, a 'Products' section with a list of products; on the right, a map of the Middle East region.

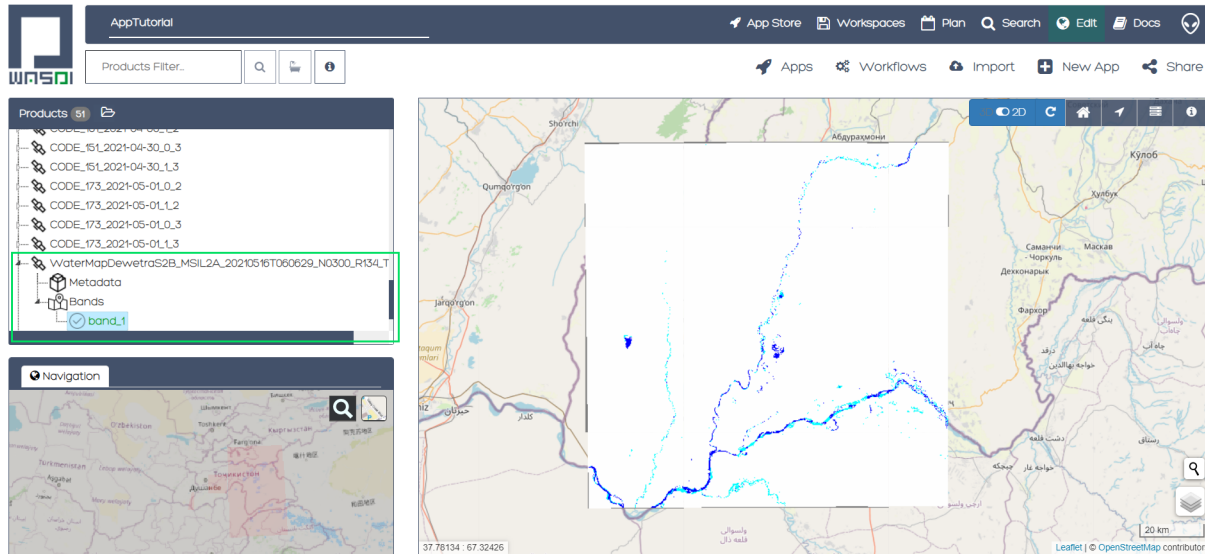
The 'Products' list contains the following items:

- S2B_MSIL2A_20210426T060629_N0300_R134_T42SUG_20210426T08
- S2B_MSIL2A_20210426T060629_N0300_R134_T42SUH_20210426T08
- S2A_MSIL2A_20210511T060631_N0300_R134_T42SUH_20210511T0813
- S2B_MSIL2A_20210426T060629_N0300_R134_T42SVH_20210426T08
- S2A_MSIL2A_20210511T060631_N0300_R134_T42SUG_20210511T0813
- S2B_MSIL2A_20210426T060629_N0300_R134_T42SVH_20210426T08
- S2B_MSIL2A_20210426T060629_N0300_R134_T42SVG_20210426T08
- S2B_MSIL2A_20210516T060629_N0300_R134_T42SVG_20210516T08C
- S2B_MSIL2A_20210509T061629_N0300_R034_T42SUG_20210509T08
- S2A_MSIL2A_20210519T055641_N0300_R091_T42SWG_20210519T08
- S2B_MSIL2A_20210429T061629_N0300_R034_T42SVH_20210429T08
- S2A_MSIL2A_20210514T061631_N0300_R034_T42SUG_20210514T082

Below the list, there's a 'Navigation' section with a search icon and a map of the Middle East region.

Selecting a result image from the product list it is possible to view the resulting GeoTiff image, geo-localized on the current map. In this case the image reports only permanent water belonging to the normal streams of rivers or from lake and sea.

Following the naming convention introduced beforehand, we can note that the current image is obtained from images acquired day before the flood event reported on the website.

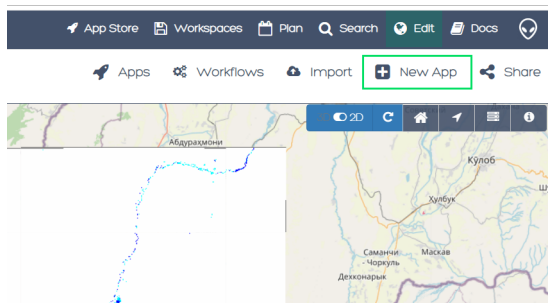


The geoTiff reported here above show in light blue the flooded areas and in dark blue the permanent waters(river streams). Congratulations for concluding your remote sensing analysis with WASDI!

3.1.3 Add a new application

We have seen how a deployed application can be launched and how products can be obtained.

For create a new application open a workspace and click on the icon **New App** from the bar.



A new dialog will be opened allowing the user to insert basic values like name and description of the new application

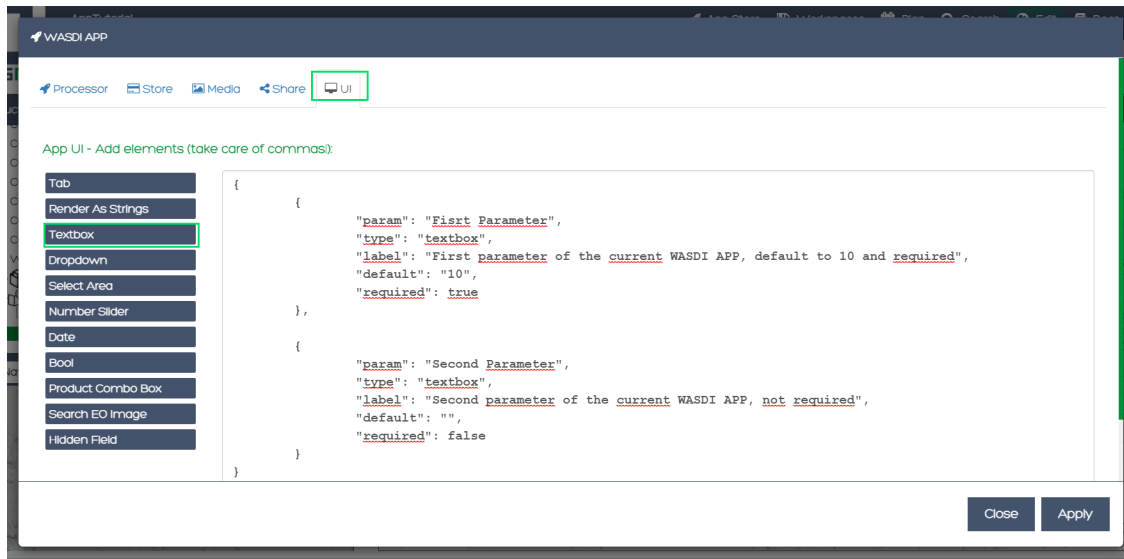
The user, that from now on we refer as the developer, can use several programming languages for the applications reported in the following image.

In order to upload the effective application a *.Zip* file, containing a set of files that must be prepared beforehand. Please refer to the particular WASDI libraries in order to acquire all the details about conventions and file format required. Note that it is possible to make the uploaded application available to all users of WASDI, by enabling the dedicated checkbox.

A great starting point as developer on WASDI is the python tutorial. Check it out !

A key factor of WASDI application is the possibility for the developer to create an user interface for the application, directly on the WASDI website. A JSON descriptor of the required fields can be edited to allow users to interact with canonical web widget.

Clicking on the User Interface(UI) tab it is possible to add such widgets by clicking to the corresponding buttons. The resulting JSON will be then parsed to check syntax coherence and, if the test is passed, used to generates UIs.



3.2 eDrift Tutorial

The flood-related algorithms available in WASDI support automated mapping of flood in:

- Open area
- Urban area

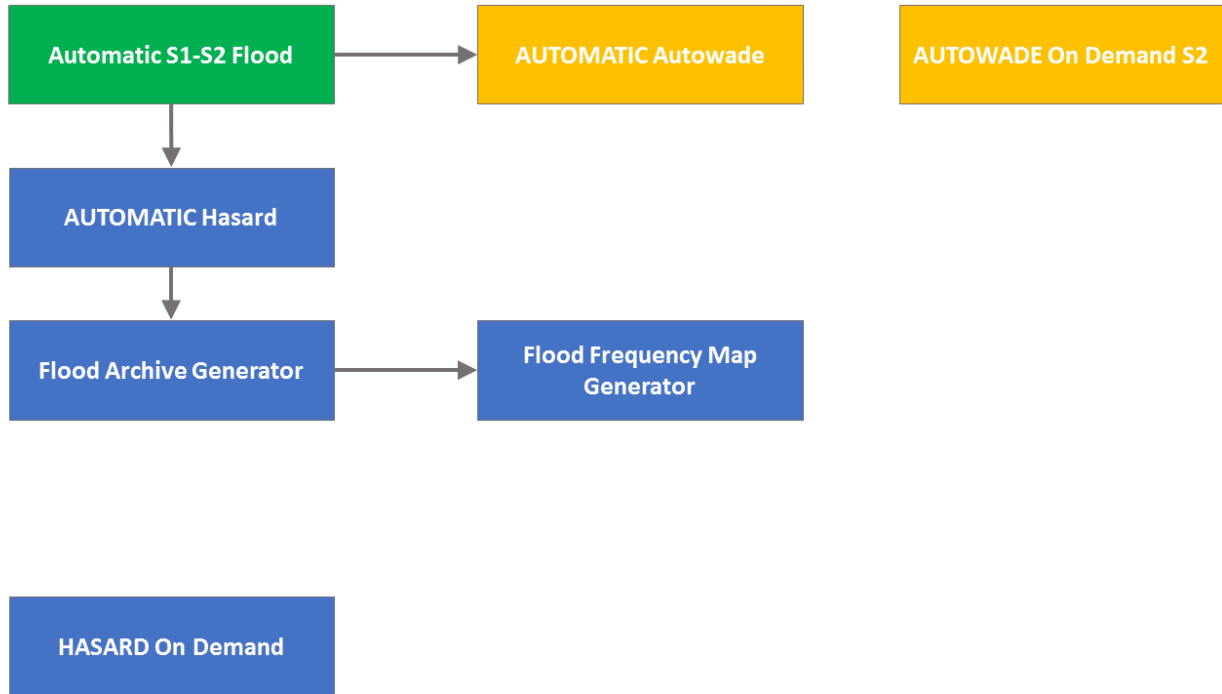
with:

- Sentinel-1, Sentinel-2 or a combination of the 2 missions
- VIIRS

3.2.1 Floods in open areas

In WASDI there are several applications available to map floods in open areas using either the Sentinel mission or the VIIRS sensor.

The scheme of Fig. 1 presents the algorithms for flood mapping in open area, using Sentinel-1 and/or Sentinel-2 and how they relate to each other. The algorithm to map flood from VIIRS sensor will be presented later and separately from the Sentinel missions.

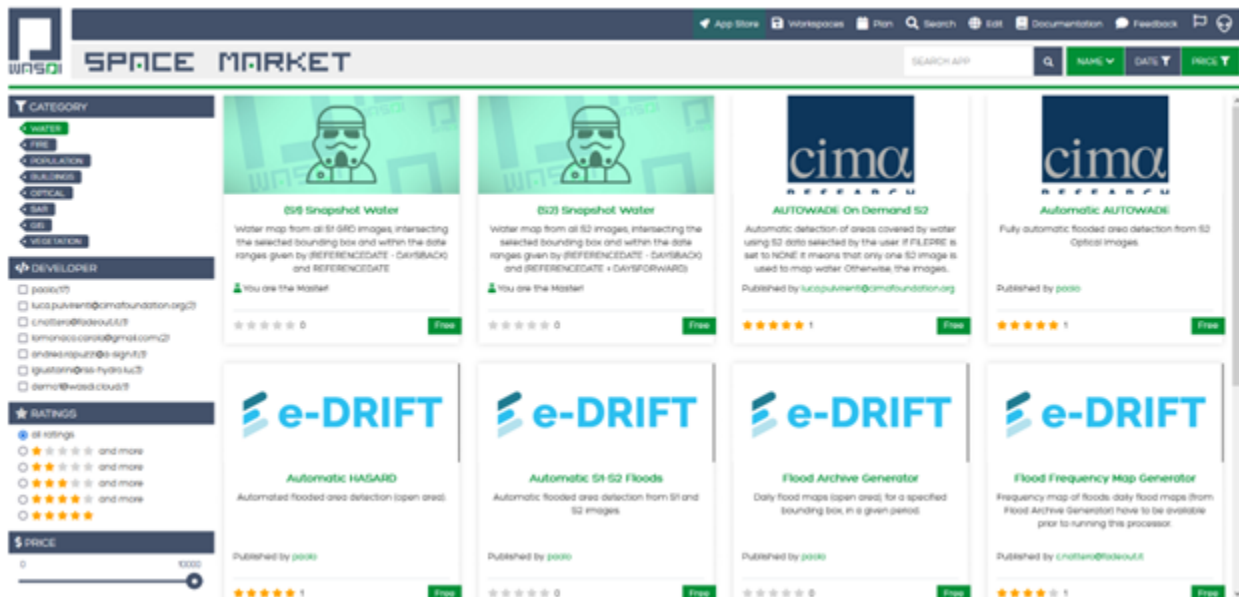


When landing in the WASDI marketplace, the user can select the filter “water” to subset only application related to water in general.

Excluding the apps designed to map a snapshot of water at a particular moment and permanent water (“(S1) Snapshot water” and “(S2) Snapshot water”),

there still remain a rather large number of apps to map floods.

This guideline should help the user select the best app for the specific need.



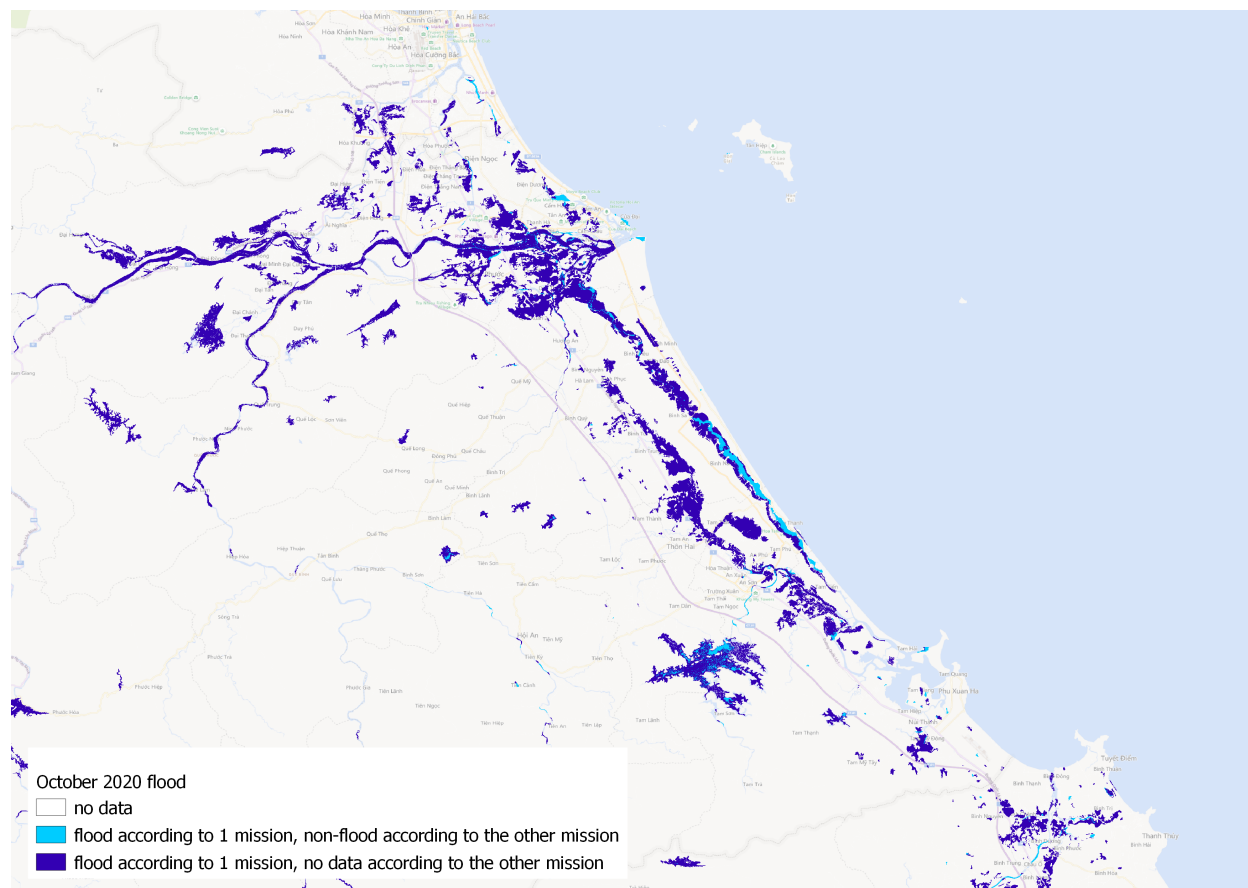
In case of flood map of open areas, the **most general app** is Automatic S1-S2 Floods.

The screenshot displays the WASDI SPACE MARKET interface. The top navigation bar includes links for App Store, Workspaces, Run, Search, Edit, Documentation, and Feedback. The main content area shows a grid of tools categorized by 'CATEGORY' (WATER, FIRE, POPULATION, BUILDINGS, OPTICAL, SAR, GIS, MISCELLANEOUS) and 'PRICE' (0 to 1000). The tools listed include:

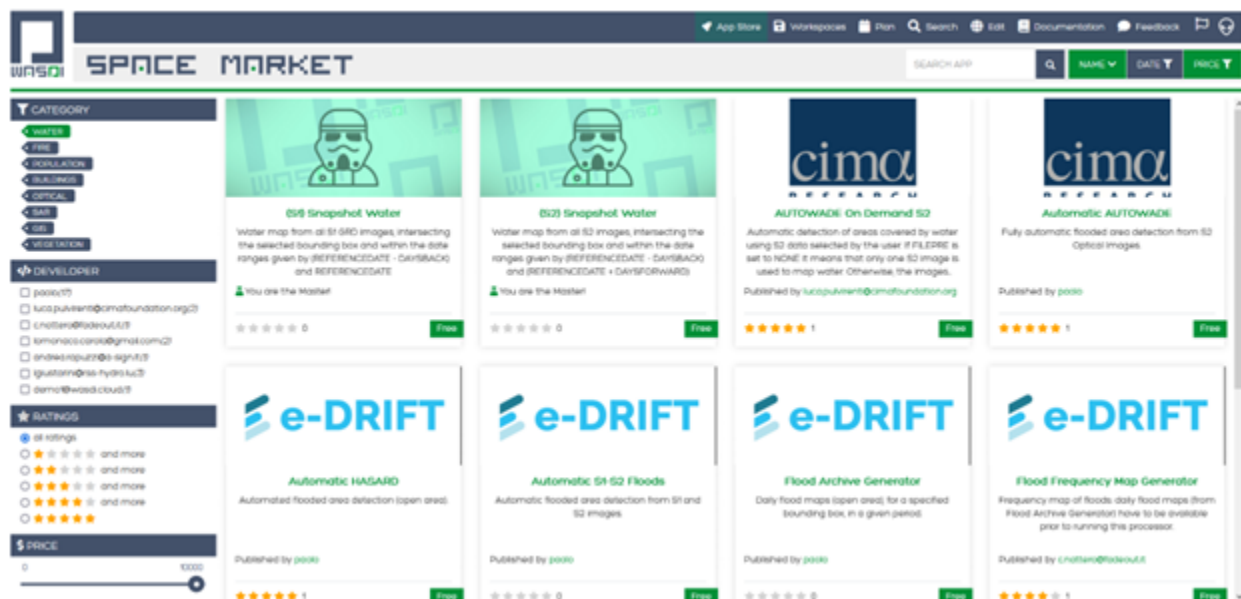
- S1-S2 Snapshot Water**: Water map from all S1 SAR images intersecting the selected bounding box and within the date ranges given by (REFERENCE DATE - DAYS BACK) and REFERENCE DATE. Published by paolo.
- S2-S2 Snapshot Water**: Water map from all S2 images intersecting the selected bounding box and within the date ranges given by (REFERENCE DATE - DAYS BACK) and (REFERENCE DATE + DAYS FORWARD). Published by paolo.
- AUTOWADE: On Demand S2**: Automatic detection of areas covered by water using S2 data selected by the user. If FLOODRE is set to NONE it means that only one S2 image is used to map water. Otherwise, the images are published by rucapulvent@amafoundation.org.
- Automatic AUTOWADE**: Fully automatic flooded area detection from S2 Optical images. Published by paolo.
- e-DRIFT Automatic HASARD**: Automated flooded area detection (open area). Published by paolo.
- e-DRIFT Automatic S1-S2 Floods**: Automatic flooded area detection from S1 and S2 images. Published by paolo.
- Flood Archive Generator**: Daily flood maps (open area) for a specified bounding box, in a given period. Published by paolo.
- Flood Frequency Map Generator**: Frequency map of floods daily flood maps (from Flood Archive Generator) have to be available prior to running this processor. Published by crottem@adevout.it.

Below the grid, the configuration page for 'Automatic S1-S2 Floods' is shown. It includes a 'New Workspace' button with a text input field 'Auto Name', an 'Open Workspace' button, and a 'RUN' button with a rocket icon. The configuration section has a sidebar with 'Basic', 'Advanced', 'SAR', 'Optical', 'GIS', 'Help', 'History', and 'JSON' tabs. The 'Basic' tab is active, showing fields for 'Date of the flood' (with a 'Select Date' button), 'Code of the file' (with a 'CODE' input field), and 'Select area of the flood' (with a map of Africa and the Middle East).

Automatic S1-S2 Floods generates a fully automatic flood map, in open areas, from **Sentinel-1** and **Sentinel-2** images.



In practice, Automatic S1-S2 Floods calls 2 other apps available in WASDI, namely: #. Automatic AUTOWADE #. Automatic HASARD



Automatic AUTOWADE performs flood area detection in open areas from Sentinel-2 optical images, while Automatic HASARD executes flood area detection in open areas from Sentinel-1 SAR images. Both apps have their corresponding On Demand version. The difference between the automatic version and the On Demand version is:

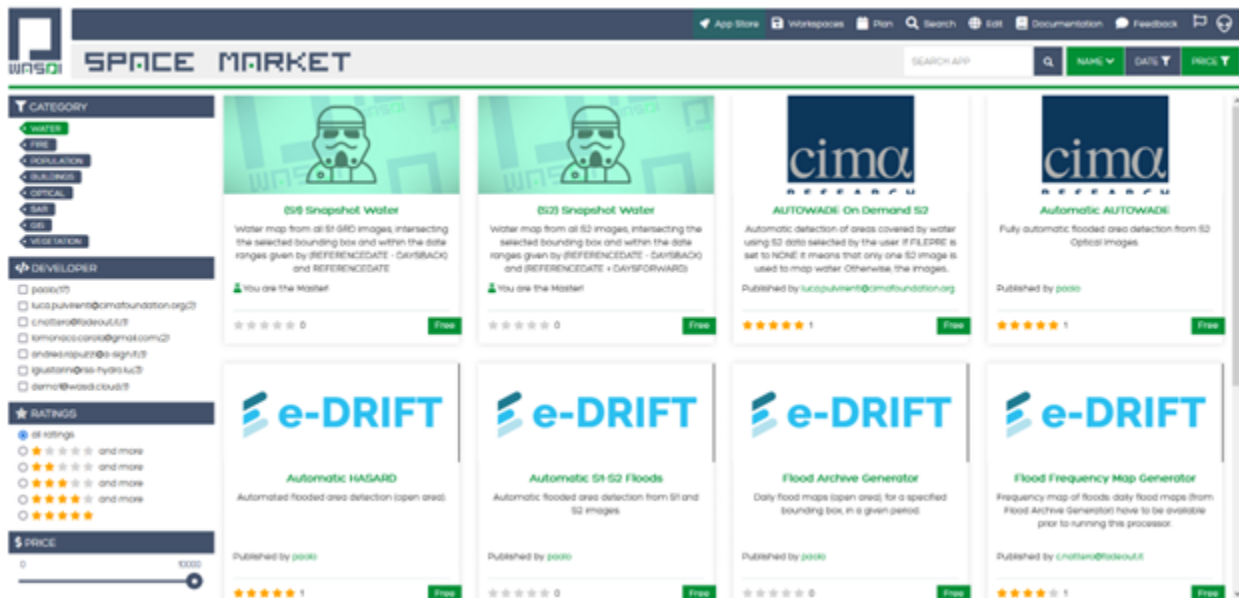
- Automatic: this version automatically identifies the pairs of pre-flood and post-flood images.
- On Demand: this version uses Sentinel images selected by the user, according to criteria defined in the help section and such images have to be imported manually in the workspace where the analysis is run.

We analyze here first the algorithms available to map floods in open areas with Sentinel-1 and then those to map floods in open areas with Sentinel-2.

3.2.2 Algorithms to map floods in open areas from Sentinel-1

Automatic HASARD

Automatic HASARD can be used to map floods, in open area, in a given region and for a certain date.



Automatic HASARD actually calls one more app, namely Flood Archive Generator. The difference between Automatic HASARD and Flood Archive Generator is that the second one is used to generate daily flood maps in the time range specified by the user. On the other hand, Automatic HASARD calls the Flood Archive Generator to create daily flood map in the time range around the date specified by the user. By default it considers 15 days after the date specified by the user and 15 days before the date specified by the user. The reason behind this is that when using Automatic HASARD, the date of the flood might not be entirely clear. In fact, the daily maps will help narrow down the day of the largest extent, supporting also monitoring the evolution of the flood around the date selected by the user. Besides the daily maps of flood, Automatic HASARD will also produce a final composite map that represents the cumulative flood of all the daily flood maps.

< Back to the Future

Automatic HASARD

f e-DRIFT Publisher: poolo [Edit](#)

[MORE DETAILS](#)

☒ New Workspace

☐ Open Workspace

[RUN](#)

Basic

Advanced

GIS Related

Help

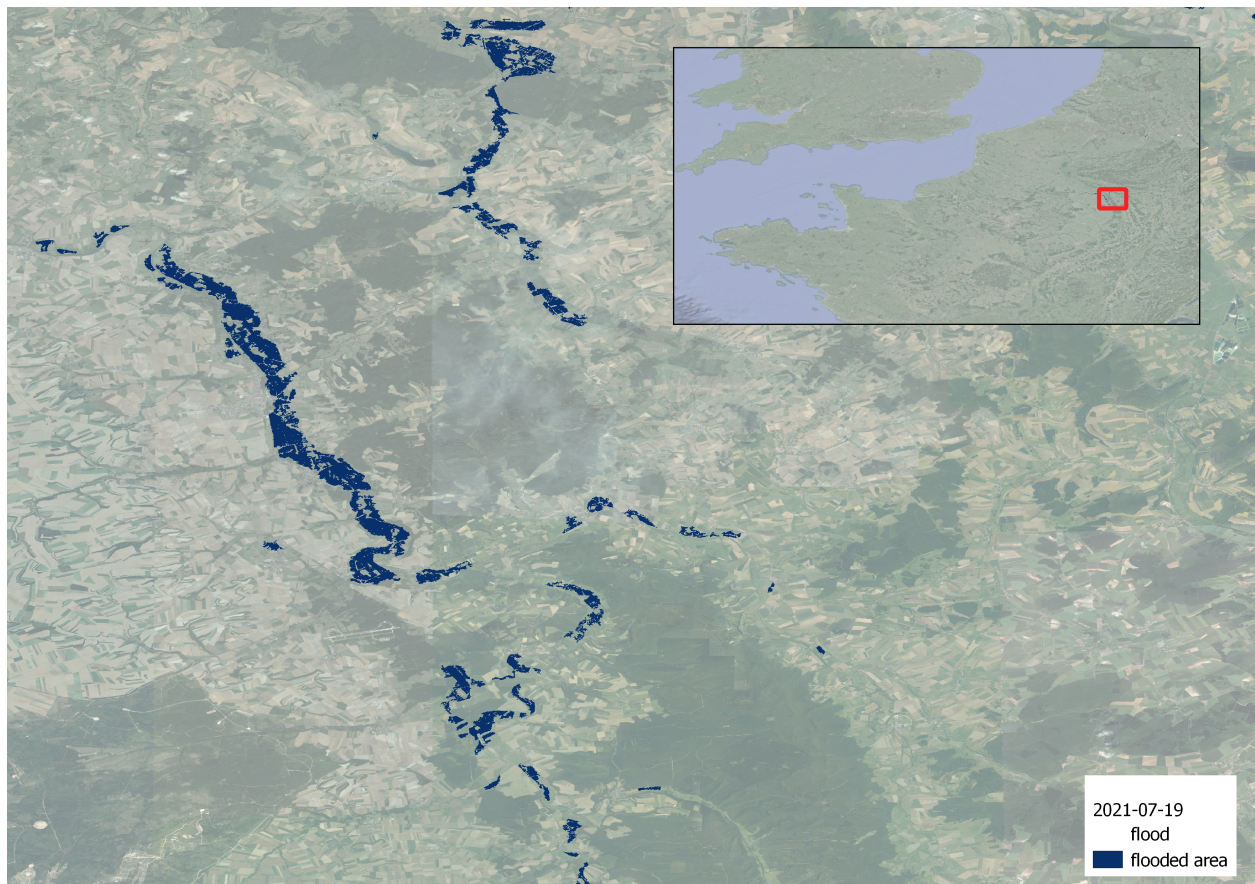
History

JSON

Date of the flood

Code of the flood

Select area of the flood



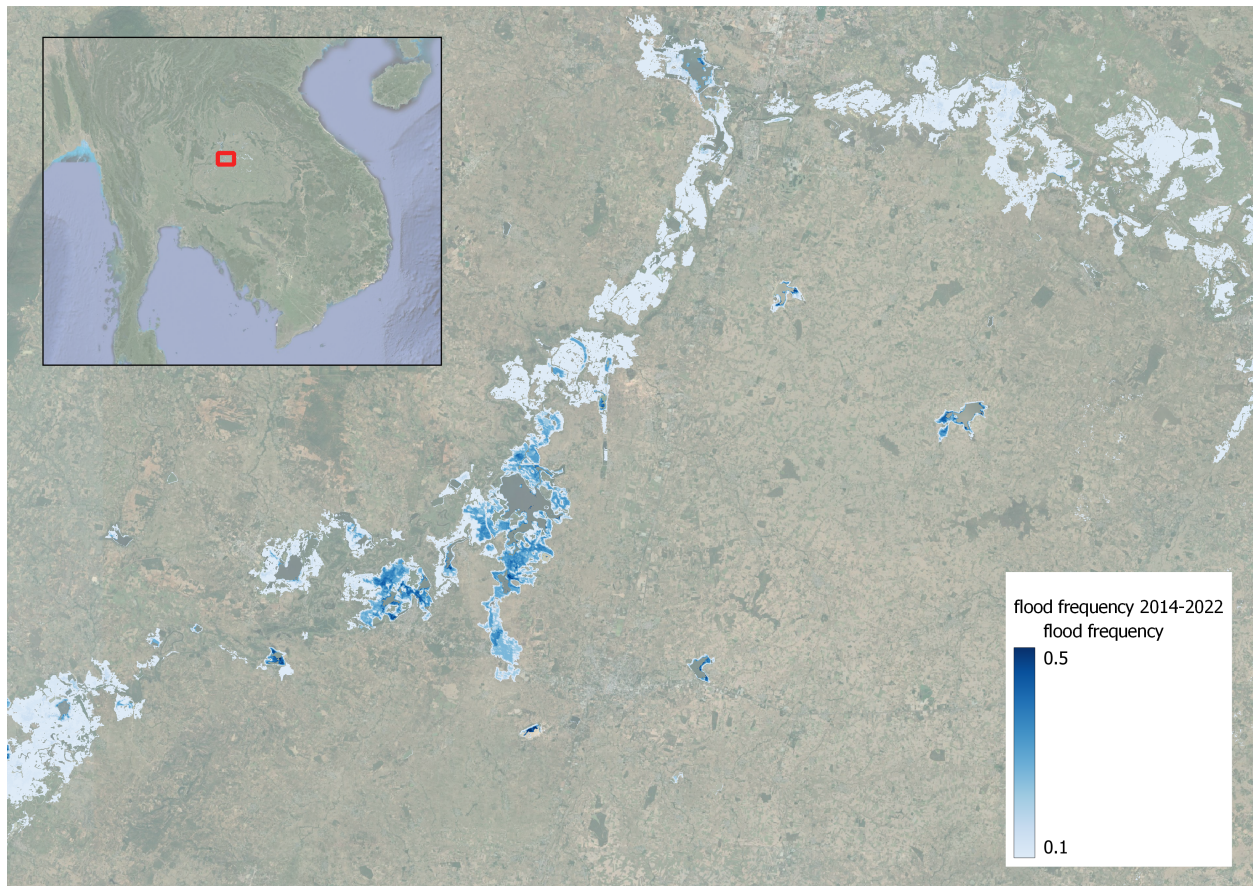
The screenshot displays the WASDI SPACE MARKET interface. At the top, there's a navigation bar with links to App Store, Workspaces, Run, Search, Edit, Documentation, and Feedback. Below this is a search bar and filters for NAME, DATE, and PRICE. The main content area is a grid of tool cards. On the left, there's a sidebar with filters for CATEGORY (WATER, FIRE, POPULATION, BUILDINGS, CRITICAL, DISEASE, DISASTERS), DEVELOPER, and RATINGS. The tool cards include:

- S1/S2 Snapshot Water** (two versions): Water map from S1/S2 images. Published by You are the Master!
- AUTOWADE: On Demand S2**: Automatic detection of areas covered by water. Published by lucapulverent@amafoundation.org.
- Automatic AUTOWADE**: Fully automatic flooded area detection. Published by pezo.
- e-DRIFT Automatic HAZARD**: Automated flooded area detection. Published by pezo.
- e-DRIFT Automatic S1/S2 Floods**: Automatic flooded area detection. Published by pezo.
- Flood Archive Generator**: Daily flood maps. Published by pezo.
- Flood Frequency Map Generator**: Frequency map of floods. Published by c.nattero@fadeout.it.

Below the grid, the detailed view of the **Flood Frequency Map Generator** tool is shown. It includes a sidebar with navigation links (Basic parameters, Advanced parameters, GIS, Diagnostics, Help, History, JSON) and a main form with the following fields:

- Input files prefix**: A text input field.
- Start date**: A date picker with a calendar icon.
- End date**: A date picker with a calendar icon, with a note: "End date (picks the last date in the workspace, if not specified)".
- Select area of the flood**: A map of Africa and the Middle East with a search bar and a selection tool.

At the top right of the detailed view, there are buttons for "New Workspace" (with a text input field labeled "[Auto Name]"), "Open Workspace", and a "RUN" button with a rocket icon.



HASARD On Demand

HASARD On Demand generates a flooded area map, in open area, using 2 Sentinel-1 images, one pre and one post the flood, with the same geometry.

Use this app when fairly certain of the date of the flood and when the 2 Sentinel-1 images have already been pre-processed from S1 GRD images and saved as .tif files.

SPACE MARKET

SEARCH APP

NAME DATE PRICE

CATEGORY

- WASDI
- GIS
- POPULATION
- BUILDINGS
- DATA
- OS
- VISUALIZATION

DEVELOPER

- poolo
- lucaspulvent@comatbundson.org
- crnntm@fideout.it
- lomonaco.pasid@gmail.com
- andreas.pulvent@fideout.it
- lguistin@fideout.it
- demot@fideout.it

RATINGS

- all ratings
- 1 star and more
- 2 stars and more
- 3 stars and more
- 4 stars and more
- 5 stars and more

PRICE

0 10000

Automatic HASARD

Automated flooded area detection (open area)

Published by poolo

★★★★★ 1

Free

Automatic S1-S2 Floods

Automatic flooded area detection from S1 and S2 images

Published by poolo

★★★★★ 0

Free

Flood Archive Generator

Daily flood maps (open area), for a specified bounding box, in a given period

Published by poolo

★★★★★ 0

Free

Flood Frequency Map Generator

Frequency map of floods, daily flood maps (from Flood Archive Generator) have to be available prior to running this processor

Published by crnntm@fideout.it

★★★★★ 1

Free

HASARD On Demand

Flood detection (open area) using 2 SAR images, one pre and one post the flood, with the same geometry

Published by poolo

★★★★★ 0

Free

Urban Flood

Fully automated flooded area detection in urban areas

Published by poolo

★★★★★ 1

Free

VIRIS Flood

VIRIS Flood

Published by poolo

★★★★★ 0

Free

LOAD MORE SPACE APPS

Back to the Future

HASARD On Demand

Publisher: poolo [Edit](#)

MORE DETAILS

☒ New Workspace

☐ Open Workspace

[RUN](#)

Basic

Advanced

Help

History

JSON

Pre-flood Image

Flood Image

Mask output file (tif format)

Name of the output file (include .tif extension)

Parameters

All these apps, working with Sentinel-1 images, share a few parameters, whose meaning and range of possible values is here discussed.

HSBA Depth

This is the Hierarchical Split Based Approach (HSBA) Depth parameter as defined in Chini et al. (2017). Its default value of -1 means that the algorithm starts from the entire S1 scene and then, if it did not find any bimodality in the histogram of the entire scene itself, it will split the entire S1 scene into 4 tiles and check each of them for bimodality in the histogram distribution of each of the 4 tiles. In case it finds bimodality in the histogram of one or more of the 4 tiles, it keeps, out of the 4 tiles, those that are bimodal, while it keeps splitting again in 4 tiles the tiles whose histogram is not bimodal.

In case this value is changed to, for instance, 2, this means that the algorithm will not check if the entire S1 image has a bimodal histogram. It will also not check if the 4 tiles in which the entire S1 scene can be split are bimodal. It will go straight to check if the 16 tiles in which the entire S1 image can be split are bimodal. This shortens the processing time and should be used only when the user is fairly certain that the flood represents only a small portion of the entire S1 scene.

Ashman Coefficient (no units)

The default value of 2.4 is general, while a higher value (e.g. 2.7) can be selected to better separate the 2 distributions

Minimum value (pixels) for bimodal identification

This parameter represents the minimum number (in pixels) that a sub-tile should have to stop further splitting. A smaller value, like 1,000 pixels, is suggested for small floods, like those that typically happen in Europe, while a larger flood, like 10,000 pixels, is more appropriate for vast events that can be observed in Asia or in North America.

**Minimum blob size (pixels) **

This parameter is used in post-processing to remove small clusters of pixels that were identified as flood but that most likely are going to be noise and/or misclassification. A smaller value, like 10 pixels, is suggested for small floods, like those that typically happen in Europe, while a larger flood, like 150 pixels, is more appropriate for vast events that can be observed in Asia or in North America.

REFERENCES

- M. Chini, R. Hostache, L. Giustarini and P. Matgen, "A Hierarchical Split-Based Approach for Parametric Thresholding of SAR Images: Flood Inundation as a Test Case," in *IEEE Transactions on Geoscience and Remote Sensing*, vol. 55, no. 12, pp. 6975-6988, Dec. 2017, doi: 10.1109/TGRS.2017.2737664.
- Chini, Marco, Ramona Pelich, Luca Pulvirenti, Nazzareno Pierdicca, Renaud Hostache, and Patrick Matgen. 2019. "Sentinel-1 InSAR Coherence to Detect Floodwater in Urban Areas: Houston and Hurricane Harvey as A Test Case" *Remote Sensing* 11, no. 2: 107. <https://doi.org/10.3390/rs11020107>

3.2.3 Algorithms to map floods in open areas from Sentinel-2

Automatic AUTOWADE

Automatic AUTOWADE can be used to map floods, in open area, in a given region and for a certain date. It will search for Sentinel-2 images pre and post flood, try to detect the flooded areas for each pair of images and then it will mosaic the final result. All the single output maps and the final mosaic will be added to the workspace.

The screenshot displays the WASDI SPACE MARKET interface. The top navigation bar includes links for App Store, Workspaces, Run, Search, Edit, Documentation, and Feedback. The main header features the WASDI logo and the text "SPACE MARKET". A search bar and filters for NAME, DATE, and PRICE are also present.

On the left, a sidebar provides filtering options by CATEGORY (WATER, FIRE, POLLUTION, BUILDING, CRITICAL, SAN, DIS, and other factors), DEVELOPER (listing various email addresses), RATINGS (all ratings, 1 star and more, 2 stars and more, 3 stars and more, 4 stars and more, 5 stars and more), and PRICE (a slider from 0 to 1000).

The main content area shows a grid of software products:

- 059 Snapshot Water**: Water map from all S1 S2 images intersecting the selected bounding box and within the date ranges given by (REFERENCEDATE - DAYSBACK) and REFERENCEDATE. Published by You are the Master! (Free).
- 052 Snapshot Water**: Water map from all S2 images intersecting the selected bounding box and within the date ranges given by (REFERENCEDATE - DAYSBACK) and (REFERENCEDATE + DAYSFORWARD). Published by You are the Master! (Free).
- AUTOWADE: On Demand S2**: Automatic detection of areas covered by water using S2 data selected by the user. If FLOOD is set to NONE it means that only one S2 image is used to map water. Otherwise, the images. Published by lucapulvent@amafoundation.org (Free, 1 star).
- Automatic AUTOWADE**: Fully automatic flooded area detection from S2 Optical images. Published by poolo (Free, 1 star).
- e-DRIFT Automatic HAZARD**: Automated flooded area detection (open area). Published by poolo (Free, 1 star).
- e-DRIFT Automatic S1-S2 Floods**: Automatic flooded area detection from S1 and S2 images. Published by poolo (Free).
- Flood Archive Generator**: Daily flood maps (open area) for a specified bounding box in a given period. Published by poolo (Free).
- Flood Frequency Map Generator**: Frequency map of floods daily flood maps (from Flood Archive Generator) have to be available prior to running this processor. Published by crotten@floodout (Free, 1 star).

Below the grid, a "Back to the Future" link is visible.

The detailed view of the **Automatic AUTOWADE** product shows the publisher as poolo. It includes buttons for "New Workspace" (with a text input field for "Auto Name") and "Open Workspace", and a "RUN" button with a rocket icon.

The configuration panel on the left has tabs for Basic, Advanced, Help, History, and JSON. The Basic tab is active, showing the following settings:

- Event Code**: EV
- Event Date**: Select Date (with a calendar icon)
- Cloud Coverage**: A slider set to 50, ranging from 0 to 100.
- Min days distance of Pre image from Post image**: A slider set to 0, ranging from 0 to 100.
- Select Event Area**: A map of North Africa and the Middle East with a search icon.

AUTOWADE On Demand S2

AUTOWADE On Demand S2 generates a flooded area map, in open area, using 2 Sentinel-2 images, one pre and one post the flood, belonging to the same Sentinel-2 tile.

Use this app when fairly certain of the date of the flood and when the 2 Sentinel-2 images have already been imported into the workspace. It can also work with only 1 Sentinel-2 image, which needs to be the one post the flood.

The screenshot displays the SPACE MARKET application interface. At the top, there's a navigation bar with 'App Store', 'Workspaces', 'Run', 'Search', 'Edit', 'Documentation', and 'Feedback'. Below this is a search bar and filters for 'NAME', 'DATE', and 'PRICE'. The main area shows a grid of app cards. The selected app, 'AUTOWADE On Demand S2', is highlighted. It is published by 'luca.pulvirenti@cimafoundation.org' and is free. The app description states: 'Automatic detection of areas covered by water using S2 data selected by the user. If FILEDRE is set to NONE it means that only one S2 image is used to map water. Otherwise, the images are used to map water. Otherwise, the images are used to map water.' The app is rated 1 star. Below the app grid, there's a 'Back to the Future' button. The bottom section shows the app's details, including the publisher 'luca.pulvirenti@cimafoundation.org' and a 'MORE DETAILS' link. On the right, there are buttons for 'New Workspace' (with a text input field 'Auto Name') and 'Open Workspace'. A 'RUN' button with a rocket icon is also present. The left sidebar contains a menu with 'Input', 'Help', 'History', and 'JSON'. The main content area shows the 'Input' section with two dropdown menus: 'Pre Flood Image' and 'Post Flood Image'.

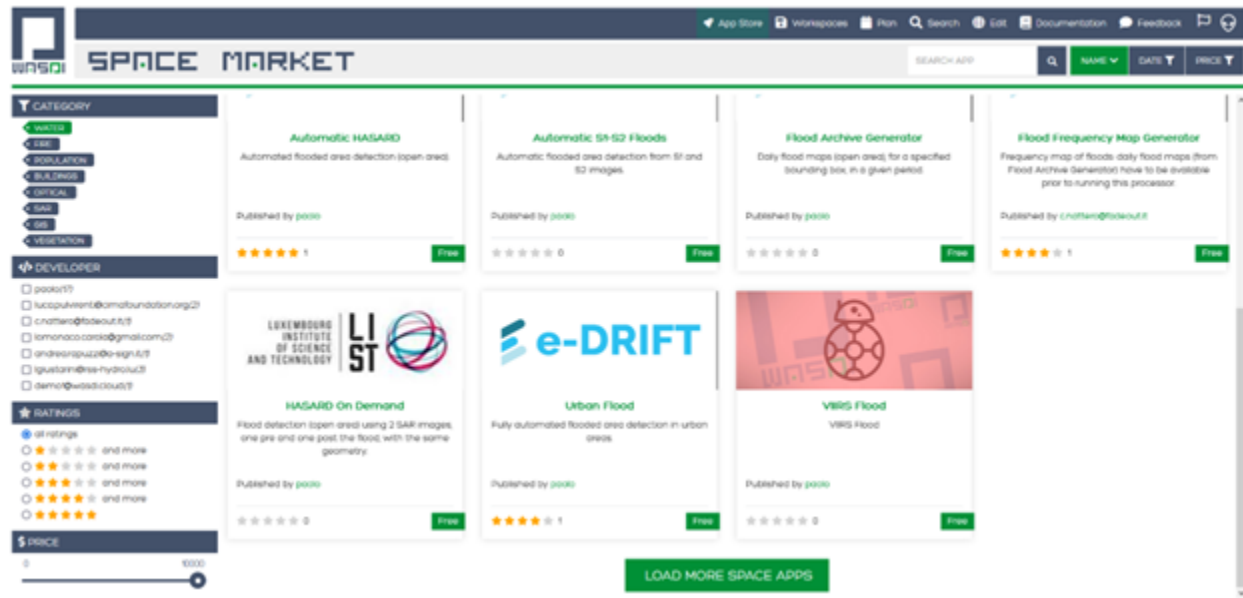
REFERENCES Pulvirenti, Luca, Giuseppe Squicciarino, and Elisabetta Fiori. 2020. "A Method to Automatically Detect Changes in Multitemporal Spectral Indices: Application to Natural Disaster Damage Assessment" Remote

Sensing 12, no. 17: 2681. <https://doi.org/10.3390/rs12172681>


3.2.4 Algorithms to map floods in open areas from VIIRS


VIIRS Flood

VIIRS Flood produces VIIRS flood map for a specific event and a given areas: it searches the nearest VIIRS images with respect to the date of the event date and it makes a mosaic in the area of interest. If more than one image is available, the closest to the event date is taken in order of priority. The ones of the following days are used to try and fill the cloud gaps. The user can control the number of such days.




[← Back to the Future](#)



VIIRS Flood
Publisher: poolo 
[MORE DETAILS](#)


☒ New Workspace
☐ Open Workspace

 RUN


Basic
Advanced
Help
History
JSON

Code

Event Date



Area





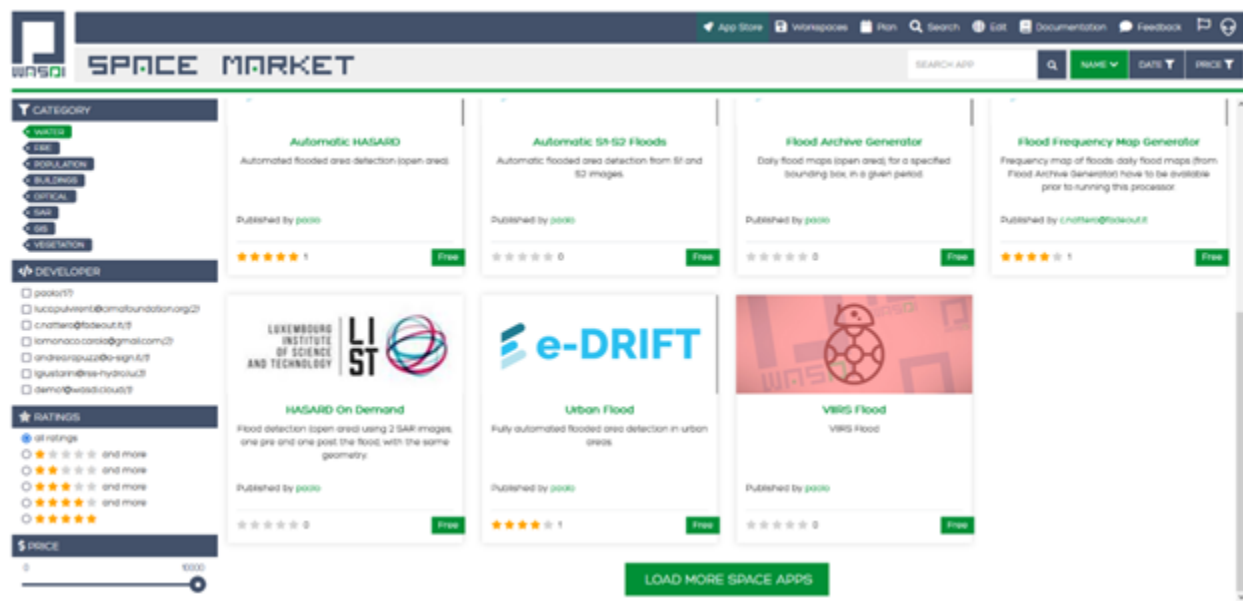
3.2.5 Floods in urban areas

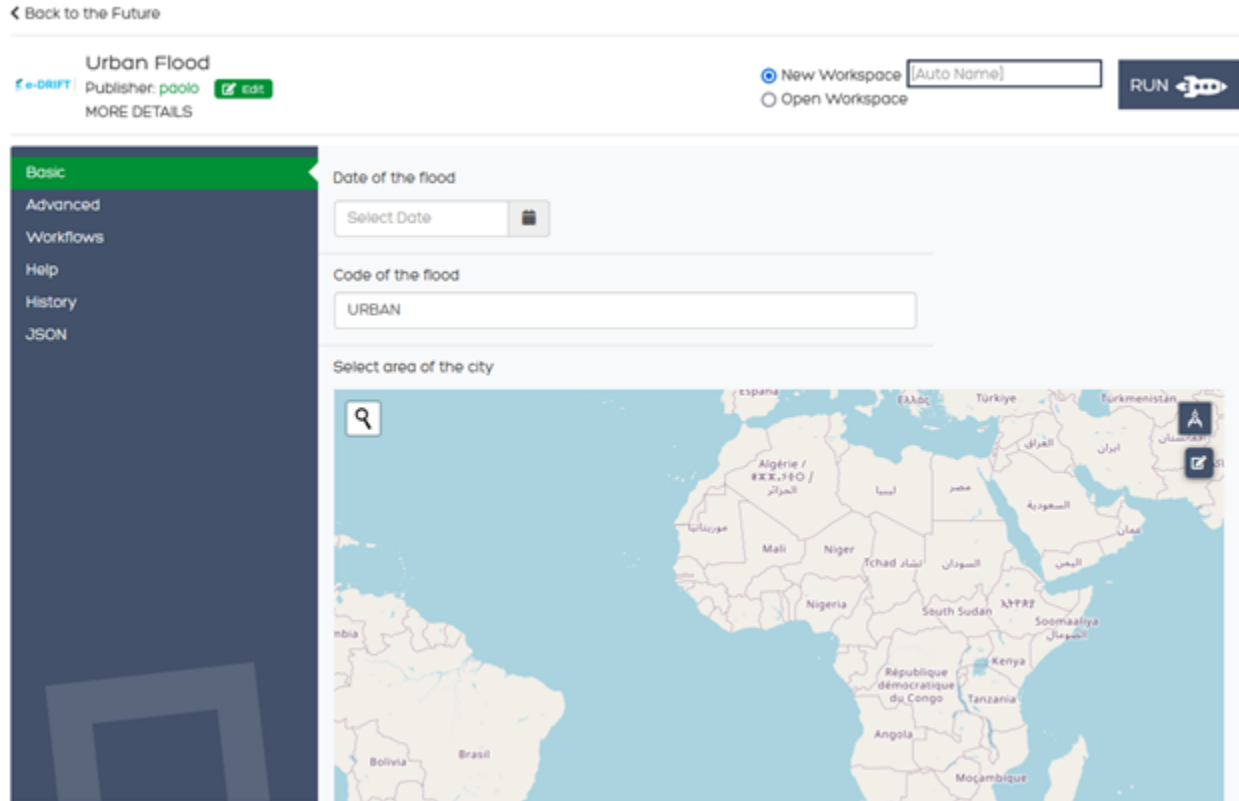
3.2.6 Algorithms to map floods in urban areas from Sentinel-1

Urban Flood

Urban Flood can be used to map floods in urban on a specific date. It is based on a multi-pass approach exploiting a stack of interferometric acquisitions. The coherence map between each consecutive pair of images is extracted using a square moving window. Given t_0 , i.e., the date of the image acquired during the flood event, we denote with c_0 the coherence of the image pair acquired on t_0 and t_1 , and c_{pre} the one with images acquired on t_1 and t_2 .

- Step (1) allows the double-bounce map to be extracted, i.e., the building footprints.
- Step (2) combines the double-bounce map and the change of $c_{pre} - c_0$. The underlying assumption is that urban areas affected by a flood have $c_0 < c_{pre}$.

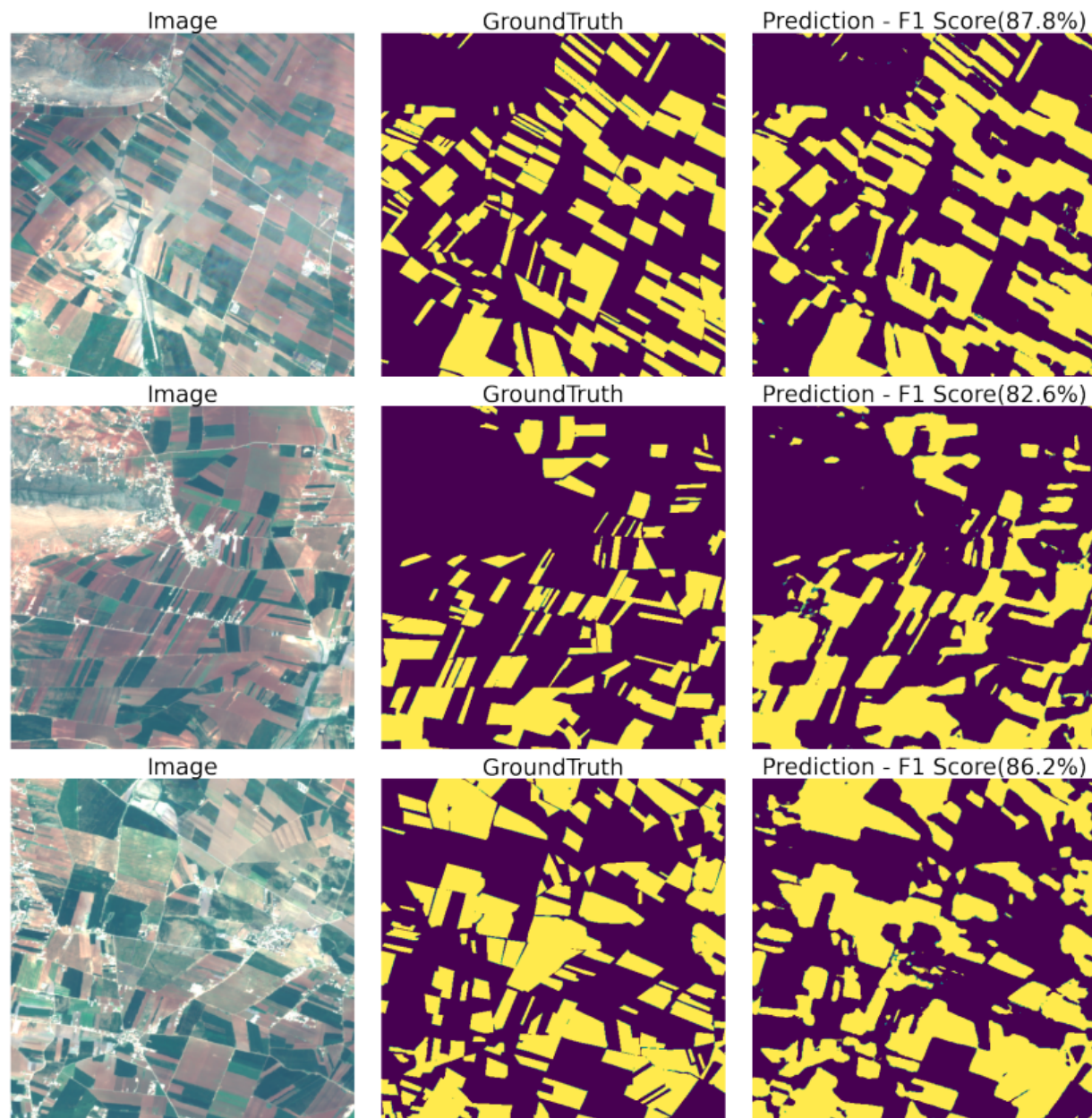




3.3 Wheat Locator

3.3.1 About?

Developed by the GEOAI research group, the “Wheat Locator” App uses a deep learning transformer-based model to extract wheat fields from Sentinel-2 images as detailed in this [manuscript](#). This work explores the feasibility of cross-area and cross-year out-of-distribution generalization of crop segmentation models. We adopted the TSViT model to address wheat field segmentation in Lebanon using PEFT techniques. We relied on an in-house labeled dataset, called the Lebanese Wheat dataset, that comprises high-quality annotated polygons for wheat and non-wheat classes for the study area in the Beqaa area, Lebanon, with a total surface of 170 km², over five consecutive years from 2016 to 2020. Using a time series of multispectral Sentinel-2 images, our model achieved an 84% F1-score. Our code is publicly available at this [Repo](#).



The App will automatically download the required Region of Interest (RoI) images or use available workspace images. Since we trained the transformer model on a Satellite Images Time Series (SITS) of Lebanon, the model's out-of-distribution (OOD) performance will be hindered when testing using new regions. This work is in the beta phase, and your feedback is highly appreciated via this [LINK](#) or through info@geogroup.ai.

Wheat Locator
 Publisher: mohazahweh@gmail.com [Edit](#)
[New Workspace](#) [Auto Name] [Open Workspace](#) [RUN](#)

Main Options
 Advanced Options
 Help
 History
 JSON

Choose Sowing (Planting) Month
 1 11 12

Choose Harvesting Month
 1 7 12

Choose Year
 2016 2019 2023

Bounding Box

3.3.2 Input Parameters:

1. Sowing (Planting) Month: wheat planting month in the selected RoI
2. Harvesting Month: wheat harvesting month in the selected RoI
3. Year: wheat harvest year
4. Bounding Box: to reduce time and computational complexity into an acceptable size, the area of the bonding box should be less than 30km² (3000ha)

3.3.3 Credits:

Models training and WASDI deployment by Eng. [Mohamad Hasan Zahweh](#) under Dr Ali J. Ghandour supervisor.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.

ADD YOUR APP TO WASDI

Unleash the real power of WASDI, developing and uploading your own downstream application to run it on EO images on the fly! Wasdi supports several programming languages:

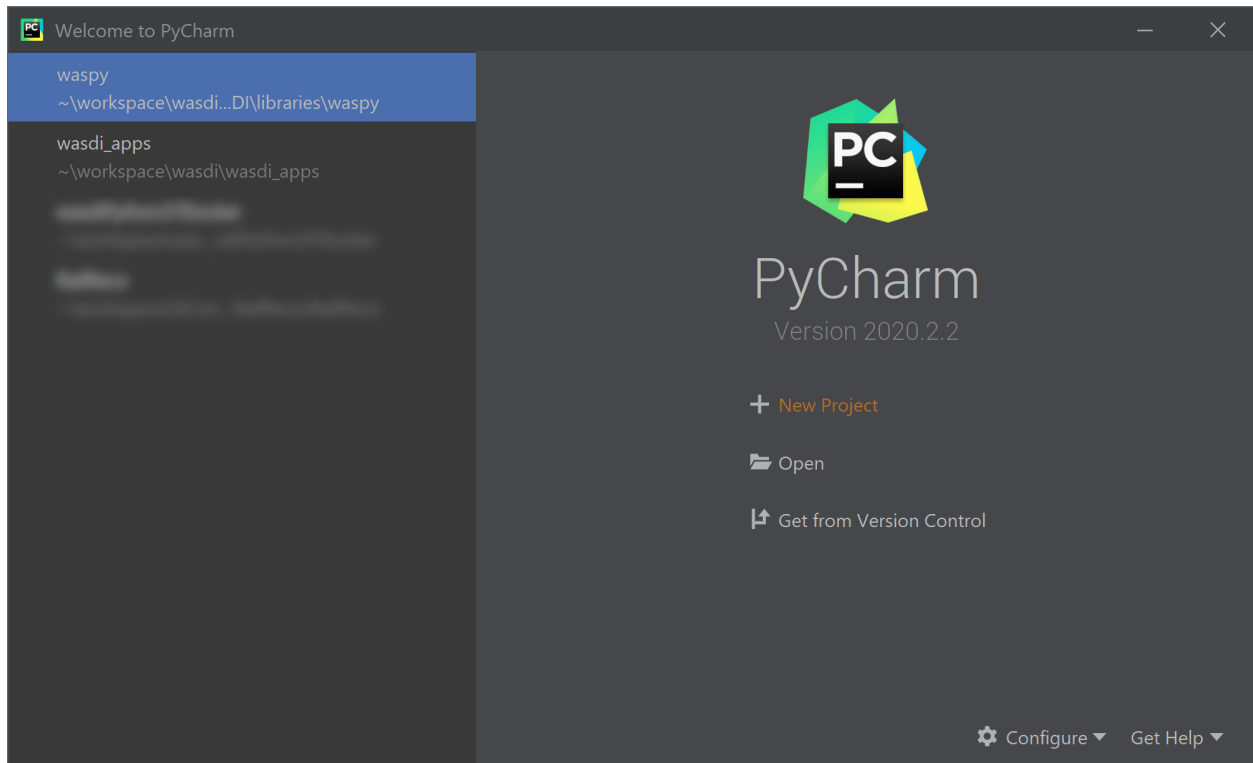
- **Python 3.x**
- **IDL 3.7.2**
- **Octave 6.x**
- **C#**
- **Javascript**

If you already know WASDI features and you are a Python developer check out the python tutorial

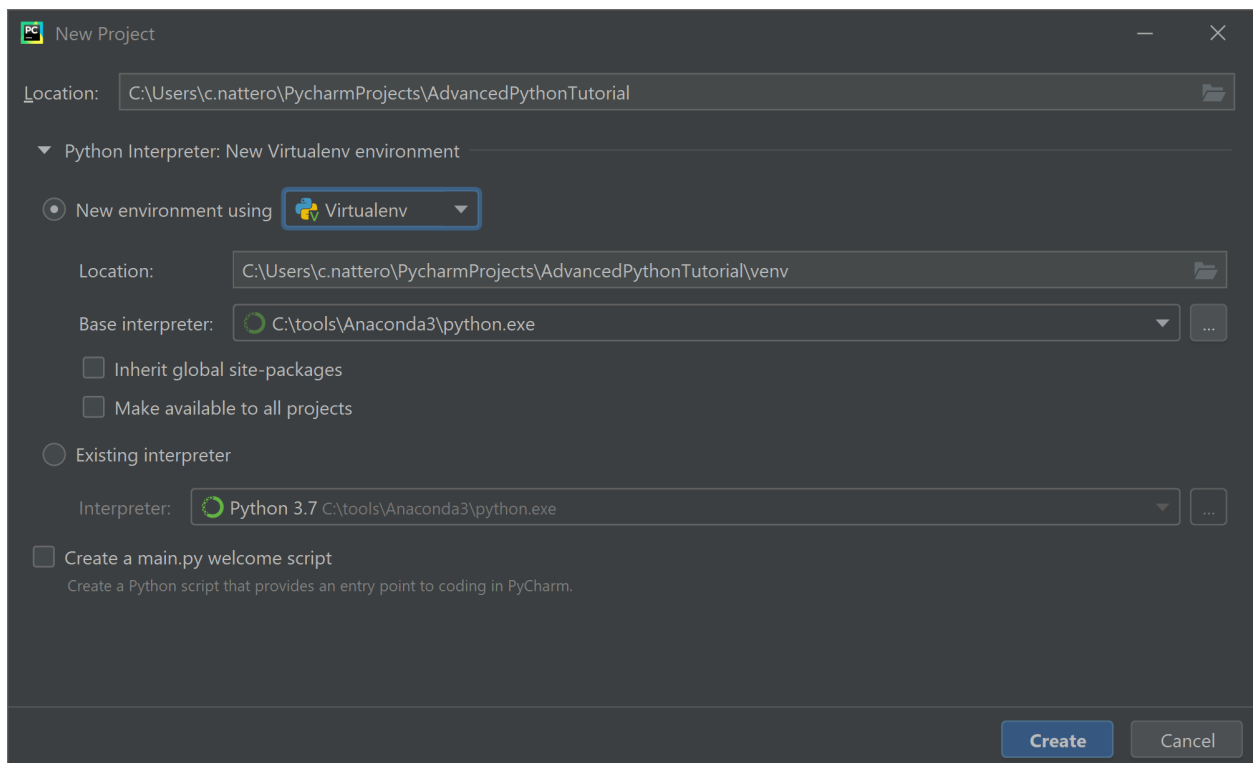
4.1 Python Tutorial

4.1.1 Setup

Open PyCharm and start a new project.



Call it “Advanced Python Tutorial” (or however you wish, just remember to be coherent). You may wish to create a new virtual environment (here we derive it from Python 3.8). Uncheck the option for creating a “main.py” welcome script (or, at least, remember to delete it later on).



Let’s install the library we need. In the terminal write:

```
pip install wasdi
```

and press enter

```

(AdvancedPythonTutorial) C:\Users\c.nattero\PycharmProjects\AdvancedPythonTutorial>pip install wasdi
Collecting wasdi
  Downloading wasdi-0.5.1.tar.gz (33 kB)
Collecting requests
  Downloading requests-2.24.0-py2.py3-none-any.whl (61 kB)
Collecting urllib3<1.25.0,!=1.25.1,<1.26,>=1.21.1
  Downloading urllib3-1.25.11-py2.py3-none-any.whl (127 kB)
Collecting idna<3,>=2.5
  Downloading idna-2.10-py2.py3-none-any.whl (58 kB)
Requirement already satisfied: certifi>=2017.4.17 in c:\tools\anaconda3\envs\advancedpythontutorial\lib\site-packages (from requests->wasdi) (2020.6.20)
Collecting chardet<4,>=3.0.2
  Downloading chardet-3.0.4-py2.py3-none-any.whl (133 kB)
Building wheels for collected packages: wasdi
  Building wheel for wasdi (setup.py) ... done
  Created wheel for wasdi: filename=wasdi-0.5.1-py3-none-any.whl size=27736 sha256=370ded1658747d7733645f9367bd768c88818debc271159387c3ad2f4f3ded8
  Stored in directory: c:\users\c.nattero\appdata\local\pip\cache\wheels\37\89\c7\6ce30bb0f7b51acd9c8f2b4845cf76a4ef148e27fce8d754ed
Successfully built wasdi
Installing collected packages: urllib3, idna, chardet, requests, wasdi
Successfully installed chardet-3.0.4 idna-2.10 requests-2.24.0 urllib3-1.25.11 wasdi-0.5.1

```

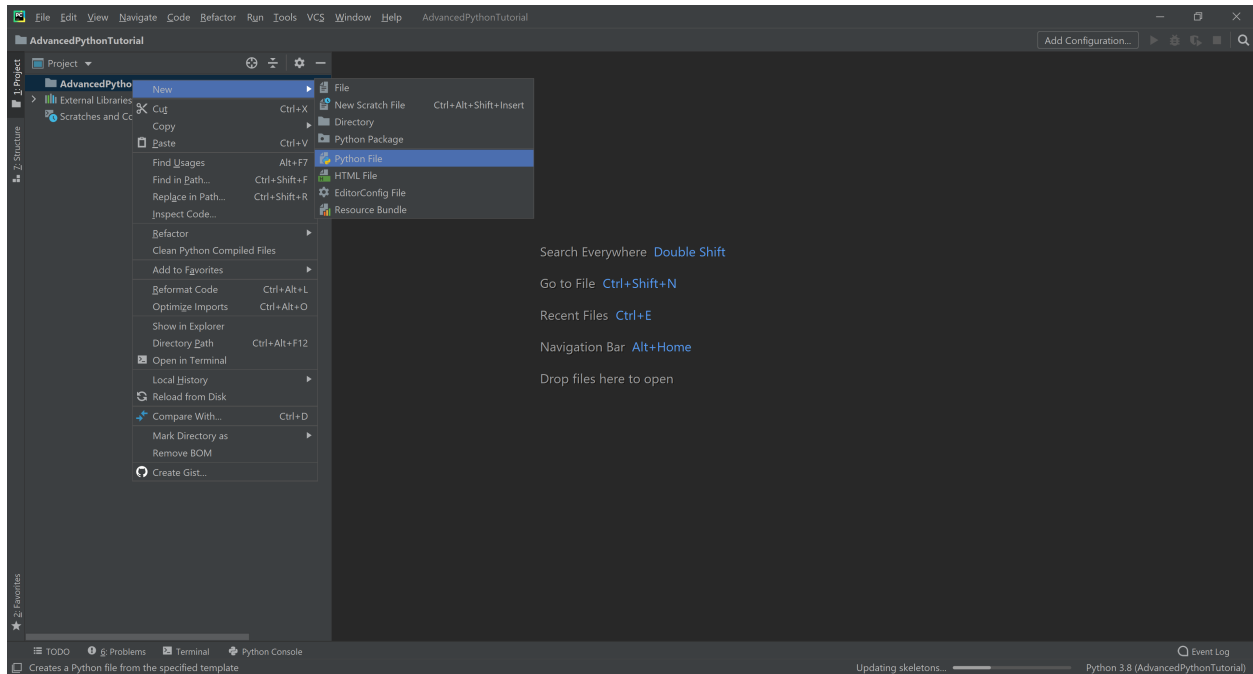
hint: if you previously installed wasdi, you may wish to update it by adding the `--upgrade` flag, i.e.:

```
pip install --upgrade wasdi
```

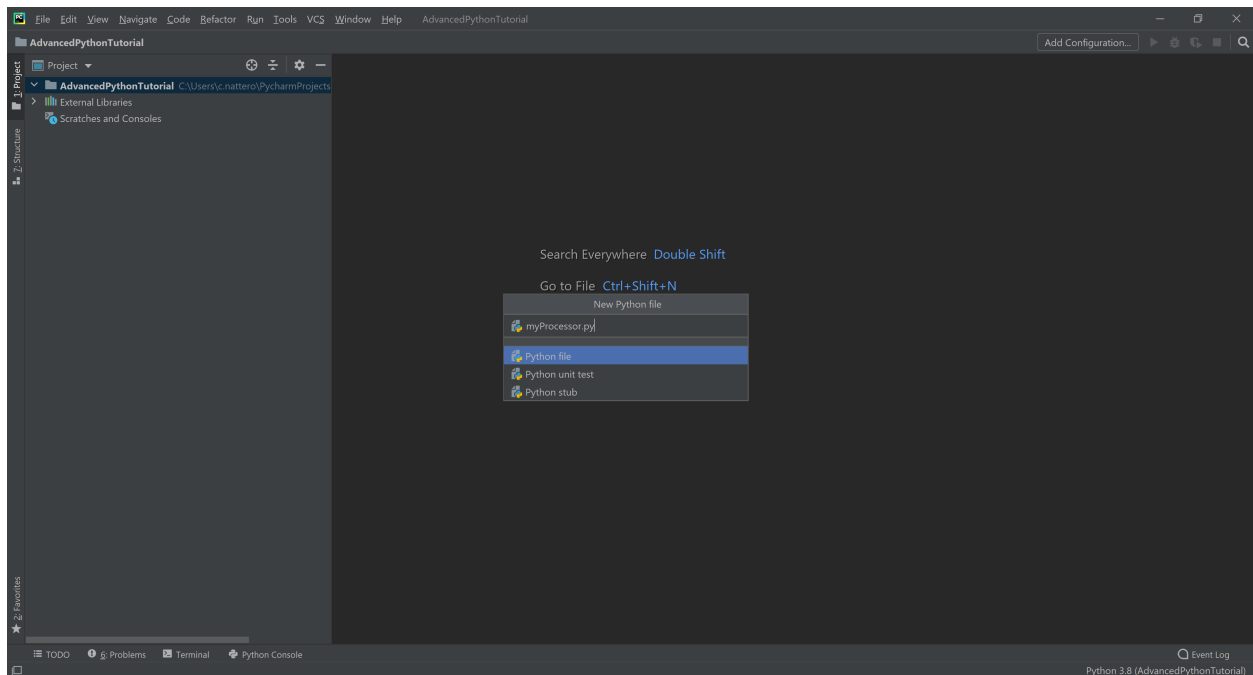
4.1.2 Create first files

Now we need to create these two fundamental files (right click on the AdvancedPythonTutorial project icon, new -> ...):

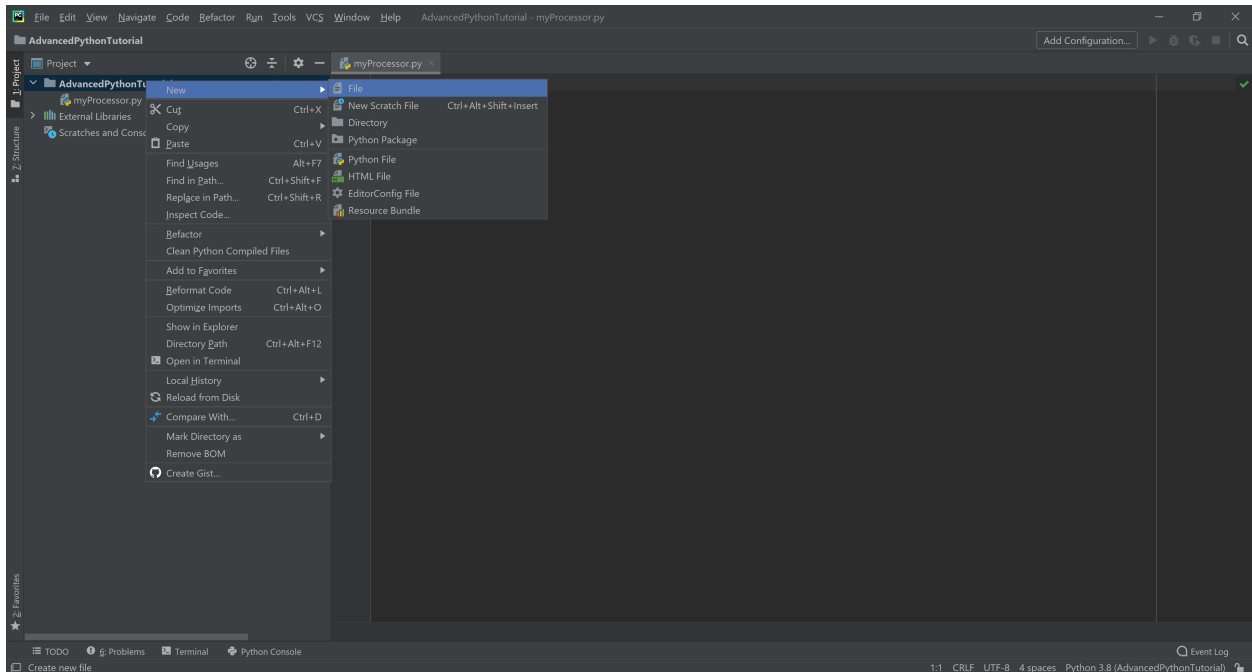
- `myProcessor.py`: create a python file, then call it `myProcessor.py`
- `config.json`: create a file, then call it `config.json` (PyCharm will recognize automatically it's a JSON file)



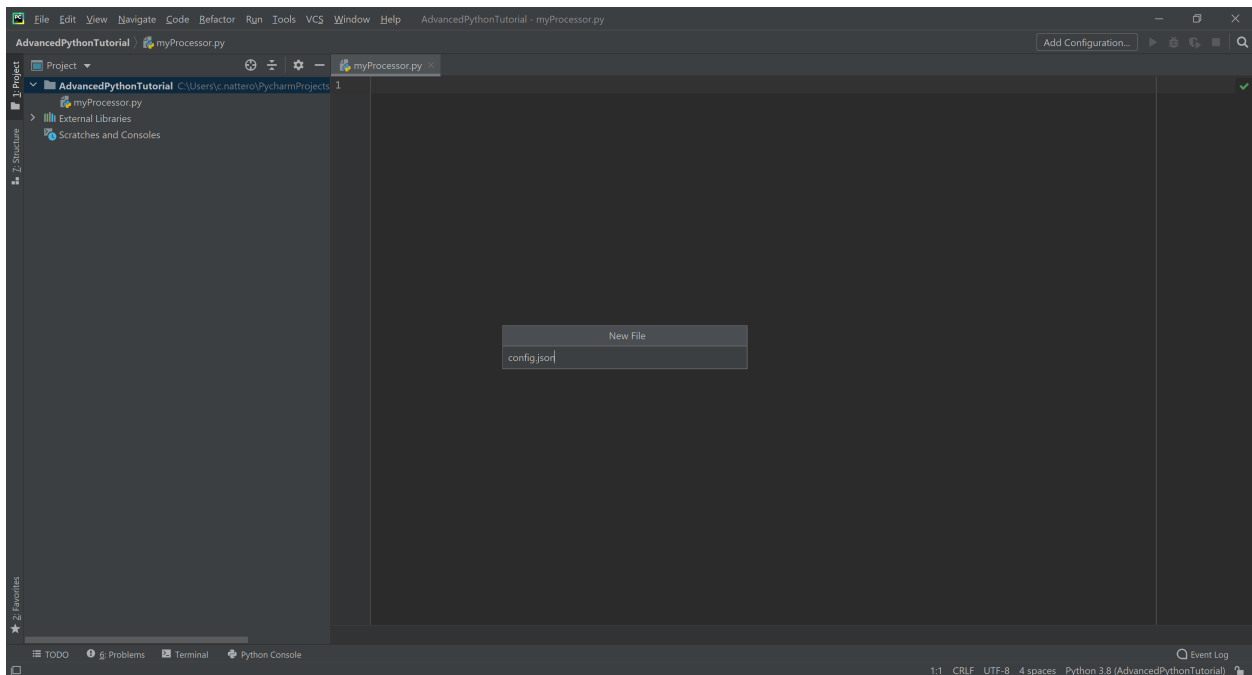
Create python file



Create myProcessor.py



Create a file



Create config.json

Next, point your browser to wasdi.net, log in, make sure you see the workspaces page (otherwise click on the workspaces menu) and create a new workspace. Call it somehow, e.g., “AdvancedTutorialTest” (or however you want, just remember it and be coherent later on). Leave the browser open on that page, we’ll need it later on.

Create a workspace called AdvancedTutorialTest

4.1.3 First lines

Let's begin by editing the config.json file. It's a JSON file, containing the user credentials and some fundamental parameters to get you started:

```
{
  "USER": "your user name here",
  "PASSWORD": "your password here",
  "WORKSPACE": "AdvancedTutorialTest"
}
```

NOTE: please, keep this file for yourself. You should never give this file to anyone else, and you do not need to upload to WASDI, as we'll see later on. You just need this file in your project for working with the WASDI python library.

Now, open myProcessor.py, create a main and a method called run. The latter is required for WASDI to work (more on that later on).

Note: these are two requirements necessary to use WASDI:

- have a python file called myProcessor.py
- have a function called run() (no params) within myProcessor.py

After that, you can include as many python files as you need, no matter if they are organized in directories. You just need to have a myProcessor.py with a method run() as entry point.

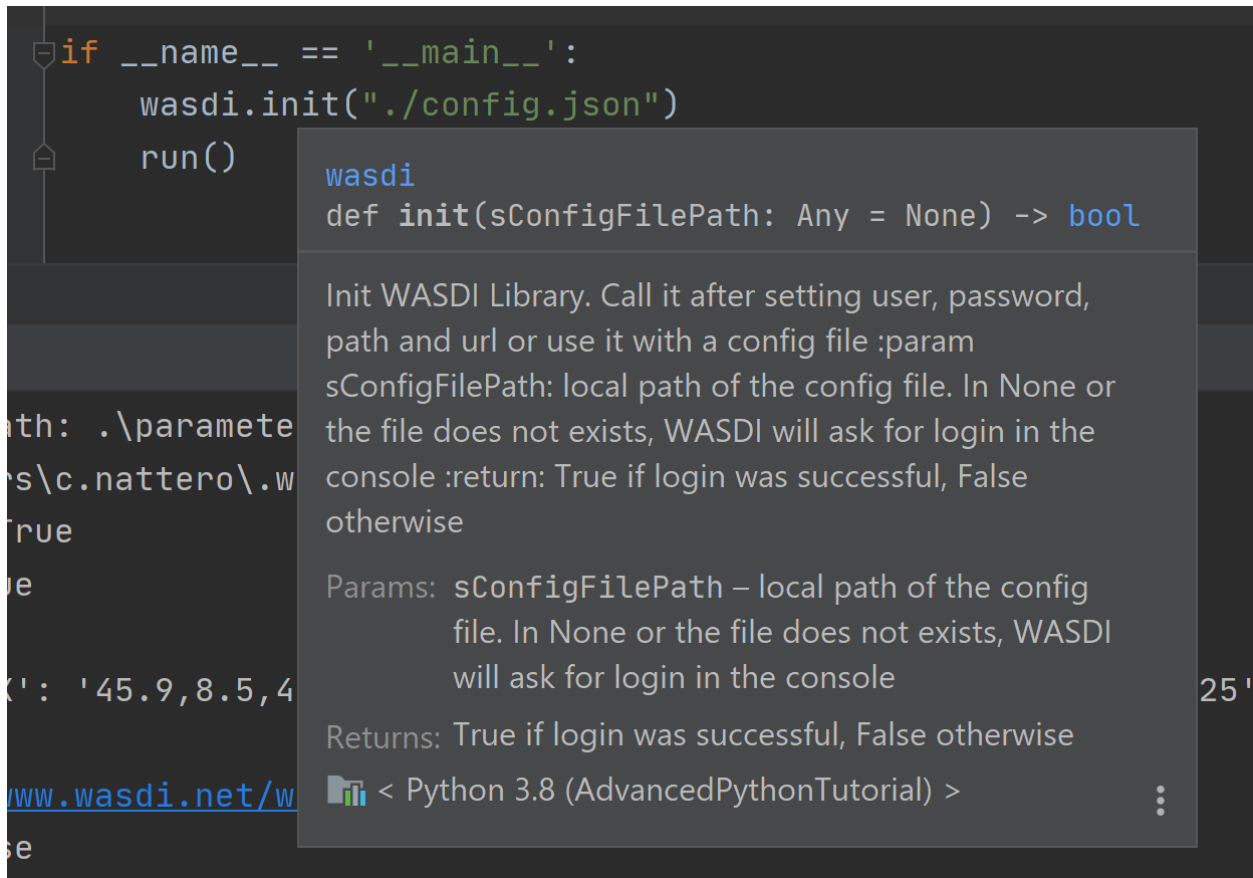
The main method will initiate the WASDI library and call the run method:

```
import wasdi

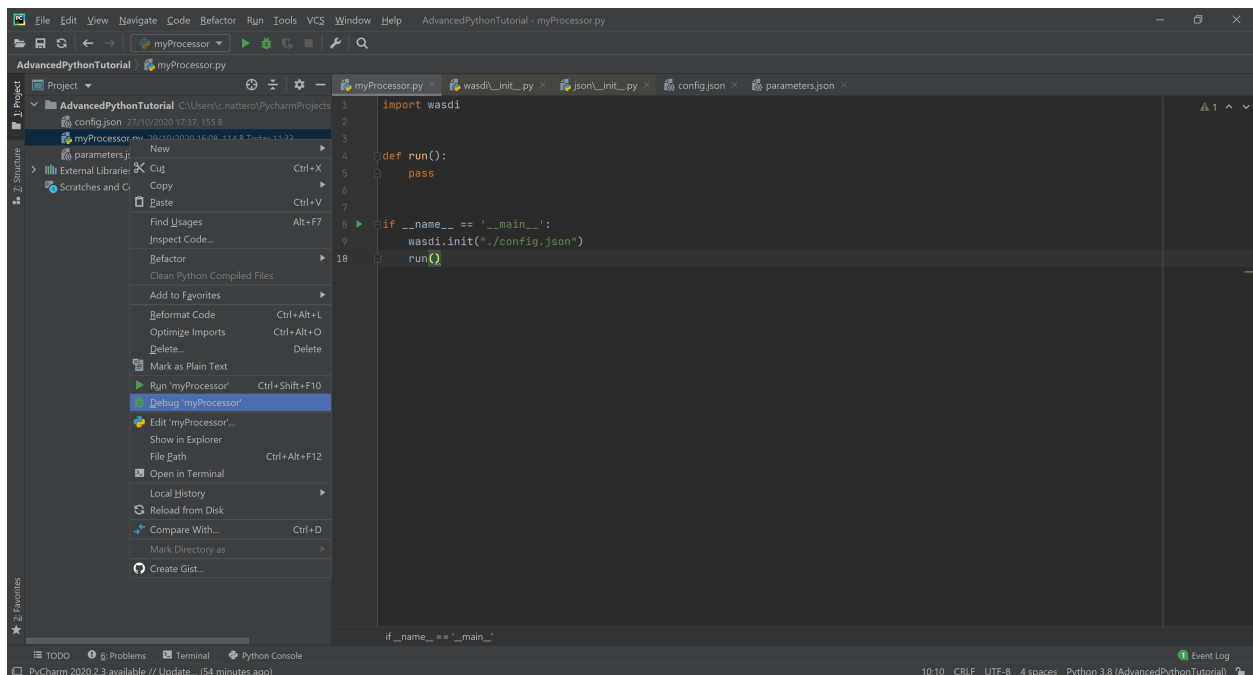
def run():
    pass

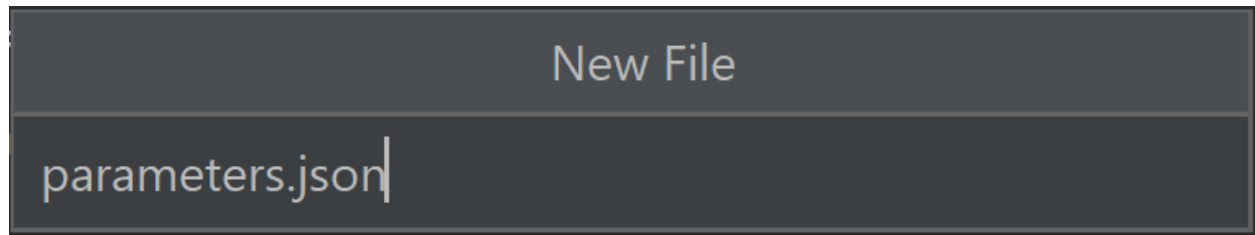
if __name__ == '__main__':
    wasdi.init("./config.json")
    run()
```

As you can see, we call wasdi.init and pass the relative path of the config file to it.



Let's debug to see the effects of this. Note: if a file `main.py` was created automatically for you, remember to define another debug configuration. The easiest way to do so is by right clicking on your code and select Debug 'myProcessor.py'.





If the setup is correct so far, we should see the output from the wasdi library that shows the initialization has gone well. Let's see it more in details:

```
[INFO] _loadParams: wasdi could not load param file. That is fine, you can still load it.  
→ later, don't worry
```

We'll see to this later, for now we trust it and do not worry ;-)

```
[INFO] waspy.init: returned session is: 0d3f3ef1-f4c3-4202-9015-6ca17fc21cc7
```

Great, we authenticated and got a session (yours is going to be different)

```
[INFO] waspy.init: WASPY successfully initiated :-)
```

Good news

```
[INFO] waspy.printStatus: user: username@email.address  
[INFO] waspy.printStatus: password: *****  
[INFO] waspy.printStatus: session id: 0d3f3ef1-f4c3-4202-9015-6ca17fc21cc7
```

Looks like our credentials worked. Your username and session id will be different, and the password will not be shown. Pay attention, if you forget to insert the password, WASDI will ask you for it.

```
[INFO] waspy.printStatus: active workspace: 4f541d2c-4b29-445b-9869-9c8d185932ce  
[INFO] waspy.printStatus: workspace owner: username@email.address
```

This code corresponds to the workspace we opened, i.e., AdvancedTutorialTest (it's going to be different for you), next is the email address you used to register on WASDI

```
[INFO] waspy.printStatus: parameters file path: None
```

We did not provide a parameter file, we'll see this later

```
[INFO] waspy.printStatus: base path: C:\Users\username\.wasdi\
```

This is the base path inside which WASDI will mirror the online file structure, creating one folder per workspace

```
[INFO] waspy.printStatus: download active: True  
[INFO] waspy.printStatus: upload active: True
```

Downloads and uploads will happen automatically when necessary

```
[INFO] waspy.printStatus: verbose: True
```

Verbosity

```
[INFO] waspy.printStatus: param dict: {}
```


No params so far

```
[INFO] waspy.printStatus: proc id:
[INFO] waspy.printStatus: base url: http://www.wasdi.net/wasdiwebserver/rest
[INFO] waspy.printStatus: is on server: False
[INFO] waspy.printStatus: workspace base url: http://www.wasdi.net/wasdiwebserver/rest
```

More config info, which are fine

```
[INFO] waspy.printStatus: session is valid :-)
```

This is good

Process finished with exit code 0

And the debug finishes

4.1.4 WASDI Hello World

Now let's try to call a WASDI API. There's a hello world API just for these tests. Let's change the run method code as follows:

```
def run():
    sHello = wasdi.hello()
    print(sHello)
```



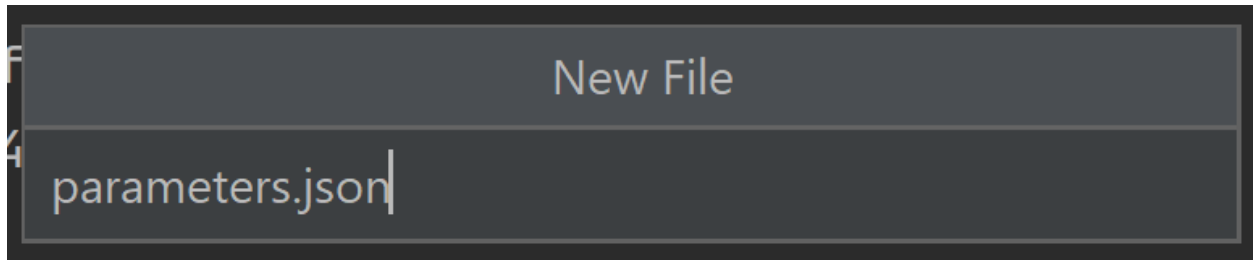
Basically, the method wasdi.hello wraps a call to WASDI hello world API and returns the response, which we print on the next line, getting the following result:

```
{"boolValue":null,"doubleValue":null,"intValue":null,"stringValue":"Hello Wasdi!!"}
```

So that's our first WASDI processor: we demonstrated we can authenticate and call an API using code.

4.1.5 Introducing parameters

Now we'll see how WASDI lets you handle the parameters for your processor. Let's create a new file called parameters.json (Right click on the project, New -> File, name it parameters.json)



That's another JSON file where the developer can set and/or simulate inputs for his processor. The idea is that WASDI processors can manipulate satellite images fed in input to create added-value products to be output. Parameters are those variables needed by the developer to retrieve input data and/or generate output data.

In the tutorial we are going to see some typical examples: the area of interest, the type of satellite data, a date or interval of dates. These parameters are defined in the file params.json During the development and the debug of the processor, the developer must write her/his input in this file. It's like a dictionary: this way, the programmer decides what are the parameters and their syntax, and by assigning them a value she/he can test them.

Let's try this example:

```
{  
  "NAME": "advanced python tutorial"  
}
```

We also need to edit the config.json file to specify that we want to use parameters.json as the parameters file, and that's done by adding the following line:

```
"PARAMETERSFILEPATH": "./parameters.json"
```

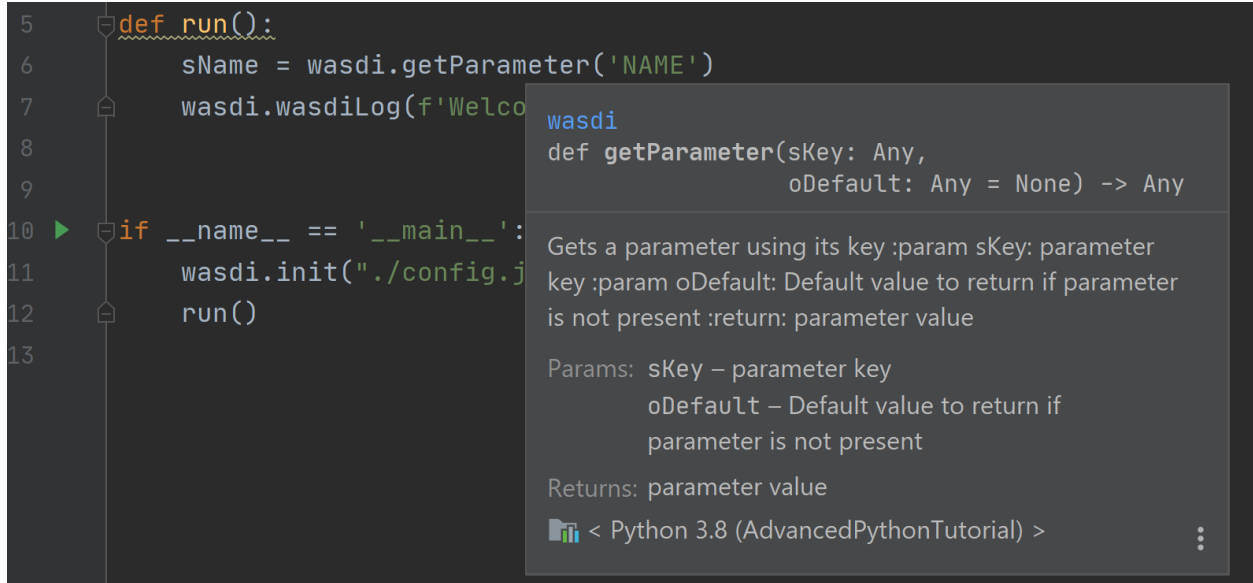
(please check that the JSON is valid, check especially your commas).

Now edit the run method and change it as follows:

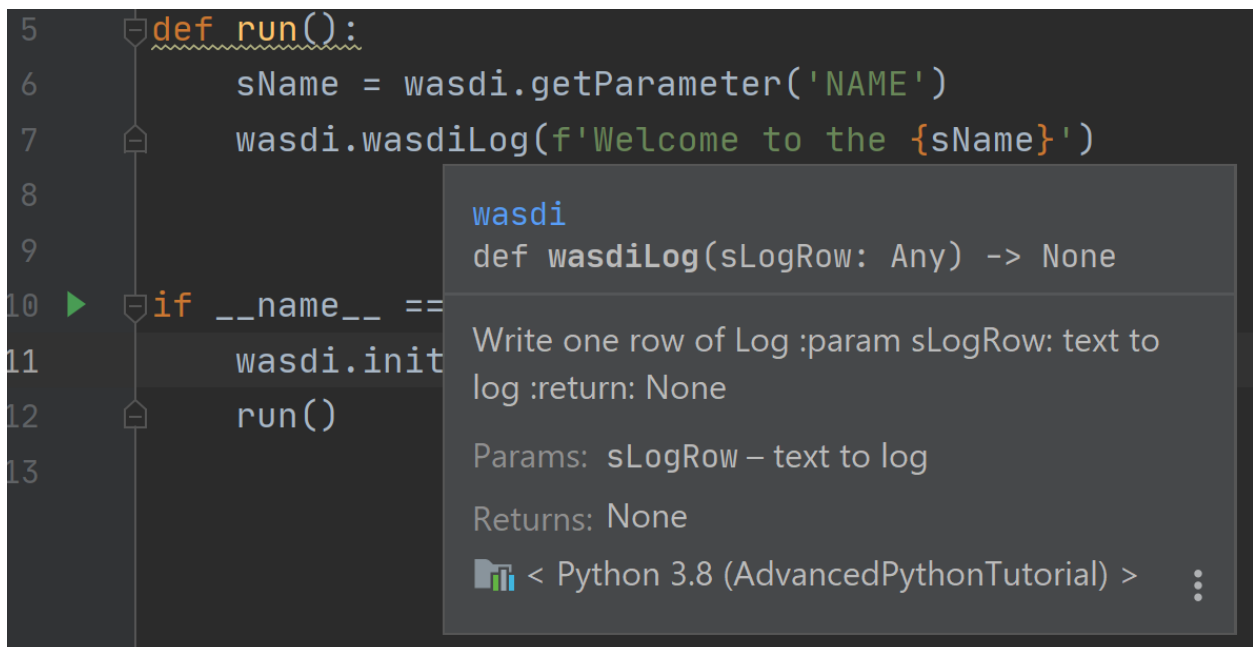
```
def run():  
    sName = wasdi.getParameter('NAME')  
    wasdi.wasdiLog(f'Welcome to the {sName}')
```

During the initialization, parameters are read from the specified file.

wasdi.getParameter is the method for reading a single parameter, and a default value can be specified.



`wasdi.wasdiLog` is the utility for logging a line. It's a print, locally, but when executed on the cloud, it prints a long line on the user interface.



Let's debug it and we're going to see, after the initialization output, the following line:

```
Welcome to the advanced python tutorial
```

Parameters can be of any type supported by the JSON format. When the processor will be deployed, the final user, or third party systems will be able to run it passing these parameters.

4.1.6 A more meaningful example

Let's try another example. We want to write a processor that searches for Sentinel-2 images and uses them to create a RGB GeoTIFF file.

You can download the final code from here:

[myProcessor.py](#)

4.1.7 Step 1: read and validate parameters

Let's change our parameters in parameters.json as follows:

```
{
  "BBOX": "45.9,8.5,45.7,8.7",
  "MAXCLOUD": "30",
  "DATE": "2020-10-25",
  "SEARCHDAYS": "20"
}
```

Now the file is in its final form, and you can download the file from here: [parameters.json](#)

These parameters represent, respectively:

- the area of interest in the format "NORTH, WEST, SOUTH, EAST"
- the maximum cloud coverage (percentage)
- a date in which we want to search images
- a maximum number of days to search back in time.

Now, edit the code of myProcessor.py

First of all, add the following imports:

```
from datetime import datetime
from datetime import timedelta
```

Next, modify the run method as follows:

```
def run():
    # STEP 1: Read "real" parameters
    sBBox = wasdi.getParameter("BBOX")
    sDate = wasdi.getParameter("DATE")
    sMaxCloud = wasdi.getParameter("MAXCLOUD", "20")
    sSearchDays = wasdi.getParameter("SEARCHDAYS", "10")
    sProvider = wasdi.getParameter("PROVIDER", "AUTO")
    # L1
    sImageType = wasdi.getParameter("IMAGETYPE", "S2MSI1C")
    # L2
    # sImageType = wasdi.getParameter("IMAGETYPE", "S2MSI2A")
    # Check the Bounding Box: is needed
    if sBBox is None:
        wasdi.wasdiLog("BBOX Parameter not set. Exit")
        wasdi.updateStatus("ERROR", 0)
    return
```

(continues on next page)

(continued from previous page)

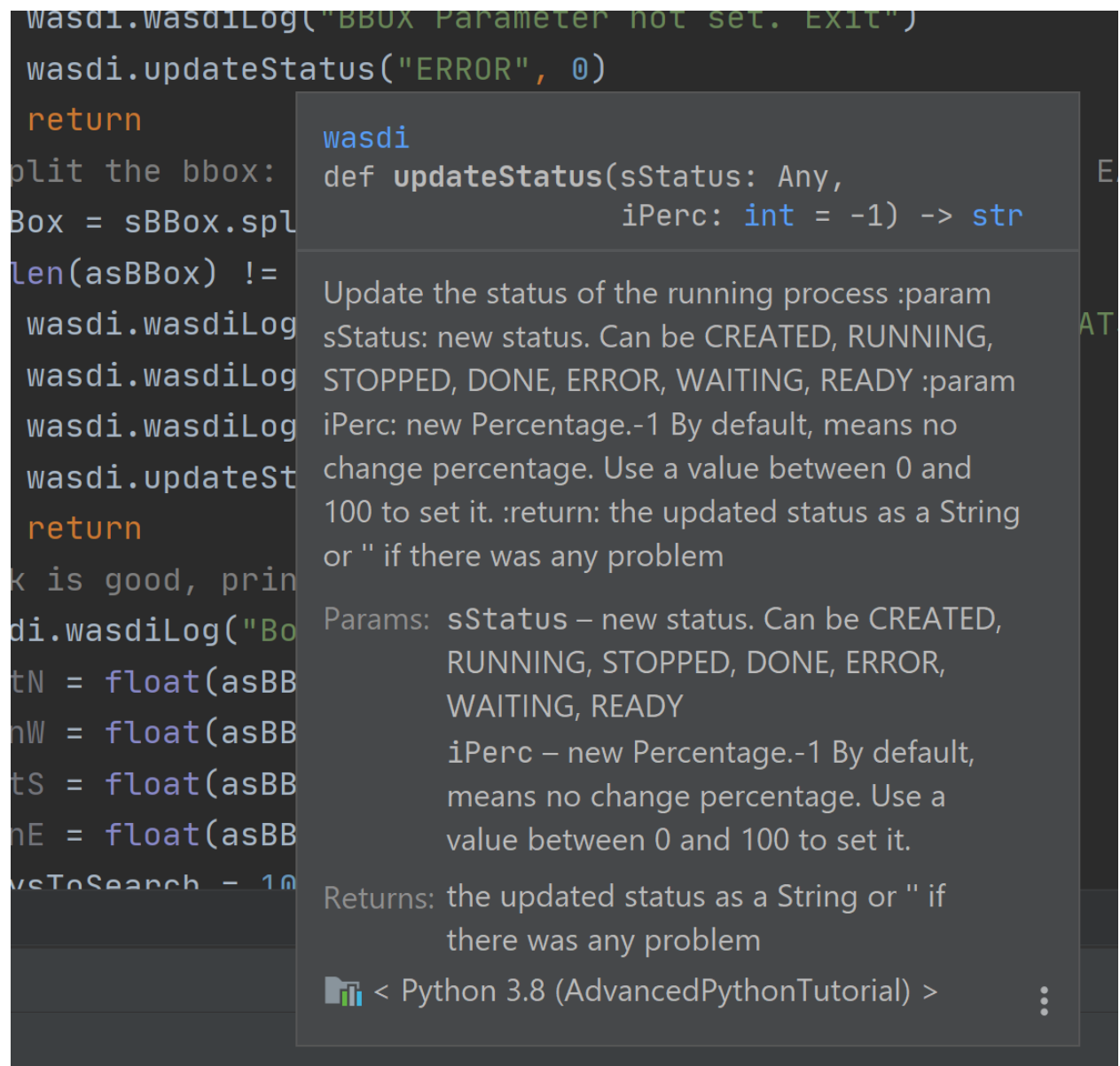
```

# Split the bbox: it is in the format: NORTH, WEST, SOUTH, EAST
asBBox = sBBox.split(",")
if len(asBBox) != 4:
    wasdi.wasdiLog("BBOX Not valid. Please use LATN,LONW,LATS,LONE")
    wasdi.wasdiLog("BBOX received:" + sBBox)
    wasdi.wasdiLog("exit")
    wasdi.updateStatus("ERROR", 0)
    return
# Ok is good, print it and convert in float
wasdi.wasdiLog("Bounding Box: " + sBBox)
fLatN = float(asBBox[0])
fLonW = float(asBBox[1])
fLatS = float(asBBox[2])
fLonE = float(asBBox[3])
iDaysToSearch = 10
try:
    iDaysToSearch = int(sSearchDays)
except Exception as oEx:
    wasdi.wasdiLog(f'Number of days to search not valid due to {repr(oEx)}, assuming 10 [
↳ ' + str(sSearchDays) + "]")
# Check the date: assume now
oEndDay = datetime.today()
try:
    # Try to convert the one in the params
    oEndDay = datetime.strptime(sDate, '%Y-%m-%d')
except Exception as oEx:
    # No good: force to yesterday
    wasdi.wasdiLog(f'Date not valid due to {repr(oEx)}, assuming today')
oTimeDelta = timedelta(days=iDaysToSearch)
oStartDay = oEndDay - oTimeDelta
sEndDate = oEndDay.strftime("%Y-%m-%d")
sStartDate = oStartDay.strftime("%Y-%m-%d")
# Print the date
wasdi.wasdiLog("Search from " + sStartDate + " to " + sEndDate)
# Check the cloud coverage
sCloudCoverage = None
if sMaxCloud is not None:
    sCloudCoverage = "[0 TO " + sMaxCloud + "]"
    wasdi.wasdiLog("Cloud Coverage " + sCloudCoverage)
else:
    wasdi.wasdiLog("Cloud Coverage not set")

```

The code reads, validates and manipulates the parameters.

updateStatus is another primitive: it allows to update the process status and the progress (percent) of its execution.



Each WASDI process has a status among the following:

- **CREATED:** a newly created process, waiting to be executed
- **RUNNING:** a process that is being executed
- **WAITING:** a process that was running and is now waiting for another resource, and has been put on hold for this reason
- **READY:** a process that obtained the resource for which it was **WAITING** and is now waiting for the WASDI scheduler to continue executing it
- **DONE:** process that completed successfully
- **ERROR:** the execution encountered some error that prevented the process from completing correctly
- **STOPPED:** process stopped by the user or by another processor.

Let's run it and, if everything is properly set, we will see the usual output, but now we are going to see these two lines too (one is different, the other is new):

```
[INFO] waspy.printStatus: parameters file path: .\parameters.json
```

```
[INFO] waspy.printStatus: param dict: {'BBOX': '45.9,8.5,45.7,8.7', 'MAXCLOUD': '30',  
↪ 'DATE': '2020-10-25', 'SEARCHDAYS': '10'}
```

Also, we are going to see our logs:

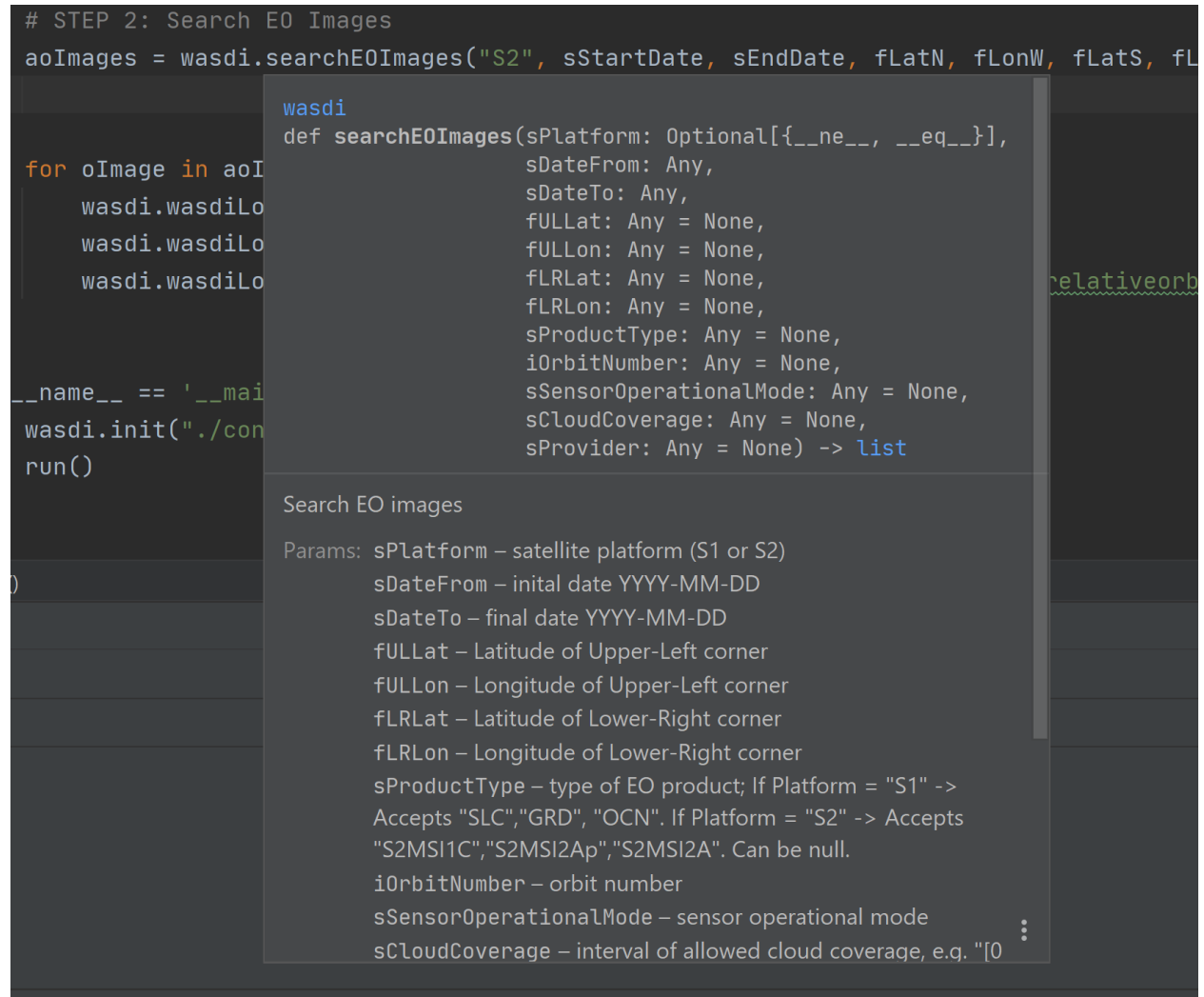
```
Bounding Box: 45.9,8.5,45.7,8.7  
Search from 2020-10-15 to 2020-10-25  
Cloud Coverage [0 TO 30]
```

4.1.8 Step 2: search the catalogs for EO data

Add the following lines to the run method to search for EO images

```
# STEP 2: Search EO Images  
aoImages = wasdi.searchEOImages("S2", sStartDate, sEndDate, fLatN, fLonW, fLatS, fLonE,  
↪ sImageType, None, None, sCloudCoverage, sProvider)  
for oImage in aoImages:  
    wasdi.wasdiLog("Image Name WITHOUT Extension:" + oImage['title'])  
    wasdi.wasdiLog("Image Name WITH Extension:" + oImage['fileName'])
```

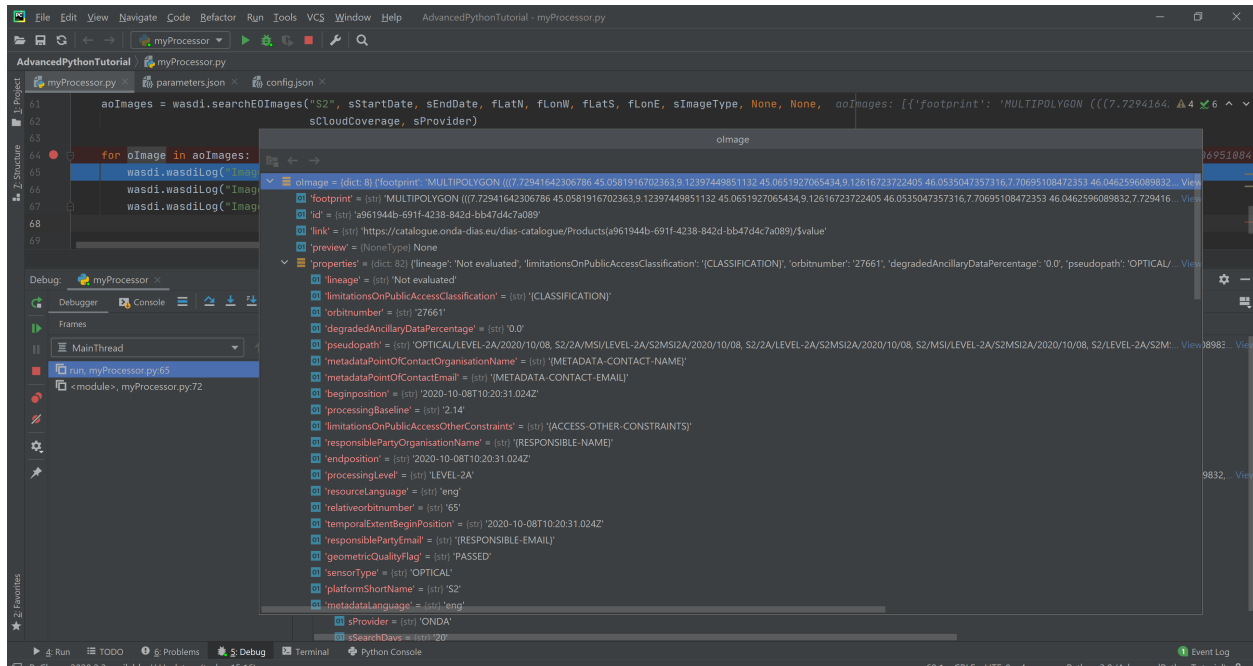
The method `searchEOImages` allows filtering for area of interest (bounding box), mission, product type, orbit number, sensor operational mode and cloud coverage (when applicable to the data type). A more advanced usage allows to specify the provider to use, but that's beyond the scope of this tutorial.



The method returns a list of objects, one per image. Each of these object is in turn a dictionary, describing the image: it contains every propriety returned by the search, such as, for example:

- footprint
- beginPosition
- endPosition
- cloudShadowPercentage
- relativeOrbitNumber
- orbitDirection

There are many more, and we can see them debugging the code. Please note that the number and type of these parameters depends on the data provider.



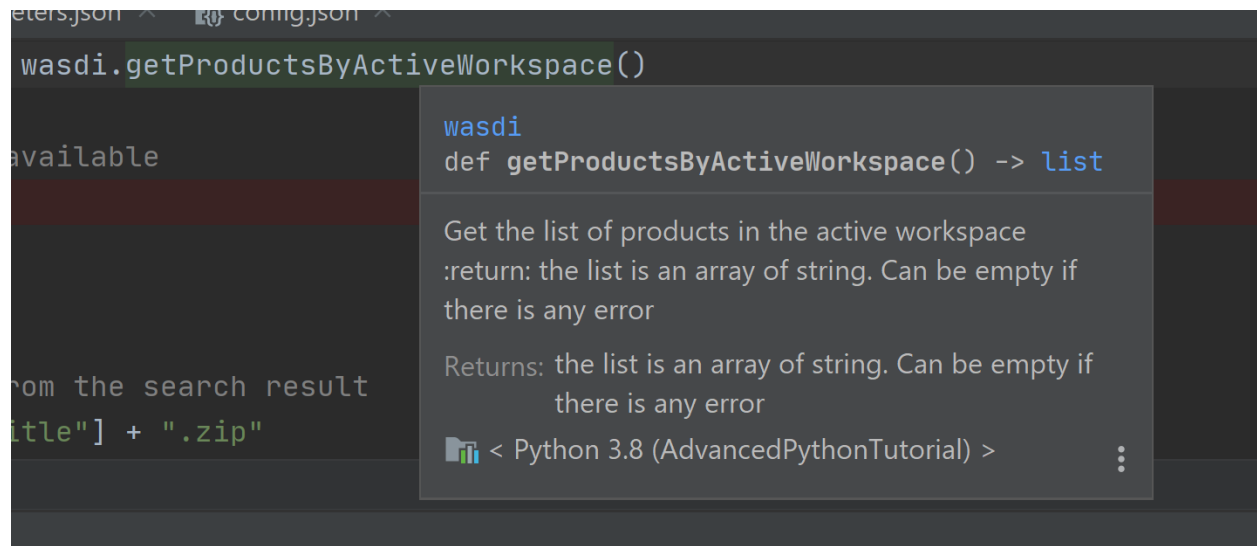
4.1.9 Step 3: import EO images in the workspace

Now we want to import selected images in the workspace.

```
# STEP 3: Import EO Images in the workspace
# Get the list of products in the workspace
asAlreadyExistingImages = wasdi.getProductsByActiveWorkspace()
# List of images not yet available
aoImagesToImport = []
# For each found image
for oImage in aoImages:
    # Get the file Name from the search result
    sFileName = oImage["fileName"]
    # If the file name is not yet in the workspace
    if sFileName not in asAlreadyExistingImages:
        # Add it to the list of images to import
        aoImagesToImport.append(oImage)
# If there are images to import
if len(aoImagesToImport) > 0:
    # Trigger the import of the images
    wasdi.importProductList(aoImagesToImport, sProvider)
    wasdi.wasdiLog("Images Imported")
```

Here we check, for each image, if it is not yet in the workspace. It's not strictly necessary, as it is handled by WASDI, but in this way we optimize the process: if an image is not already present, then we add it to the list of images to be imported, and finally we retrieve them from the provider.

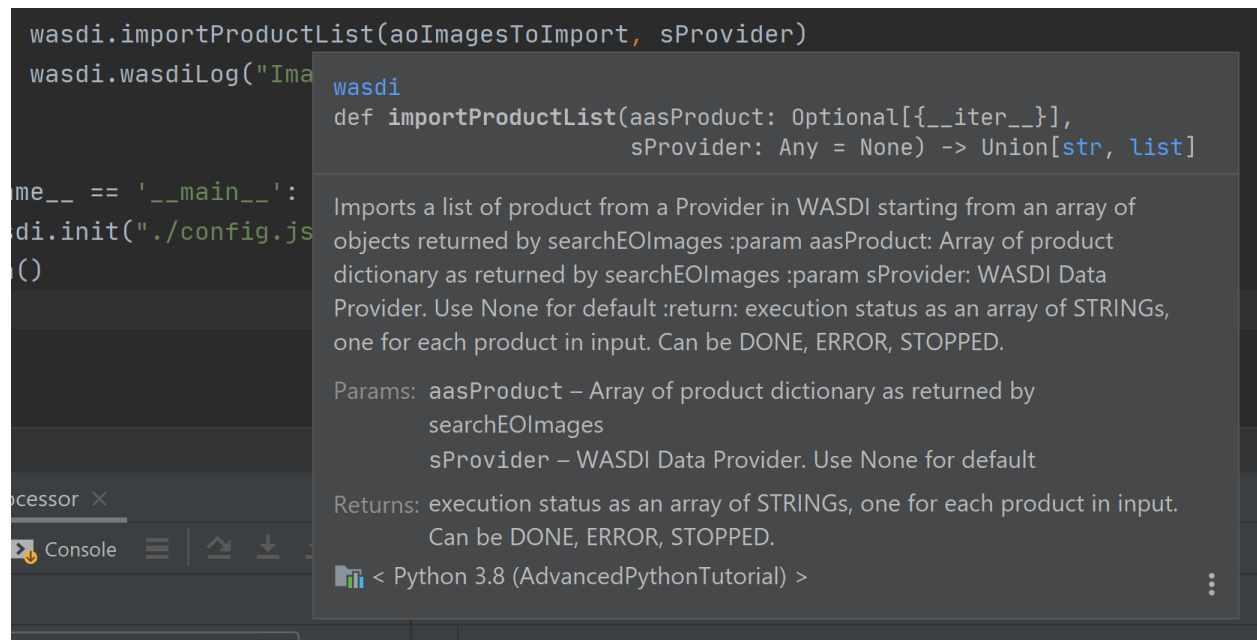
The method `wasdi.getProductsByActiveWorkspace` returns a string array with the names of files in the workspace.



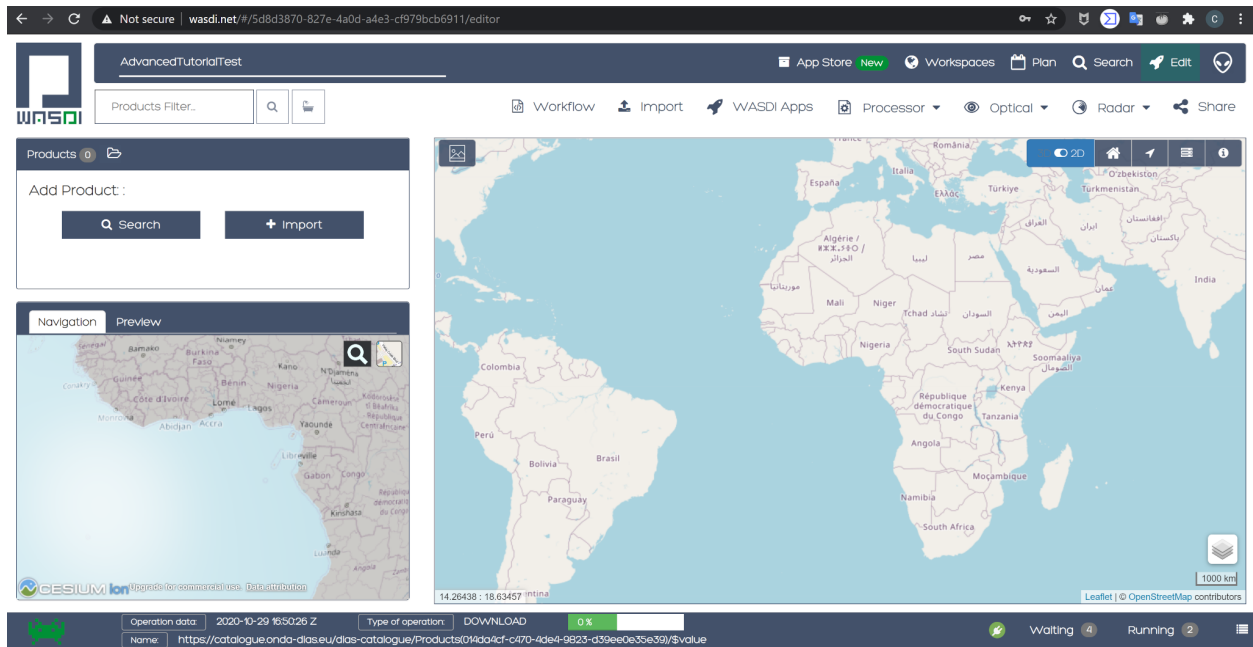
To compare the search results with the files in the workspace we need to obtain the file name. This can depend on the provider and on the image type but, for instance, with every Sentinel image it's easily reproduced with:

```
sFileName = oImage["title"] + ".zip"
```

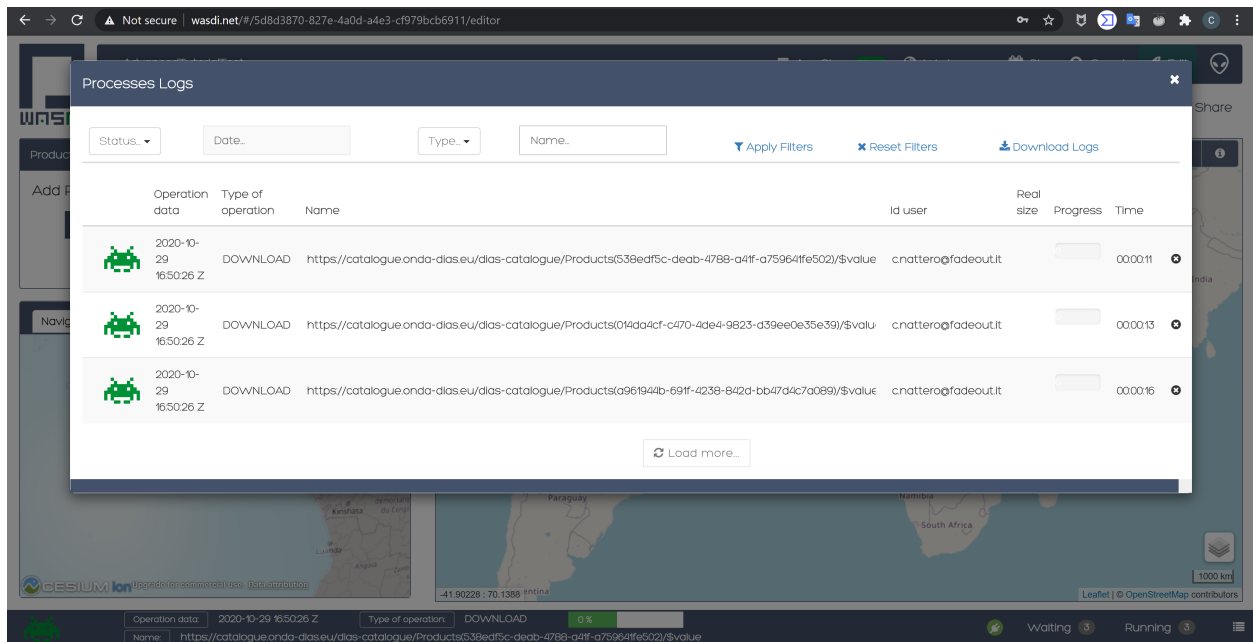
Next, `wasdi.importProductList` allows to import a batch of images from the specified provider.



WASDI will take control of the process and handle the task by queuing the requests in the background. The entire operation runs in the cloud. It will take a while, and during that time you will not be able to control the debugger. However, if you get back to your browser (did you remember to leave it open on the workspace editor?) you will be able to check the status of the operation.



If you click on the list icon in the bottom right corner of the screen, you can also view details for each operation in progress:



4.1.10 Step 4: create an 8-bit RGB GeoTIFF out of a Sentinel-2 image

Now we want to open one of those Sentinel-2 images, extract bands for Red, Green and Blue (RGB) and use them to construct an RGB GeoTIFF. We are going to use numpy and GDAL. GDAL is a set of tool for working with geo referenced images. You may need to install it in your environment. In that case, execute

```
pip install GDAL
```

in your terminal.

Make sure you have the latest version of Microsoft Visual C++ Build Tools installed, you can download the installer from this link <https://visualstudio.microsoft.com/it/thank-you-downloading-visual-studio/?sku=BuildTools&rel=16>.

If you get some error, try to install GDAL following these steps:

1. Download the wheel from (<https://www.lfd.uci.edu/~gohlke/pythonlibs/#gdal>) the website has many libraries that are very useful.
 - 1.1 Pay attention to the last part of the wheel to choose the correct one suitable for your PC, for example check if it is 64 or 32 bit.(For more information check: <https://realpython.com/python-wheels/#what-is-a-python-wheel>).
2. Open the Windows Terminal (CMD) and type “pip install (the path the wheel located in)”.

Add the following imports:

```
import numpy
import zipfile
import os
from osgeo import gdal
```

Now we need a way to extract the three bands from the Sentinel-2 image.

```
run()
```

Here in the following you can find the lines to add to the run method. Beware, there are two calls to two methods, extractBands and stretchBandValues, which will not work: we are going to implement them in a moment, keep reading. Here's the snippet:

```
# STEP 4: From the S2 image create a 8-bit RGB GeoTiff
# Get again the list of images in the workspace:
asAvailableImages = wasdi.getProductsByActiveWorkspace()
# Check if we have at least one image
if len(asAvailableImages) <= 0:
    # Nothing found
    wasdi.wasdiLog("No images available, nothing to do.")
    wasdi.updateStatus("DONE", 100)
    return

# Initialize the image to process as None
sImageToProcess = None

# Take the first S2 image
for sImg in asAvailableImages:
    if sImg.startswith("S2"):
        sImageToProcess = sImg
        break
```

(continues on next page)

(continued from previous page)

```

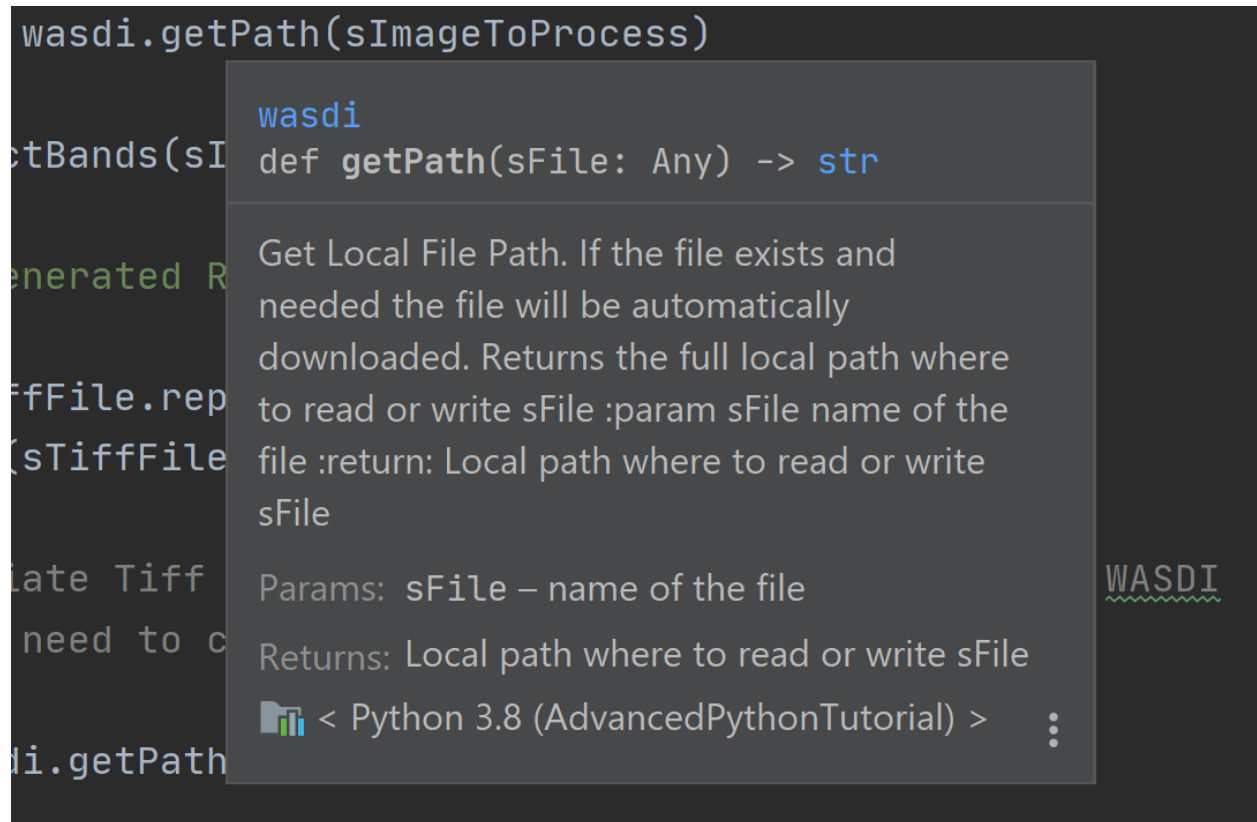
if sImageToProcess is None:
    wasdi.wasdiLog("Cannot find our S2 image in the workspace")
    wasdi.updateStatus("DONE", 100)
    return

# Get the local path of the image: this is one of the key-feature of WASDI
# The system checks if the image is available locally and, if it is not, it will
↳ download it
sLocalImagePath = wasdi.getPath(sImageToProcess)
sTiffFile = extractBands(sImageToProcess, sImageType)
wasdi.wasdiLog("Generated RGB Tiff: " + sTiffFile)
sOutputFile = sTiffFile.replace(".tif", "_rgb.tif")
stretchBandValues(sTiffFile, sOutputFile)
# Delete intermediate Tiff File: NOTE this has not been added to WASDI
# so there is the need to clean only the physical file
try:
    os.remove(wasdi.getPath(sTiffFile))
except:
    wasdi.wasdiLog("Error removing " + sTiffFile)
# Add the real output to the WASDI Workspace
# NOTE: here starts the opposite path: when running locally, WASDI will upload the file
↳ to the cloud
wasdi.addFileToWASDI(sOutputFile)

```

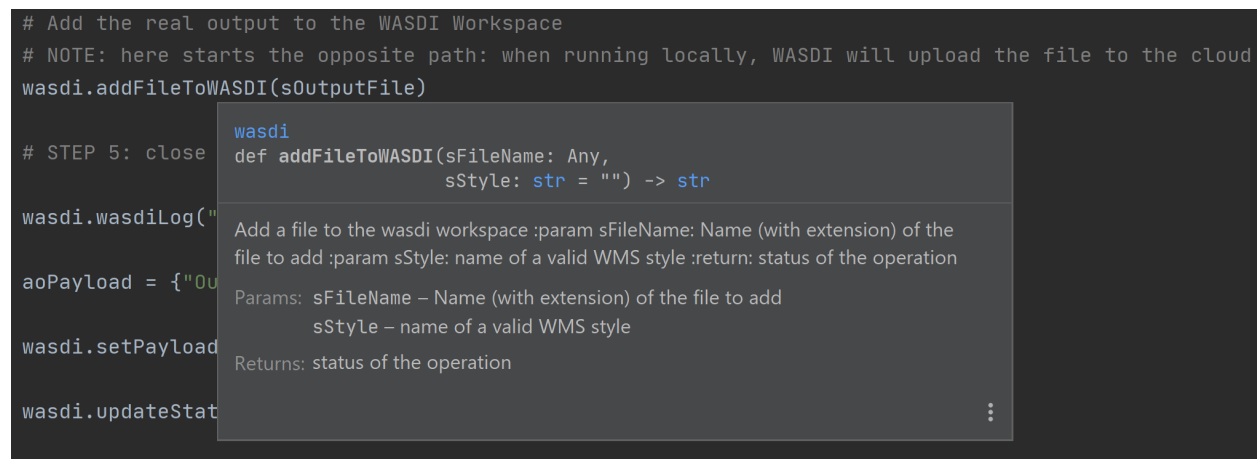
You noticed the call to `wasdi.getPath`: those are very important lines, since it's one of the basis on which we built the library. Up to now, every file path we used was relative (we actually used just the file name). The key concept here is that a file is not needed locally until it is open (think of it as a kind of lazy access). WASDI's `getPath` method is used to translate a file name to an absolute path. When the developer tries to access the file, WASDI understands if it exists in the platform or not and returns, depending on the case, either the local absolute path in which to create a file that does not yet exist, or the local absolute path from which it is possible to read the searched file. In this second case, the system automatically understands that the developer is currently working locally and downloads the required file automatically. As soon as the file is available, the control returns to the IDE, and the debug can proceed.

Note: in general, downloading file is not what we want and, once the processor will be deployed on the cloud, it is not going to happen any more: we will thus be able to process also large batches of EO data. However, as long as we are developing, we need to download some images, just to check that everything works as we expect it to work. The WASDI python library is smart enough to understand whether the code is running on our PC or in the cloud, and change behavior transparently and automatically.



The call to `addFileToWasdi` is worth a mention too. The method adds the file entry to the WASDI system so that it can be accessed and further used by WASDI. Again, this call has a double way of working: on the cloud, it simply adds the product to the WASDI data collection; when executed locally, the library realizes automatically that the file is missing on the platform and uploads.

Note: we do not like uploads either. However we decided to implement this functionality because being able to test our processor from end to end to is fundamental. So, uploads will take place only during development, whereas they will not be necessary when the processor will run on the cloud.



Next, we need to create the following two methods that we wish to call:

- `extractBands`
- `stretchBandValues`

extractBands

This method gets a collection of bands, and extracts them as a virtual GeoTIFF from the Sentinel-2 image, and finally creates a GeoTIFF with the extracted bands.

```
def extractBands(sFile, sImageType):
    try:
        sOutputVrtFile = sFile.replace(".zip", ".vrt")
        sOutputTiffFile = sFile.replace(".zip", ".tif")
        # Get the Path
        sLocalFilePath = wasdi.getPath(sFile)
        sOutputVrtPath = wasdi.getPath(sOutputVrtFile)
        sOutputTiffPath = wasdi.getPath(sOutputTiffFile)
        # Band Names for S2 L2
        asBandsJp2 = ['B04_10m.jp2', 'B03_10m.jp2', 'B02_10m.jp2']
        if sImageType != "S2MSI2A":
            # Band Names for S2 L1
            asBandsJp2 = ['B04.jp2', 'B03.jp2', 'B02.jp2']
        with zipfile.ZipFile(sLocalFilePath, 'r') as sZipFiles:
            asZipNameList = sZipFiles.namelist()
            asBandsS2 = [name for name in asZipNameList for band in asBandsJp2 if band in
↪name]

            asBandsZip = ['/vsizip/' + sLocalFilePath + '/' + band for band in asBandsS2]
            asOrderedZipBands = []
            for sBand in ['B04', 'B03', 'B02']:
                for sZipBand in asBandsZip:
                    if sBand in sZipBand:
                        asOrderedZipBands.append(sZipBand)
                        break
            gdal.BuildVRT(sOutputVrtPath, asOrderedZipBands, separate=True)
            # , options="-tr " + sResolution + " " + sResolution
            gdal.Translate(sOutputTiffPath, sOutputVrtPath)
            os.remove(sOutputVrtPath)
            return sOutputTiffFile
    except Exception as oEx:
        wasdi.wasdiLog(f'extractBands EXCEPTION: {repr(oEx)}')
    return ""
```

Sentinel-2 images contain Blue in band 2, Green in band 3, and Red in band 4. Here we extract them from the original file, save them into a virtual GeoTIFF (a .vrt file), and then we create a GeoTIFF. Finally, we can delete the virtual file, and return the name of the GeoTIFF RGB file we just created

stretchBandValues

Here, using numpy, we manipulate the bands. Add the following method to myProcessor.py:

```
def stretchBandValues(sOutputTiffPath, sStretchedOutputFile):
    oDataset = gdal.Open(wasdi.getPath(sOutputTiffPath))
    if not oDataset:
        wasdi.wasdiLog("Impossible to get Dataset from " + sOutputTiffPath)
        return ""
    [iCols, iRows] = oDataset.GetRasterBand(1).ReadAsArray().shape
    oDriver = gdal.GetDriverByName("GTiff")
```

(continues on next page)

(continued from previous page)

```

oOutDataFile = oDriver.Create(wasdi.getPath(sStretchedOutputFile), iRows, iCols,
                              oDataset.RasterCount, gdal.GDT_Byte, ['COMPRESS=LZW',
↪ 'BIGTIFF=YES'])
# sets same geotransform as input
oOutDataFile.SetGeoTransform(oDataset.GetGeoTransform())
# sets same projection as input
oOutDataFile.SetProjection(oDataset.GetProjection())
for iBand in range(oDataset.RasterCount):
    iBand += 1
    oBand = oDataset.GetRasterBand(iBand)
    if oBand is None:
        wasdi.wasdiLog("BAND " + str(iBand) + " is None, jump")
        continue
    adBandArray = numpy.array(oBand.ReadAsArray())
    adBandArray[adBandArray > 5000] = 5000
    adBandArray = adBandArray.astype(float)
    adBandArray *= 0.051
    adBandArray = adBandArray.astype(int)
    oOutDataFile.GetRasterBand(iBand).WriteArray(adBandArray)
    oOutDataFile.GetRasterBand(iBand).SetNoDataValue(0)
    oBand = None
# saves to disk!!
oOutDataFile.FlushCache()
wasdi.wasdiLog("Saved " + sStretchedOutputFile)

```

This method opens the tif file we just created, reads the bands as numpy arrays, and cuts each band empirically at a value of 5000, then scales their values down into [0, 255]. The file is saved to disk and its name is returned.

4.1.11 Step 5: close the WASDI processor

We're almost done! Add the remaining part to myProcessor.py:

```

# STEP 5: close the processor
wasdi.wasdiLog("Created output file " + sOutputFile)
aoPayload = {"OutputFile": sOutputFile}
wasdi.setPayload(aoPayload)
wasdi.updateStatus("DONE", 100)

```

Here we set a payload for the processor. The payload consists of the output parameters, i.e., a dictionary that can be retrieved later in form of a JSON object.


```
# Add the re
# NOTE: here
wasdi.addFil

# STEP 5: cL

wasdi.wasdiL

aoPayload =

wasdi.setPayload(aoPayload)
```

wasdi

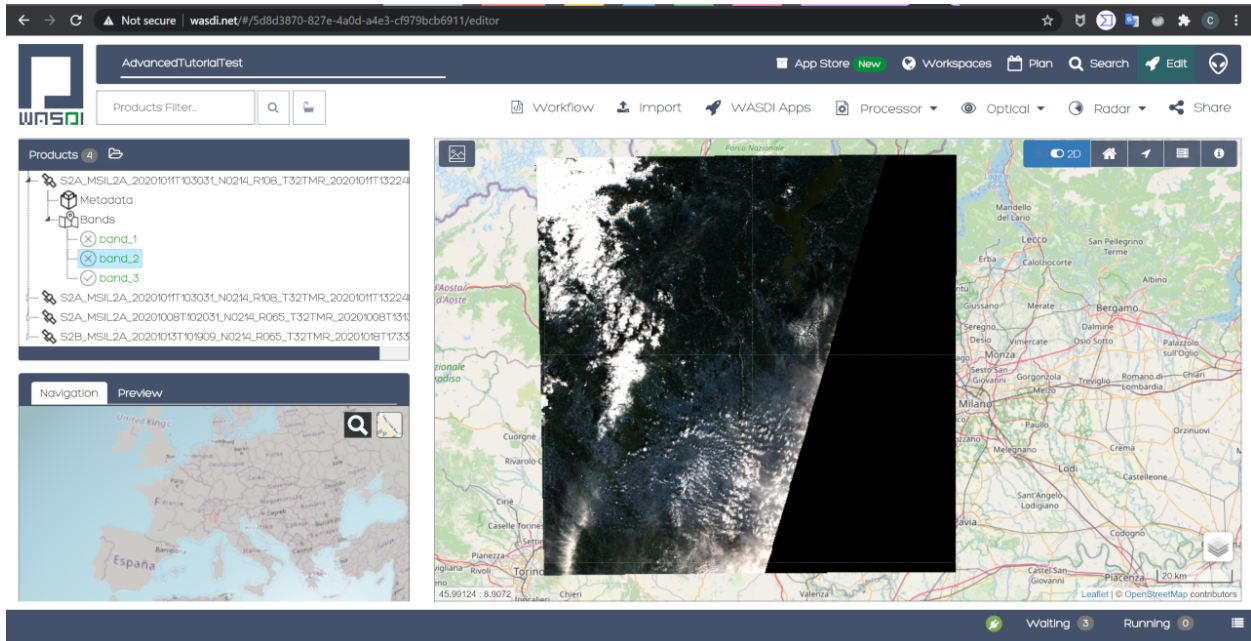
```
def setPayload(data: Any) -> None
```

Set the payload of the actual running process.
The payload is saved only when run on Server.
In local mode is just a print. :param data: data
to save in the payload. Suggestion is to use
JSON return None

Params: data – data to save in the payload.
Suggestion is to use JSON return
None

To retrieve that payload you can use `wasdi.getProcessorPayloadAsJson` and pass the `processID` as argument. You can obtain the process ID from the UI or programmatically: it's given in output when launching another processor. Finally, we set the status to `DONE` before closing the processing.

Once the processor is done, we can go to the web UI and open the final result:



To wrap up, you can download the complete code from here: [myProcessor.py](#)

4.1.12 Creating a help file

You can create a manual for those who are going to use your processor by adding a file called `readme.md` to your project. As you can see, it is a markdown file that, once the processor will be deployed, will be rendered to the users.

In the help, it's a good idea to describe what the processor does and how to use its parameters. This is an example you can copy and paste in your file:

```
# WASDI Advanced Python Tutorial
This processor searches for Sentinel-2 images and extract an RGB GeoTIFF from it.
## Parameters
```

Parameters are in this form:

```
{
  "BBOX": "45.9,8.5,45.7,8.7",
  "MAXCLOUD": "50",
  "DATE": "2020-10-25",
  "SEARCHDAYS": "20",
  "PROVIDER": "AUTO"
}
```

where:

- BBOX is the bounding box represented as a string with the format: "LATN,LONW,LATS,LONE"
- MAXCLOUD is an integer representing the maximum cloud coverage (percent)
- DATE is a date for the search
- SEARCHDAYS is the maximum number of days to search in the past, so the search will be performed on the BBOX and in the period [DATE - SEARCHDAYS, DATE], and for images with at most MAXCLOUD% cloud coverage
- PROVIDER is the data provider: use "AUTO"

Check your file locally, you're going to use it in a moment. If you wish, you can download it from here: [readme.md](#)

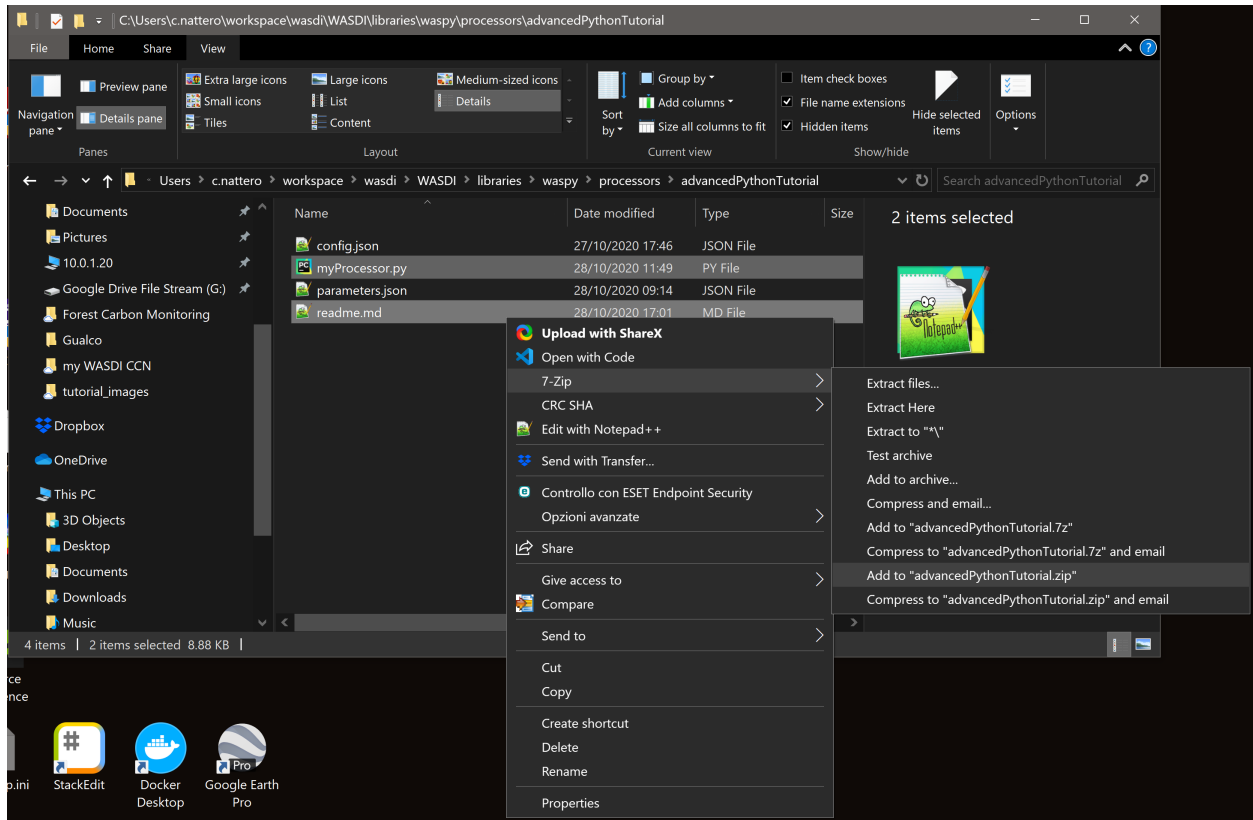
4.1.13 Deploy the processor on WASDI

Now go to the folder containing your processor, create a zip file containing only the following two files:

- `myProcessor.py`
- `readme.md`

Pay attention: `parameters.json` is not necessary, and it is definitely safer not to add `config.json`

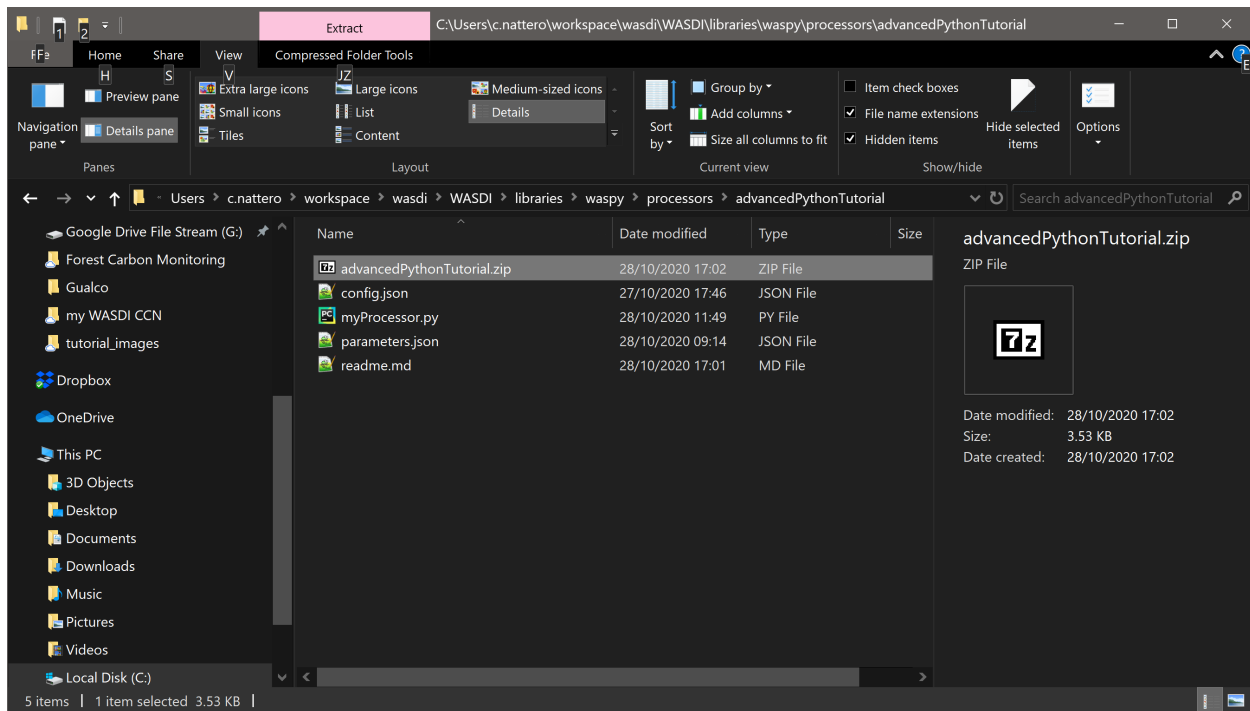
Let us stress the latter once more: do not include `config.json` in the zip!



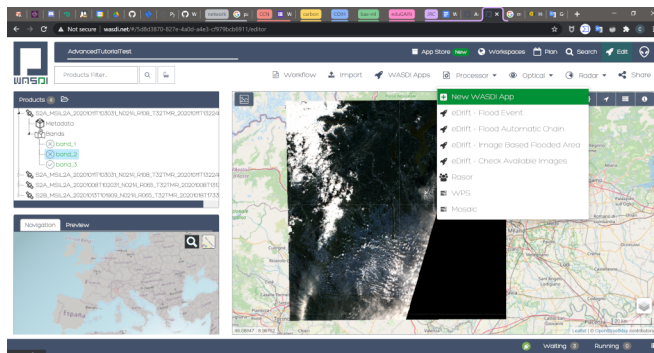
You may call the zip file `advancedPythonTutorial.zip` if you need a suggestion, but the name really makes no difference.

Note: in a more realistic situation, your processor would probably consist of several files, directory and additional ancillary data (e.g., a DTM); in such a case, be sure to:

- make the `run` method in `myProcessor.py` the entry point
- include every relevant file in the zip archive



Now go to the WASDI web UI, make sure you are in editing mode (i.e., you have a workspace open). Clic the Processor menu, clic New WASDI app.



A dialog opens:

WASDI APP

Processor Store Media Share UI

Name

Type

Drag and Drop here the zip file with your WASDI App or click to browse.

Short Description

TimeOut (min)

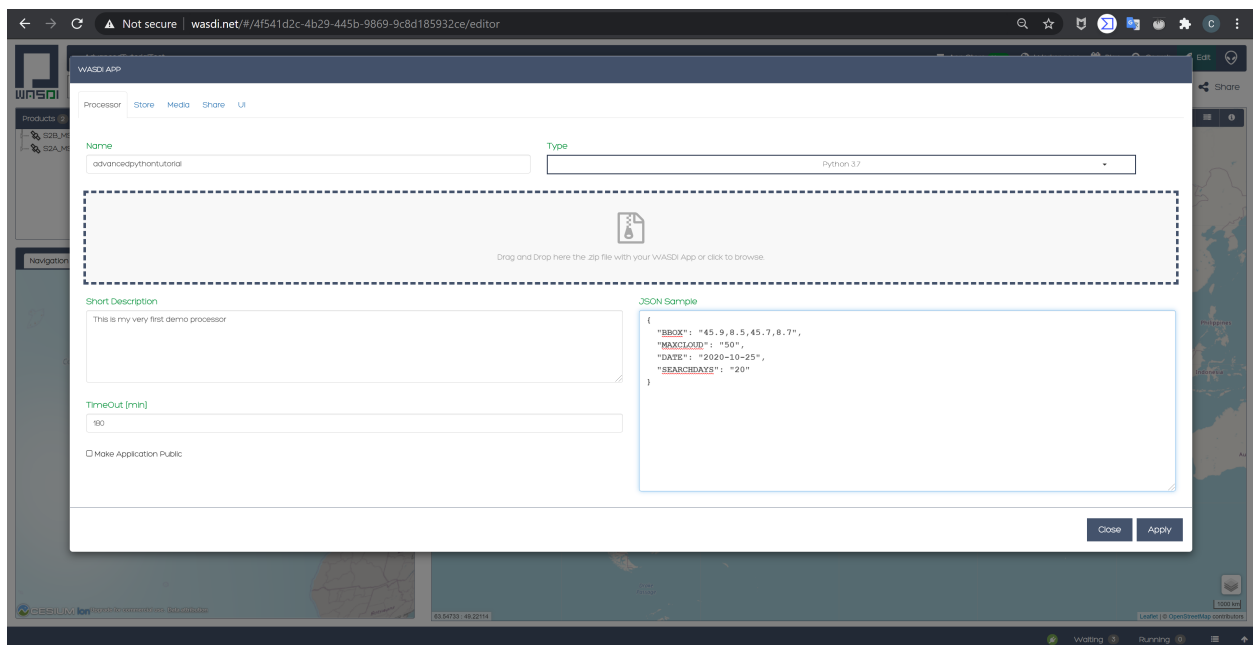
☒ Make Application Public

JSON Sample

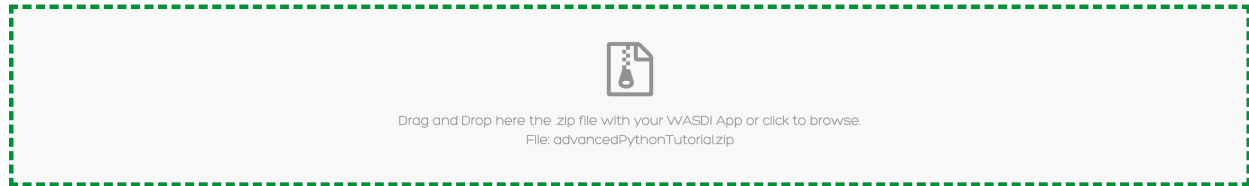
Close Apply

In the dialog:

- give your processor a name (e.g., advancedpythontutorial). It must be one small case string
- select Python 3.7 as Type
- write a short description, e.g. “This is my very first demo processor”
- leave the TimeOut with its default value (180)
- paste the content of your parameters.json into the JSON sample
- make sure you uncheck the Make Application Public box (yes, it’s definitely a nice processor, but we are going to have plenty of copies of it... ;-)

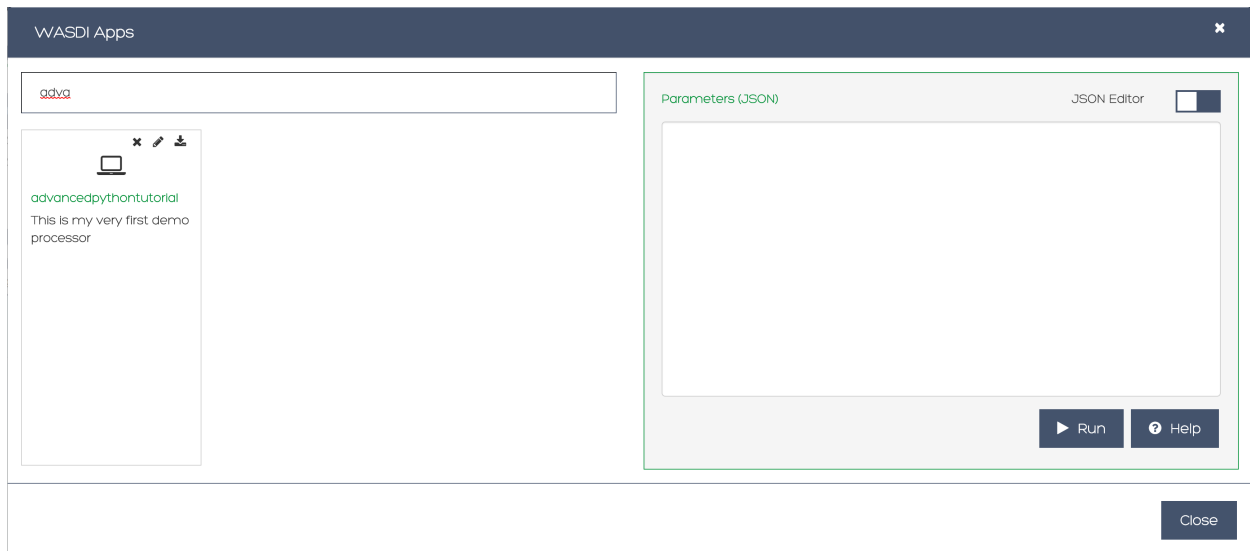


drag and drop your newly created zip file into the area for download

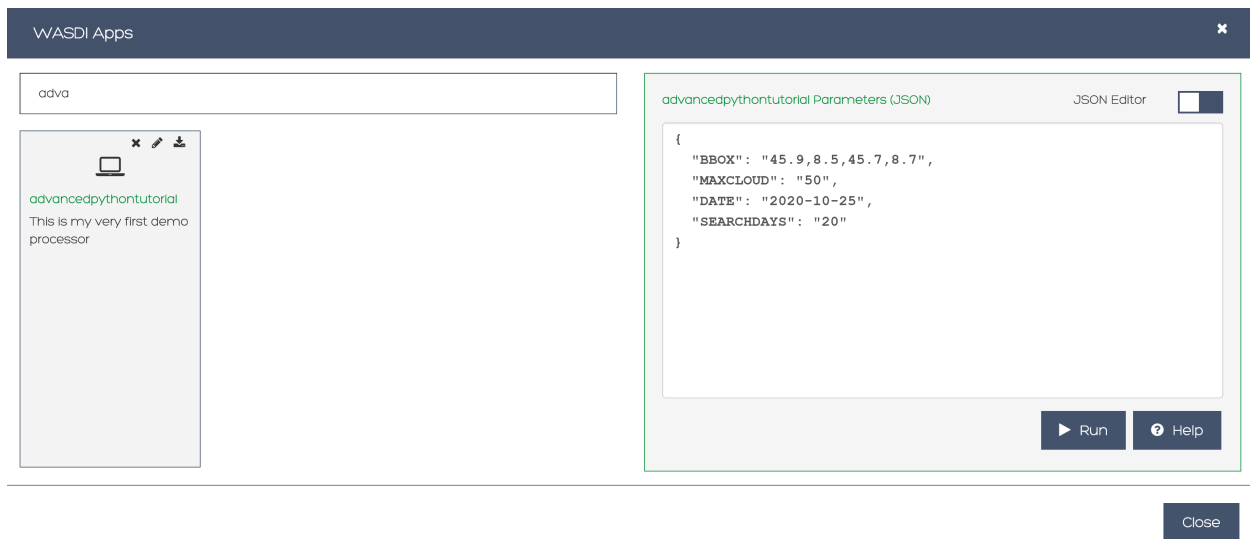


Are you done? Click Apply!

The processor is uploaded to WASDI and automatically deployed. Give it a moment and then click on WASDI Apps. Start writing the name of the processor to search for it.



Select it, and your JSON demo will be displayed. From here you can run it!



4.1.14 Turn the processor into an app on the marketplace

Well, well, you did great! Now it's time to let others use your processor (in a not too distant future, you will even be able to monetize your processor by selling its usage): enter the WASDI app store!

Go back to the apps, search for `advancedpythontutorial`, select it, and clic on the pencil icon to edit its properties. ...
 image:: ../_static/python_tutorial_images/editYourApp.png

Edit your app You will see that the dialog has some more other than the one we took care of. Now, we are going to see all of them in details:

- Processor
- Store
- Media
- Share
- UI

4.1.15 Processor tab

We already discussed its usage, but there are still some tweaks we can do here. If you ever needed to edit one or more of the files involved, simply make a zip containing just the files you need to modify, drag and drop it as usual, and click apply. Of course, you can always change any other propriety you wish, from here. Moreover, there are three cases in which you wish to click the Force refresh button:

- you added new pip packages. If you wish to use other packages, you need to write them down, one per line, in a text file called `pip.txt`. Add the file to the zip and deploy it
- you need additional system packages installed. If you need to install additional packages using `apt` (your code runs on a Ubuntu distro), add a text file called `packages.txt` and list the packages you need, one per line. As in the previous case: add the file to the zip and deploy it
- you updated the wasdi lib

4.1.16 Store

The screenshot shows the 'Store' tab in the WASDI APP interface. It contains several input fields and a list of categories. The 'Friendly Name' field is filled with 'advancedpythontutorial'. The 'Link' field is filled with 'Link'. The 'E-Mail' field is filled with 'Email'. The 'On Demand Price' field is filled with '0'. The 'Subscription Price' field is filled with '0'. The 'Show Application in the Store' checkbox is checked. The 'Long Description' text area contains the text 'Oh, it's such an amazing processor, you cannot really do without'. The 'Category' list includes 'Flood', 'Population', 'Optical' (checked), 'Impacts', 'Vegetation', 'Forest Fires', 'Urban Footprints', 'SAR', and 'GIS'. At the bottom right, there are 'Close' and 'Apply' buttons.

Here it's where you can choose to show your application on the marketplace. You can give it a more friendly name, add a link and an email address for the users to reach out to support, add prices for the on demand and subscription-based usage modes, write a longer and nicer description, flag some categories, and, above all, flag the box to show your application on the app store!

4.1.17 Media

The screenshot shows the 'Media' tab in the WASDI APP interface. It contains a section for 'Application Logo' with a preview image and a text area for 'Add an Image to the App Gallery (Available: 6)'. There are two dashed boxes for dragging and dropping images, labeled 'Drag and Drop here the Application Logo' and 'Drag and Drop here a new Application Image'. At the bottom right, there are 'Close' and 'Apply' buttons.

Here you can add a logo and an image for your application

4.1.18 Share

WASDI APP

Processor Store Media **Share** UI

Share Application: Share

Actual Users:

Close Apply

You can add a user to your application. Think of a colleague: you both will be able to contribute to the same processor.

4.1.19 UI

WASDI APP

Processor Store Media Share **UI**

App UI - Add elements (take care of commas):

Tab	<pre>{ "tabs": [] }</pre>
Render As Strings	
Textbox	
Dropdown	
Select Area	
Number Slider	
Date	
Bool	
Product Combo Box	
Search EO Image	
Hidden Field	

Close Apply

This is where magic happens again: the WASDI interface generator! Using a JSON you can describe a web user interface, which will be generated automatically for you. You can fiddle around and you will learn how to use, but let's make the UI for our processor together.

- Click to put the cursors inside the curly brackets, before “tabs”, then click Render As Strings
- move inside the square brackets after “tabs” and use the Tab button. Name it “Input”. Remove the trailing comma (or the JSON will not be valid) before the last closing square bracket. Click between the square brackets of your newly created tab.
- Use the Date button. Call the parameter “DATE” and mark it required. Click after the comma at the end of the DATE parameter block
- Use the Number slider. Call the parameter “SEARCHDAYS”, mark it required. Give it a description. Give it boundaries and a default value (e.g., 5-20, default: 10).
- Use the Select Area button. Call the parameter “BBOX”. Mark it required.
- Use the Number slider. Call the parameter “MAXCLOUD”. Mark it required. Bounds are 0 and 100. Default: 30. Give it a description.
- Check there are no trailing commas!

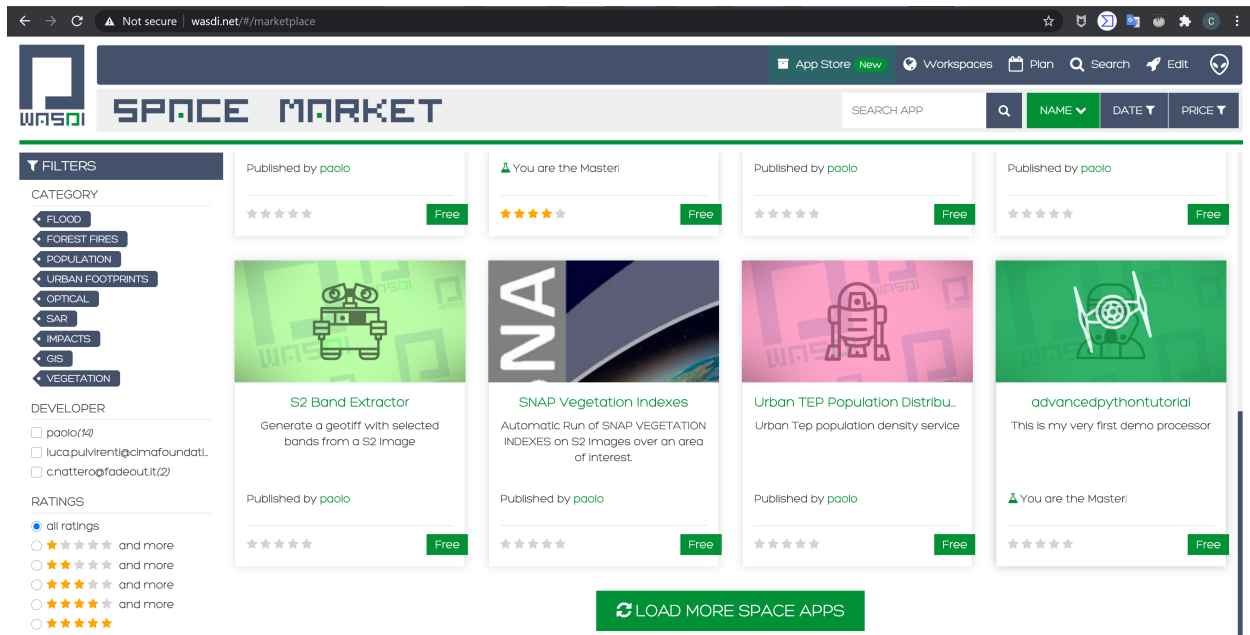
This is what it should look like in the end (you can copy paste this, if you did not manage to build your own):

```
{
  "renderAsStrings": true,
  "tabs": [{
    "name": "Input",
    "controls": [{
      "param": "PROVIDER",
      "type": "dropdown",
      "label": "Data Provider:",
      "default": "AUTO",
      "values": [
        "AUTO",
        "EODC",
        "SOBLOO",
        "CREODIAS"
      ]
    }, {
      "param": "DATE",
      "type": "date",
      "label": "Date",
      "required": true
    }, {
      "param": "SEARCHDAYS",
      "type": "slider",
      "label": "Days to search in the past",
      "default": 10,
      "min": 5,
      "max": 20,
      "required": true
    }, {
      "param": "MAXCLOUD",
      "type": "slider",
      "label": "Max cloud cover (percent)",
      "default": 30,
      "min": 0,
      "max": 100,
      "required": true
    }, {
      "param": "BBOX",
      "type": "bbox",
      "label": "Bounding Box",
      "required": true
    }
  ]
}
]
```

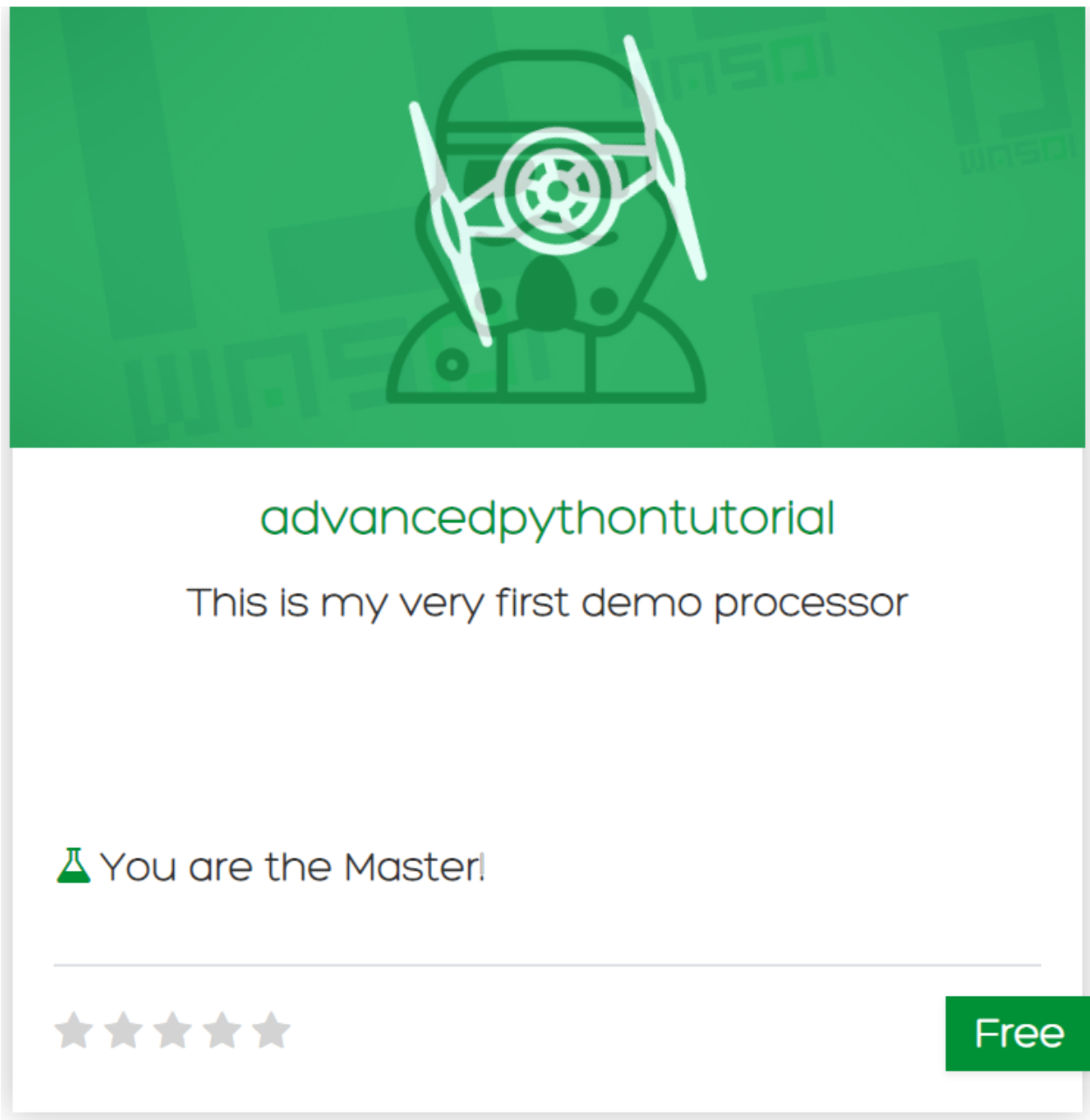
If you wish, you can download the UI description as a JSON file from here: [UI.json](#)

4.1.20 The app store

Now go to the app store, and try to use your app from there. To find it, you can filter using your user, or search using the name.



Once you opened the app presentation page,



The app in the store open the application to test it for real.

[← Back to the Future](#)

advancedpythontutorial

★★★★★ 0 reviews

▲ You are the master: [Edit](#)

🔍 CATEGORIES

• OPTICAL

FREE

📖 DESCRIPTION

This is my very first demo processor

Open Application



🔍 APPLICATION DETAILS

Publisher: cnattero@fadeout.it

Contact:

WebSite:

Publication Date: [10/28/2020](#)

Last Update: [10/28/2020](#)

Purchased: 1

💬 REVIEWS

★★★★★

0 total reviews

★★★★★ 0

★★★★★ 0

★★★★★ 0

★★★★★ 0

★★★★★ 0


★★★★★ 0

NOT FAIR TO AUTO REVIEW

NO MORE REVIEWS


There you can see the interface you just described. Use it, and to see if it works as expected

[Back to the Future](#)



advancedpythontutorial
 Publisher: cnattero@fadeout.it
[MORE DETAILS](#)

☒ New Workspace
☐ Open Workspace




Input

Help

History

JSON

Date



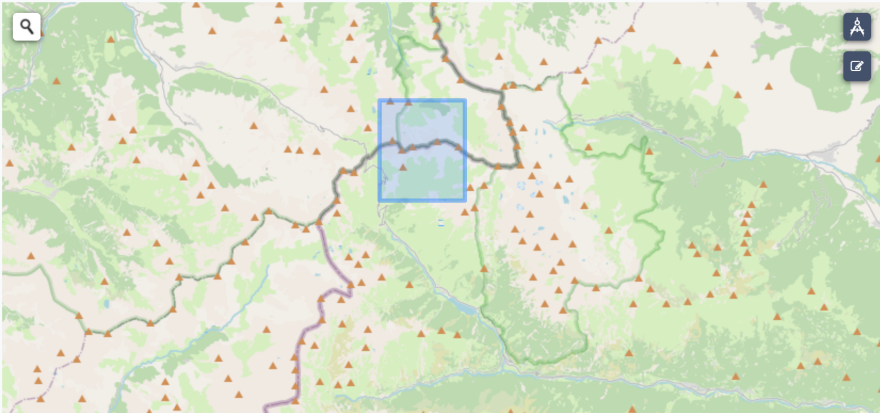
Days to search in the past

5

19


20

Bounding Box




before running the processor, you can also check the JSON that will be generated automatically with the parameters your processor need:

[Back to the Future](#)



advancedpythontutorial
 Publisher: cnattero@fadeout.it
[MORE DETAILS](#)

☒ New Workspace
☐ Open Workspace



Input

Help

History

JSON

```
{
  "DATE": "2020-10-30",
  "SEARCHDAYS": "19",
  "BBOX": "44.71,6.99,44.66,7.04",
  "MAXCLOUD": "30"
}
```

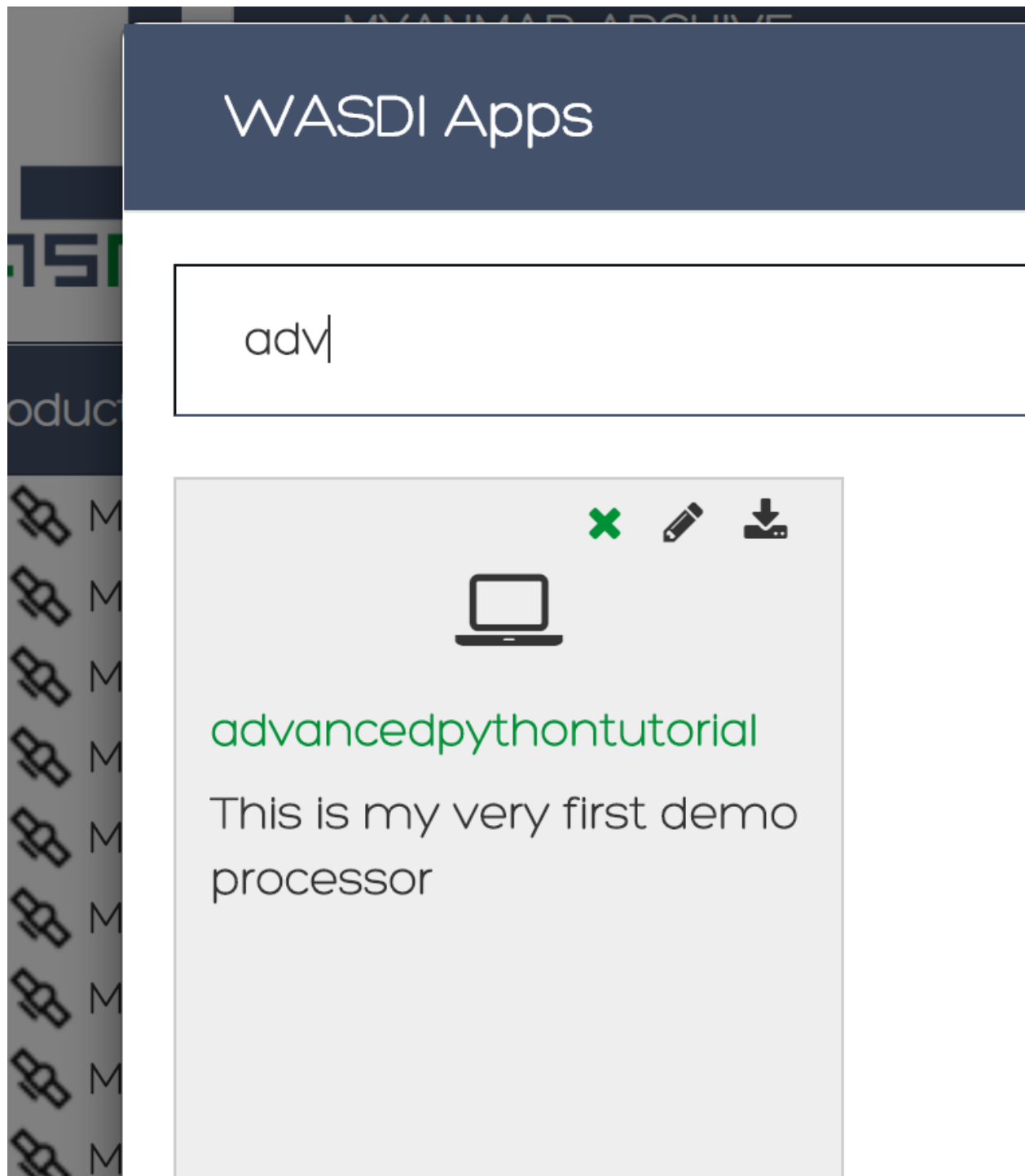
Feel free to play with your processor and tweak it.

4.1.21 Delete your app

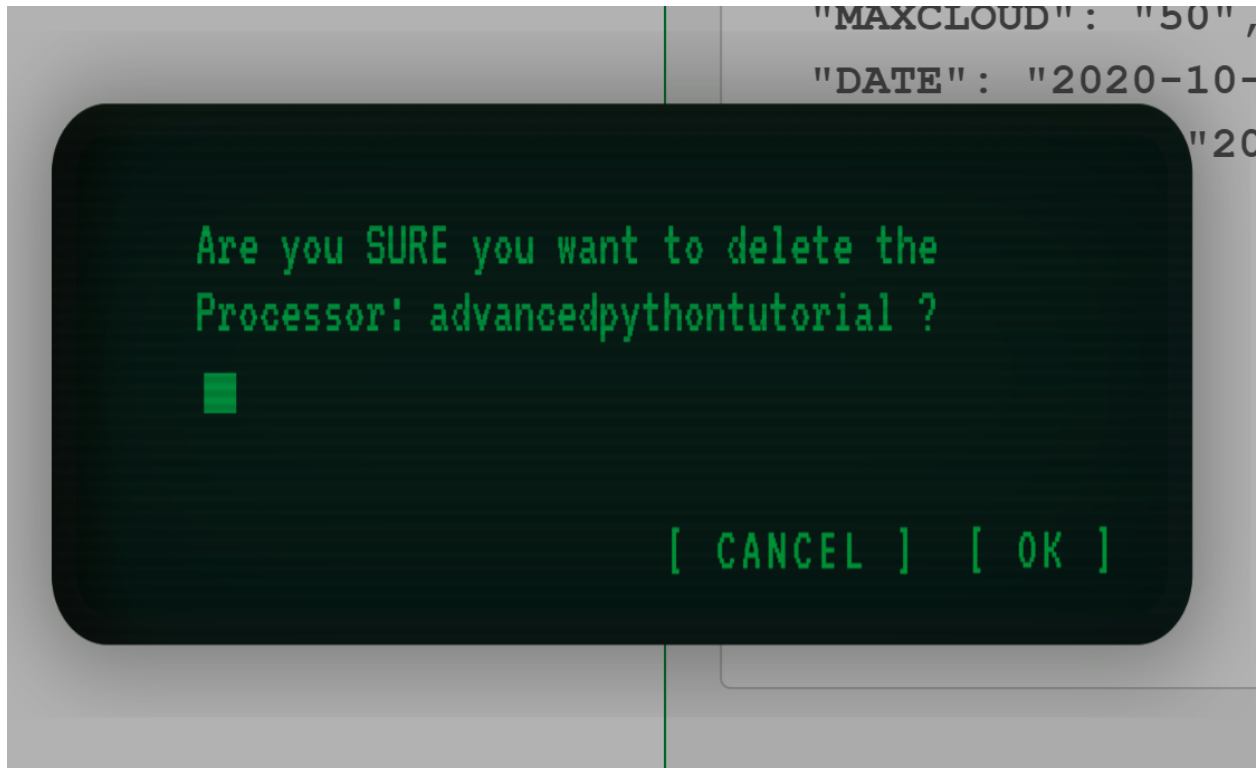
Are you done? Here you are two sad facts:

- this processor is not a milestone in the history of remote sensing
- Santa Claus does not exist

We cannot change the second, but we can solve the first by deleting the processor: got to the editor (i.e., open a workspace), search for your app in the WASDI apps menu, clic on the x symbol to delete the app



That's how you delete a processor Click OK to confirm that you want to delete it



Confirm processor deletion That's it, now you know how to manage the entire lifecycle of a WASDI app!

Have fun, and let us know your thoughts

4.2 Jupyter Notebook Tutorial

This tutorial will guide you through the creation and usage of [Python Jupyter Notebooks](#) in WASDI.

The (toy) problem we are addressing is retrieving and visualizing data. We will use Sentinel-5p products because they are small and quick to import, open and visualize, but the procedure works with any type of product supported by WASDI.

The tutorial follows the structure of a webinar we did for ESA:

<https://youtu.be/eNkWPD7nehY>

4.2.1 Prerequisites

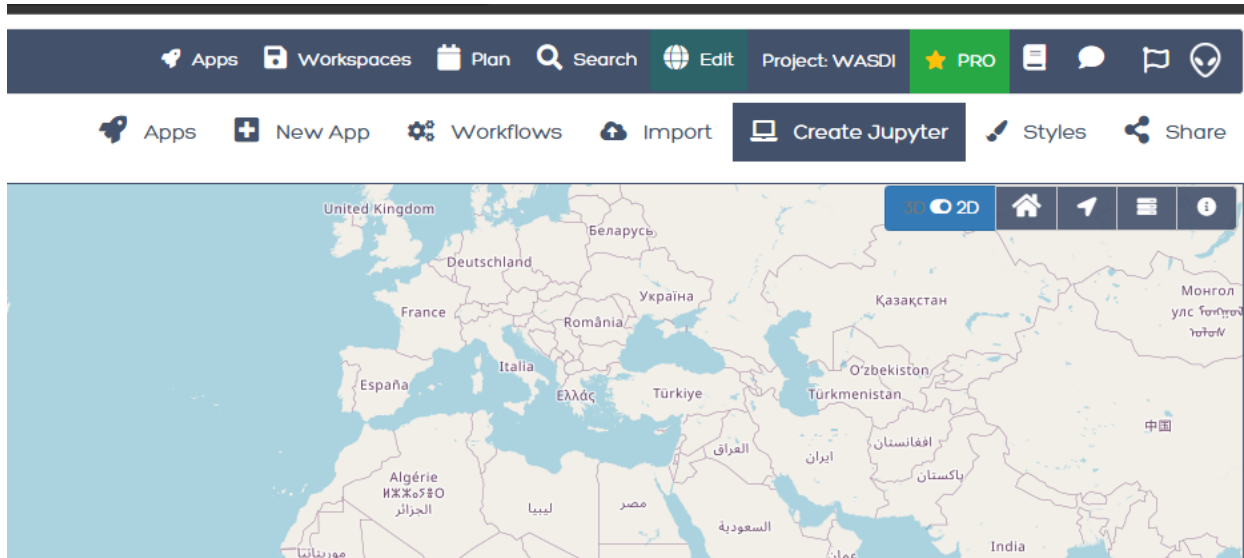
You will need:

- a valid WASDI user
- a workspace

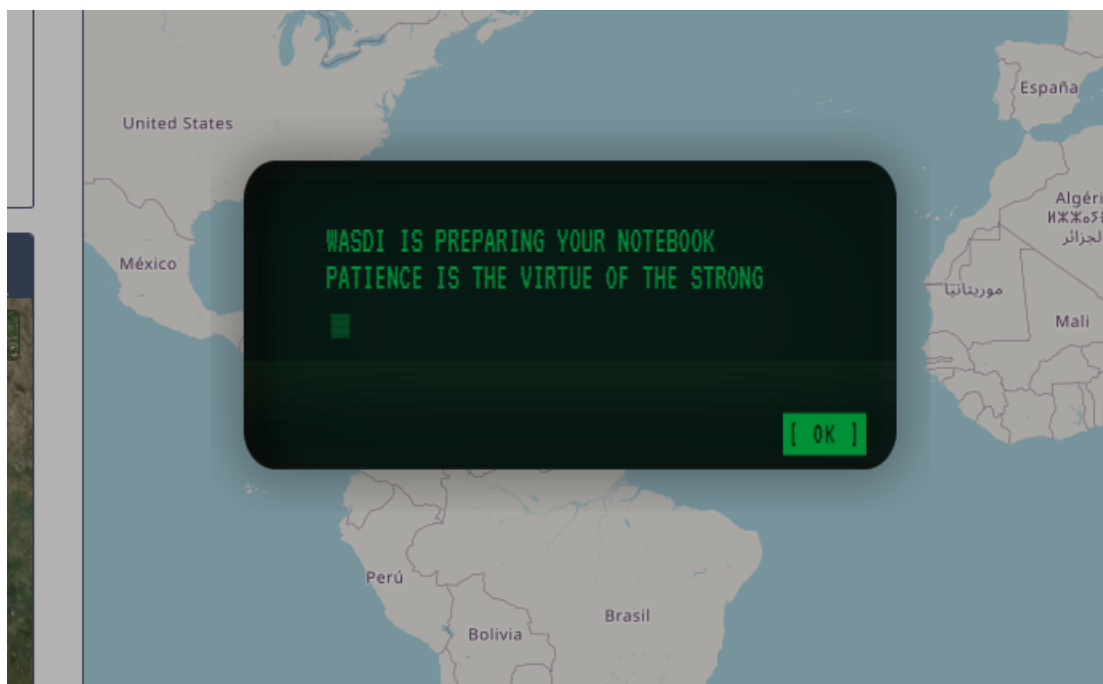
You don't have them? Then [go get them!](#)

4.2.2 Create and Open Jupyter Lab

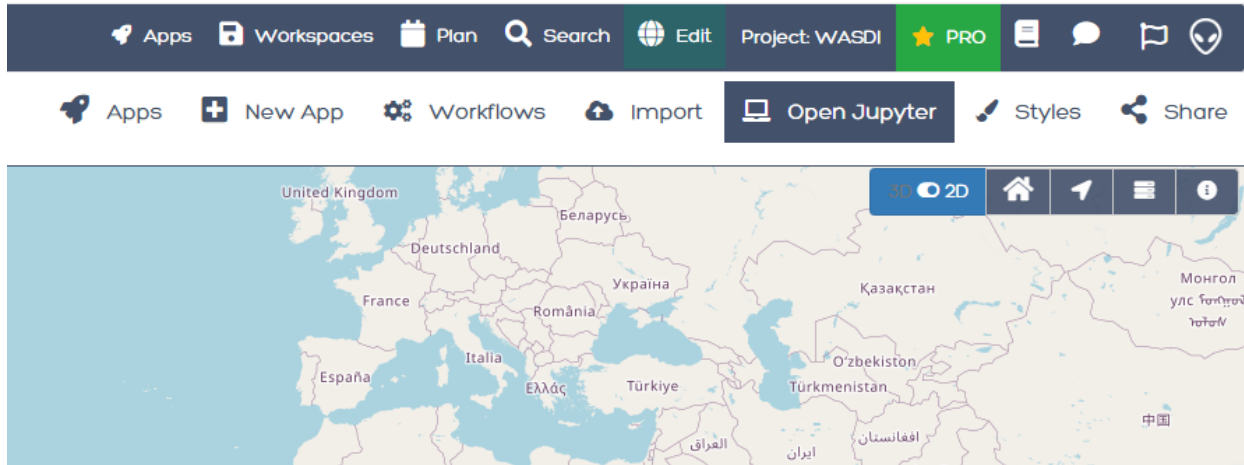
Open the workspace and once you are in the editor, click on the “Create Jupyter” button:



An alert will invite you to wait for it to be ready. Do it: click OK, but wait a moment



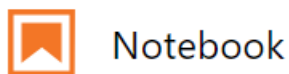
Jupyter won't open automatically. If you pay attention, you will notice that the button changed into “Open Jupyter”



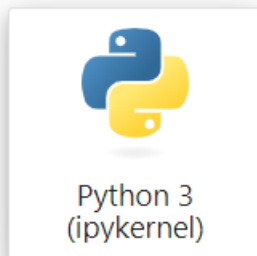
Click on it, and Jupyter Lab will open in another tab of the browser

4.2.3 Get started coding

Let's create a jupyter: click on the icon to create a new Python 3 (ipykernel) notebook:



Notebook



We will use *matplotlib* to generate visualization, so let's install it. In the first cell type:

```
!pip install matplotlib
```

You can execute it.

Next, create a new cell. Let's import some packages: type the following, and run it

```
from datetime import datetime
from datetime import timedelta
import numpy
from osgeo import gdal
import matplotlib.pyplot as plt
import wasdi
```

We can now start the WASDI Python library:

```
wasdi.init()
```

Run the cell. Pay attention to the output and enter your username and password when requested. The library will then proceed to open the current workspace and confirm that the session is valid.

Note: If you don't get a valid session, you probably inserted the wrong username and password. No problem: re-execute the cell and try again.

Let's now see how to log to WASDI, create a new cell, type the following code and run it:

```
# this is how we log a line in WASDI
wasdi.wasdiLog("Welcome to tutorial on Jupyter Notebooks")
```

Next, we will set up a parameter dictionary. Normally, you wouldn't do this in a WASDI application, but rather read the parameters from input. However, to learn how the library works, let's do this preparatory step. Create a cell and type the following (read the comments to understand what's going on):

```
# define input parameters (just for this exercise: normally parameters are fed from the
→user)

# In any geospatial query in WASDI, we will need:
# 1. an AoI in the form of a bounding box
# 2. a time interval, in the form of a start and an end date
# 3. collection-specific parameters

# begin with a bounding box
oInputParameters = {
    "BBOX": {
        "northEast": {
            "lat": 51.0,
            "lng": 7.7
        },
        "southWest": {
            "lat": 50.0,
            "lng": 6.5
        }
    }
}

# Define a time interval
oEndDay = datetime.today()
oStartDay = oEndDay - timedelta(days=2)

# stringify the dates
oInputParameters['endDate'] = oEndDay.strftime("%Y-%m-%d")
oInputParameters['startDate'] = oStartDay.strftime("%Y-%m-%d")

# print the params so far
print(oInputParameters)

# now let's define search parameters specific for this collection:
```

(continues on next page)

(continued from previous page)

```
# it's Sentinel-5p, so let's look for a pollutant: NO2
searchParams = {
    'producttype': 'L2__NO2___'
}

# save parameters in the WASDI session
# (normally we would not do this, WASDI would give us the parameters provided by the
↪ user)
wasdi.setParametersDict(oInputParameters)
```

Now let's see how we can read parameters from within a WASDI application

Note: From within the code of a WASDI application, we don't know **how** the parameters were fed to the application. Several ways are possible: if your code is running locally, then you may have loaded a JSON file, if it's running in the cloud, then the parameters might have been passed through the web UI, or maybe they have been passed programmatically by the calling code. However, we can read the parameter dictionary items with `getParameter`, or the entire parameter dictionary using `getParametersDict`

Create a new cell, input this code, and run it:

```
# Let's see how we read parameters from WASDI

# Read Bounding Box(LATN, LONW, LATS, LONE)
# the second string is the (optional) default value
sBbox = '51.0,6.5,50.0,7.7'
oBbox = wasdi.getParameter('BBOX', sBbox)

# did you notice we specified the default value as a string?
# If your python-sense tickled you here, good, you paid attention!
# In the previous cell, we specified the BBOX as a dictionary, and here
# it's a string, so? No problem! WASDI supports both formats in queries

# Let's log it
wasdi.wasdiLog(f'Bounding box: {oBbox}')

# Read time interval
sStartDate = wasdi.getParameter('startDate')
sEndDate = wasdi.getParameter('endDate')
print(f'Interval: {sStartDate} - {sEndDate}')

# let's log one line
wasdi.wasdiLog(f'Searching Sentinel-5p product for time period: {sStartDate} - {sEndDate}
↪')
```

Let's do the search now. Create a new cell, type the following code, run it:

```
wasdi.wasdiLog('Search for available Sentinel-5 products')

aoSearchResults = wasdi.searchEOImages('S5P', sStartDate, sEndDate, oBoundingBox=oBbox,
↪ aoParams=searchParams)

wasdi.wasdiLog(f'Found {str(len(aoSearchResults))}')
```

The results are stored in a list, and the last instruction will print just the number of results. Now let's explore the first one in a new cell:

```
# let's explore the results of the search
# pick the first one:
if len(aoSearchResults) > 0:
    print(aoSearchResults[0])
```

Run the cell: you will see that each result is a dictionary with some items, including the conventional name, the footprint, and a link for importing it from the data provider. Let's import it, and don't worry, WASDI will handle it. Create a new cell and paste this code in:

```
# import one product

if len(aoSearchResults) > 1:
    # as an example, import only the first one
    wasdi.importProduct(aoSearchResults[0])

print('Product imported')
```

Run the code to import the product. As an exercise, you can import them all. One way would be to loop through all the results. A more efficient way, is to use the `importProductList` method

Now create another cell, and let's discover how to run a WASDI processors programmatically. Put this code in a new cell, and run it:

```
wasdi.wasdiLog('Convert S5 product to GeoTIFF')

# pick just the first product returned from the search
oFoundProduct = aoSearchResults[0]

# we will call a processor named s5_2_tiff
# which is already deployed in WASDI

# Prepare the inputs for the processor
aoInputs = {}
aoInputs["S5Image"] = oFoundProduct["fileName"]
aoInputs["Band"] = "nitrogen_dioxide_tropospheric_column"

# call the processor...
wasdi.wasdiLog('starting the processor')
sProcessId = wasdi.executeProcessor("s5_2_tiff", aoInputs)
wasdi.wasdiLog(f'waiting for process {sProcessId} to complete')
# and wait for it to complete
sStatus=wasdi.waitProcess(sProcessId)
wasdi.wasdiLog(f'Process {sProcessId} completed in status {sStatus}')
```

Processors, in WASDI, have a payload, which is a way of saving textual data. It's a JSON object. Let's see what `s5_2_tiff` saved in its payload: create a new cell, paste this code in it and run it:

```
# read the payload
oPayload = wasdi.getProcessorPayload(sProcessId, True)

if oPayload is not None:
```

(continues on next page)

(continued from previous page)

```
print(f'Payload saved for the process:\n{oPayload}')
sTiffFile = oPayload["output"]
```

As an exercise, try to convert all imported files to tiff. Check the payloads of the processors you execute. Finally, remove imported files (.zip) from workspace and keep just the tiffs you extracted with *s5_2_tiff*

Finally, let's have a look at the band. Create a final cell, copy this code there, run it:

```
# let's see the band

if oPayload is not None:
    sPath = wasdi.getPath(sTiffFile)

    dataset = gdal.Open(sPath)
    band1 = dataset.GetRasterBand(1)
    b1 = band1.ReadAsArray()
    b1[numpy.where(b1 > 1)] = numpy.nan #no data is 9.96921e+36

    f = plt.figure()
    plt.imshow(b1)
    plt.savefig('Tiff.png')
    plt.show()
```

What happened here? *wasdi.getPath(sTiffFile)* returns the path to the GeoTIFF file we just created. In general, it returns the path to the product you pass as argument.

Note: If your code is running locally on your PC, the library realises if the product is missing and in that case downloads it from WASDI to a folder on your hard disk. That means that the first time you will incur a significant delay, but from the second run on it will be faster.

Then the following code opens the band with GDAL as a dataset, turns it into a numpy array, and then replaces invalid values with *NaN*. Finally, the last 3 lines plot the data with *matplotlib*

4.3 Python Landsat Tutorial

4.3.1 Requirements

This tutorial is designed to show how to work with Landsat 8 files in WASDI. Details of the Landsat 8 mission and/or guidelines on how to configure your own environment are out of the scope of this tutorial. In this tutorial we use PyCharm as a free Python Development tool, but the code can be executed on every different Python environment.

Note: This tutorial requires gdal working in your python env: we know this can be tricky.

For Windows 10, we suggest following this tutorial:

<https://opensourceoptions.com/blog/how-to-install-gdal-for-python-with-pip-on-windows/>

The key is that here:

<https://www.lfd.uci.edu/~gohlke/pythonlibs/#gdal>

you can find different pre-build GDAL “images” that can be installed directly asking pip to use the file.

4.3.2 Overview

In this tutorial, we will learn how to work with Landsat 8 Images in WASDI. The tutorial will implement a processor that takes as input:

- Name of a Landsat 8 file
- List of Bands to extract, by default B5 and B4
- Resolution to use

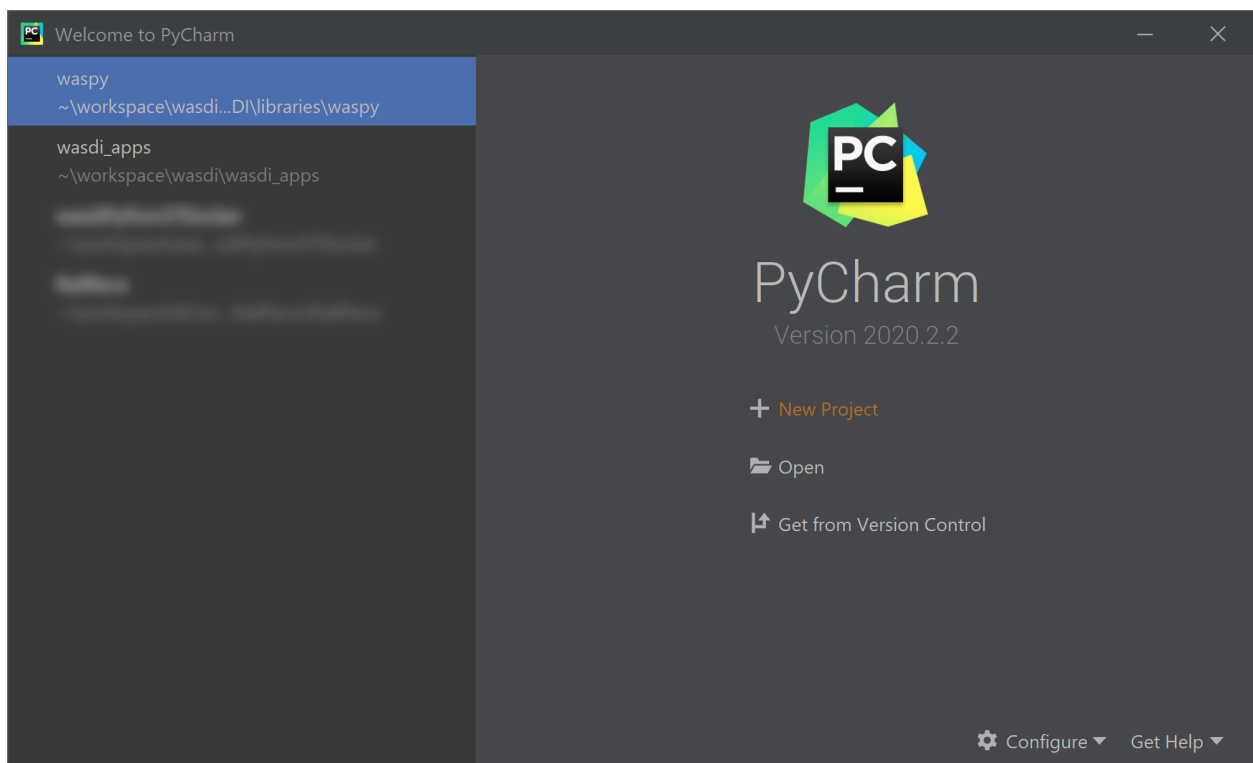
The processor will open the Landsat image, create a tiff with only the selected bands, reprojected to the desired resolution, and then it will calculate NDVI, assuming that the first extracted band is NIR and the second is RED.

The equation to compute NDVI is: $[NIR-RED]/[NIR+RED]$.

Note: It is mandatory that at least 1 Landsat image is imported in the workspace (using the WASDI web interface, i.e. Search) BEFORE running this processor. Check [Wasdi Web Platform access and basic usage](#) for more general info on this.

4.3.3 Setup

Open PyCharm and start a new project.



Name it “Landsat8Tutorial” (or however you wish, just remember to be coherent). You may wish to create a new virtual environment or use an existing one. Uncheck the option for creating a “main.py” welcome script (or, at least, remember to delete it later on).

Let’s install the library we need. In the terminal write:


```
pip install wasdi
```

and hit enter

```

AdvancedPythonTutorial - config.json
Project
  AdvancedPythonTutorial
    config.json
    myProcessor.py
  External Libraries
Terminal
  Local
  Microsoft Windows [Version 10.0.17763.1518]
  (c) 2018 Microsoft Corporation. All rights reserved.

  (AdvancedPythonTutorial) C:\Users\c.nattero\PycharmProjects\AdvancedPythonTutorial>pip install wasdi
  Collecting wasdi
    Downloading wasdi-0.5.1.tar.gz (33 kB)
  Collecting requests
    Downloading requests-2.24.0-py2.py3-none-any.whl (61 kB)
    | 61 kB 4.8 MB/s
  Collecting urllib3!=1.25.0,!=1.25.1,<1.26,>=1.21.1
    Downloading urllib3-1.25.11-py2.py3-none-any.whl (127 kB)
    | 127 kB 6.8 MB/s
  Collecting idna<3,>=2.5
    Downloading idna-2.10-py2.py3-none-any.whl (58 kB)
    | 58 kB 1.7 MB/s
  Requirement already satisfied: certifi>=2017.4.17 in c:\tools\anaconda3\envs\advancedpythontutorial\lib\site-packages (from requests->wasdi) (2020.6.20)
  Collecting chardet<4,>=3.0.2
    Downloading chardet-3.0.4-py2.py3-none-any.whl (133 kB)
    | 133 kB 6.4 MB/s
  Building wheels for collected packages: wasdi
    Building wheel for wasdi (setup.py) ... done
    Created wheel for wasdi: filename=wasdi-0.5.1-py3-none-any.whl size=27736 sha256=370ded1650747d7733645f9347bd760c88818debcb271159387c3ad2f4f3ded8
    Stored in directory: c:\users\c.nattero\appdata\local\pip\cache\wheels\37\09\c7\6ce30bb0f7b51acd9c8f2b4845cf76a4af148e27fce8d754ed
  Successfully built wasdi
  Installing collected packages: urllib3, idna, chardet, requests, wasdi
  Successfully installed chardet-3.0.4 idna-2.10 requests-2.24.0 urllib3-1.25.11 wasdi-0.5.1
  
```

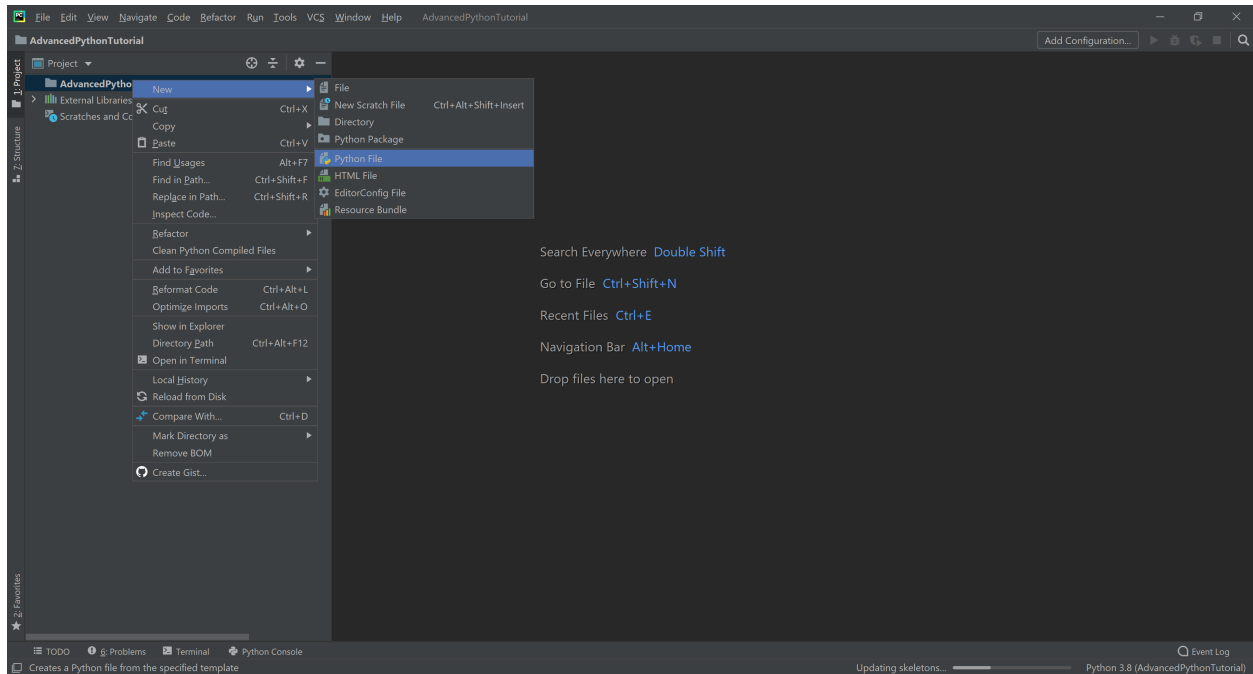
Note: Remember you need also to have gdal installed. If you previously installed wasdi, you may wish to update it by adding the `--upgrade` flag, i.e.:

```
pip install --upgrade wasdi
```

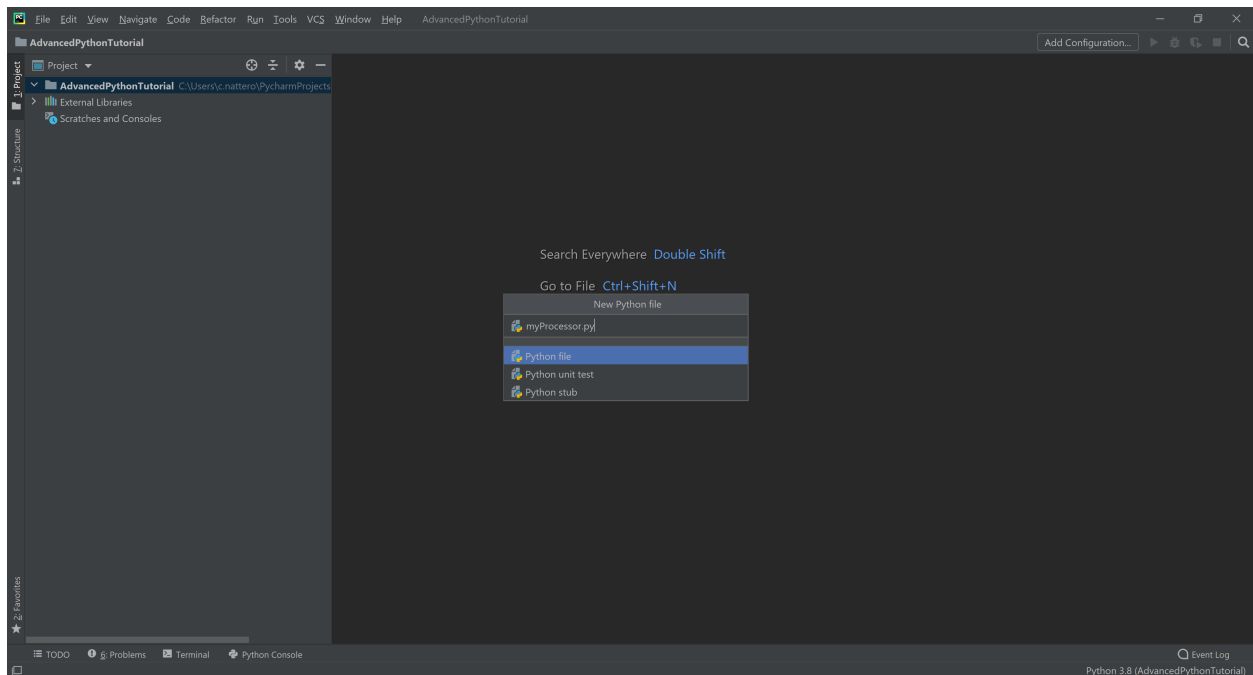
4.3.4 Create first files

Now we need to create these three fundamental files (right click on the Project Icon, new -> ...):

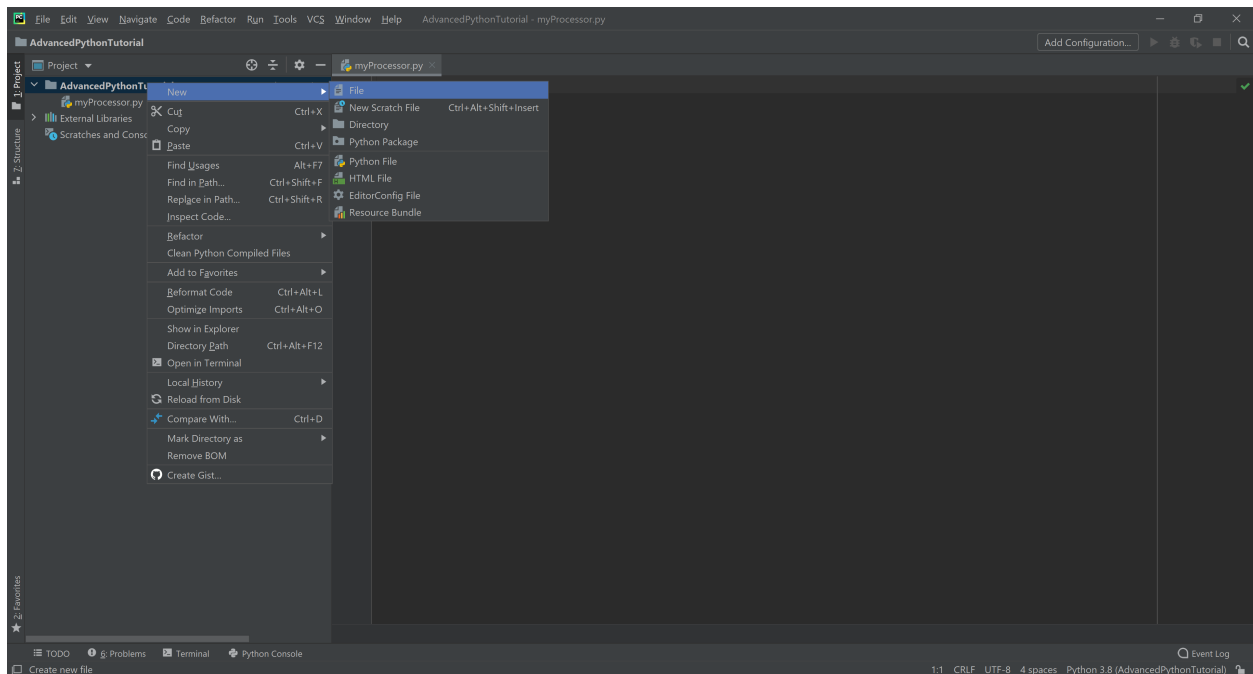
- `myProcessor.py`: create a python file, then call it `myProcessor.py`
- `config.json`: create a file, then call it `config.json` (PyCharm will recognize automatically it is a JSON file)
- `params.json`: create a file, then call it `params.json` (PyCharm will recognize automatically it is a JSON file)



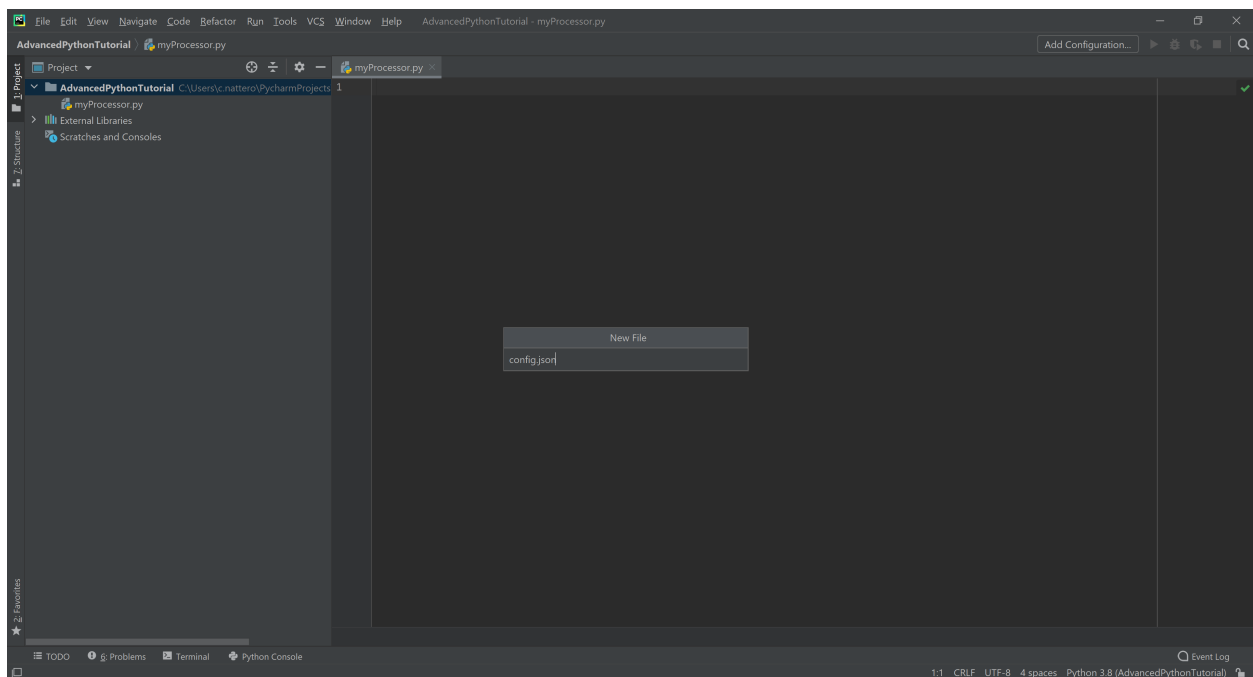
Create python file



Call it myProcessor.py

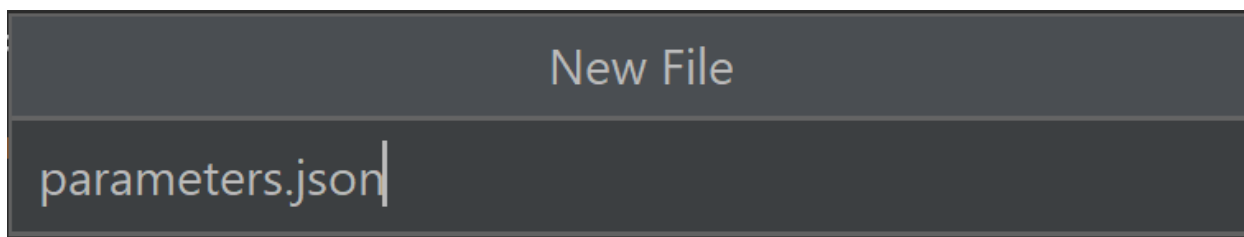


Create a json file



Call it config.json

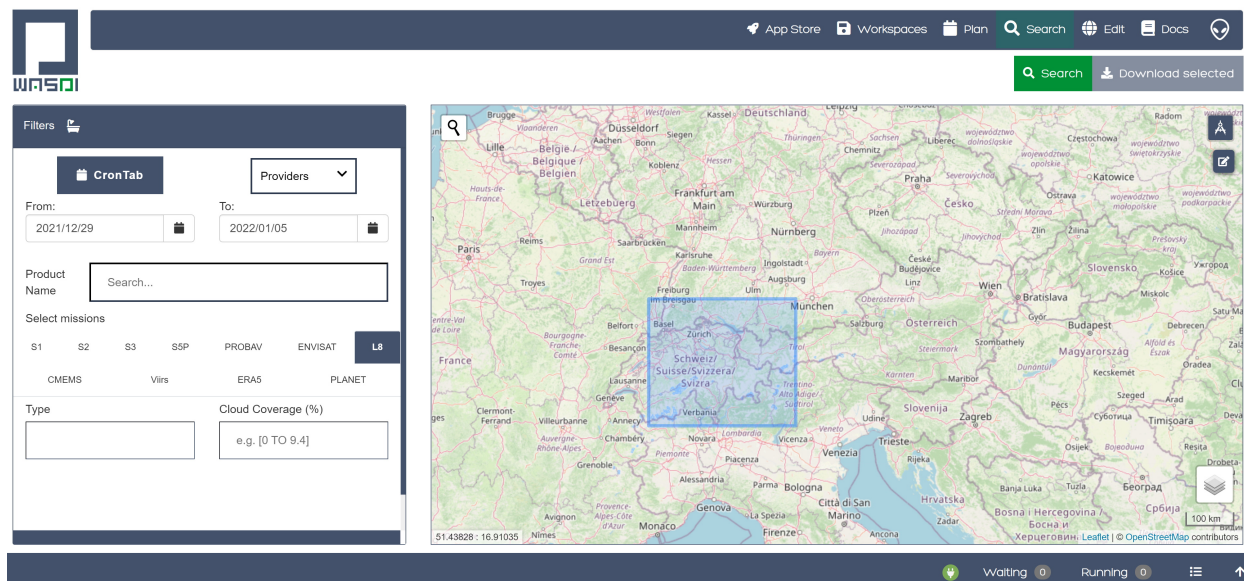
Create a json file



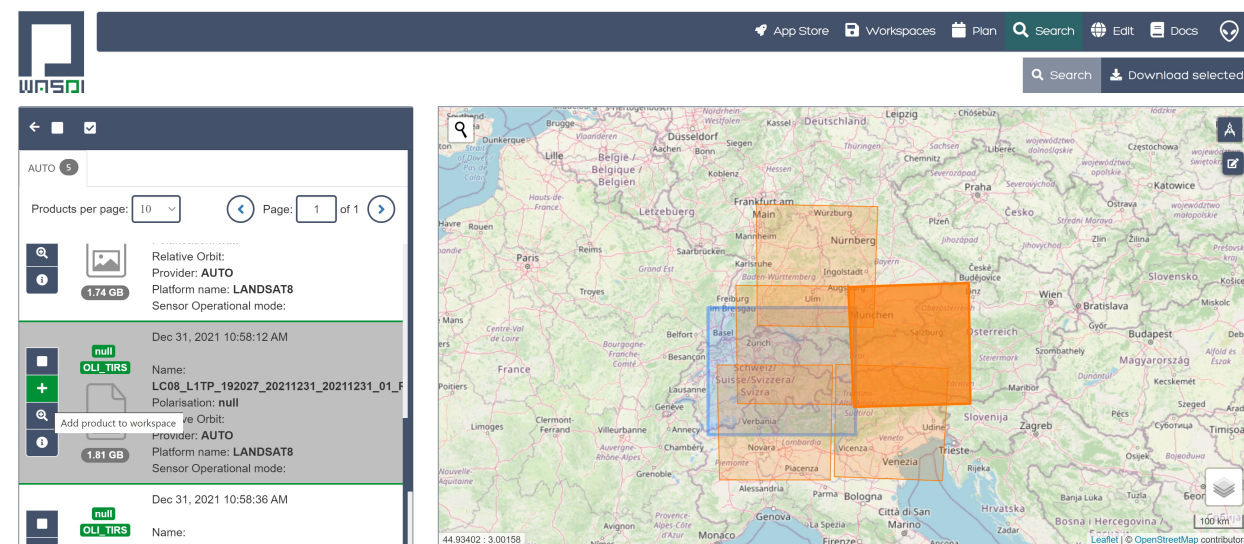
Call it params.json

Next, point your browser to wasdi.net, log in, go in the Workspaces Section and create a new workspace. Call it “Landsat8Tutorial”.

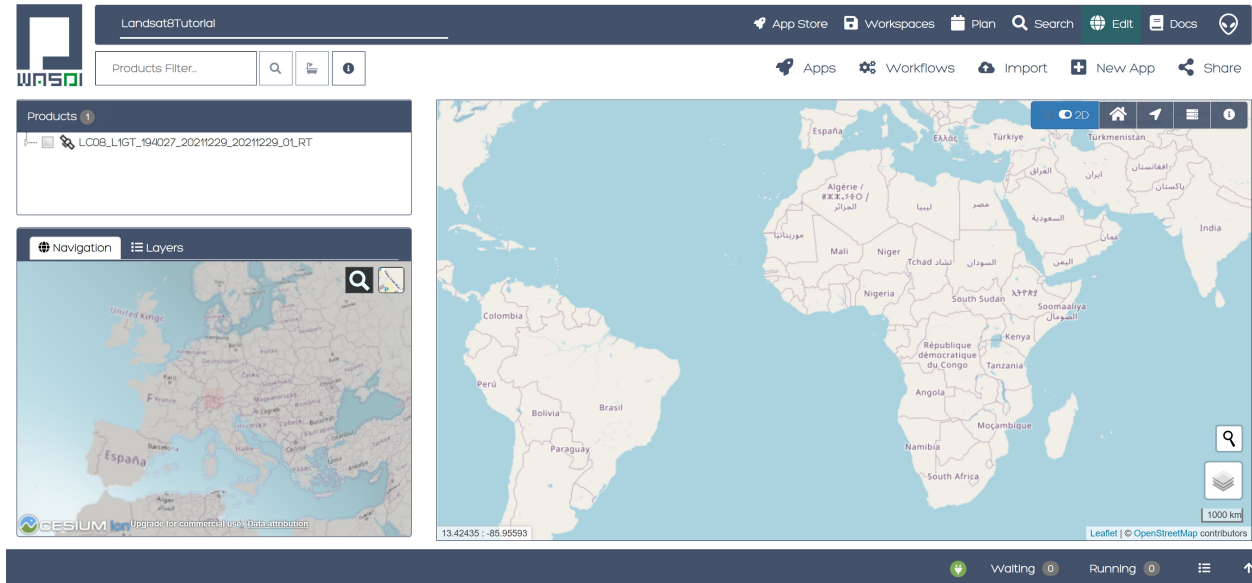
Go to the search section and select L8 data type and a bounding box in Europe



Select one image and click on the + button to add the image to the Landsat8Tutorial Wokspace



Come back to the edit section, and check that WASDI has been able to fetch the image.



Take note of the file you imported, we will need it later. For this tutorial we assume:

LC08_L1GT_196029_20211227_20211227_01_RT

but this can be changed with any image you imported.

Leave the browser open on that page, we will need it later on.

4.3.5 First lines

Let's begin by editing the **config.json** file. It is a JSON file, containing the user credentials and some fundamental parameters to get you started (see Wasdi Libraries Concepts):

```
{
  "USER": "your user name here",
  "PASSWORD": "your password here",
  "PARAMETERSFILEPATH": "./params.json"
  "WORKSPACE": "AdvancedTutorialTest"
}
```

Note: please, keep this file for yourself. You should never give this file to anyone else, and you do not need to upload to WASDI, as we'll see later on. You just need this file in your project for working with the WASDI python library. Use this file to change the workspace where you want to work.

Let's then edit **params.json** file. It is a JSON file that represents the inputs needed by our processor. The WASDI Developer can decide what parameters are needed; each parameter has a unique name within the processor. Each parameter can be of different types (i.e. Strings, Integers, Float, Arrays, Complex Objects...). **params.json** is where you declare and valorize your inputs. The same inputs will be available in the WASDI Web Interface when publishing the processor.

```
{
  "BANDS": ["B5", "B4"],
  "RESOLUTION": "30",
```

(continues on next page)

(continued from previous page)

```
"L8FILE": "LC08_L1GT_196029_20211227_20211227_01_RT.zip"
}
```

Now, open **myProcessor.py**, create a main and a method called run. The latter is required for WASDI to work (more on that later on).

Note:

These are two requirements necessary to use WASDI:

- have a python file called myProcessor.py
- have a function called run() (no params) within myProcessor.py

After that, you can include as many python files as you need, regardless their organization in directories. You just need to have a myProcessor.py with a method run() as entry point.

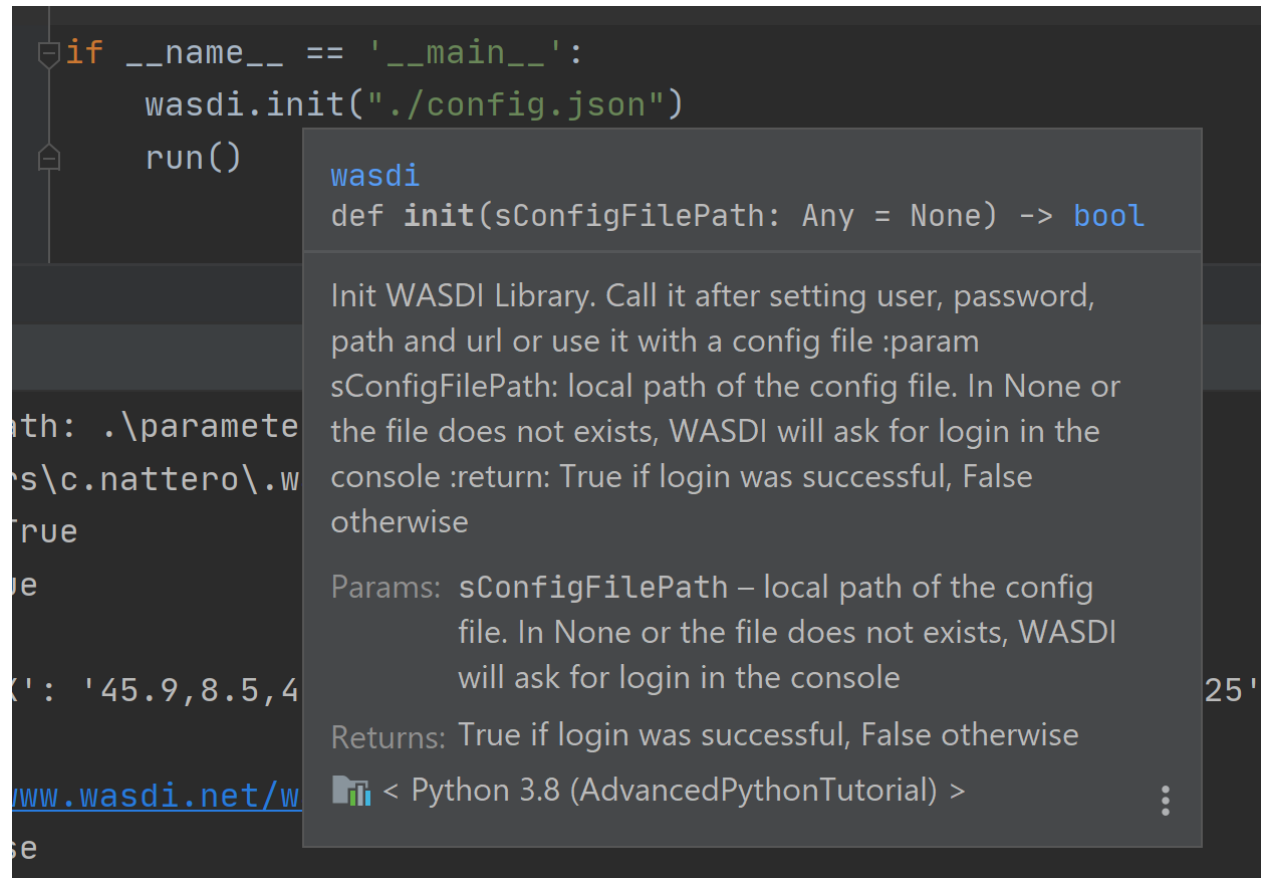
The main method will initiate the WASDI library and call the run method:

```
import wasdi

def run():
    pass

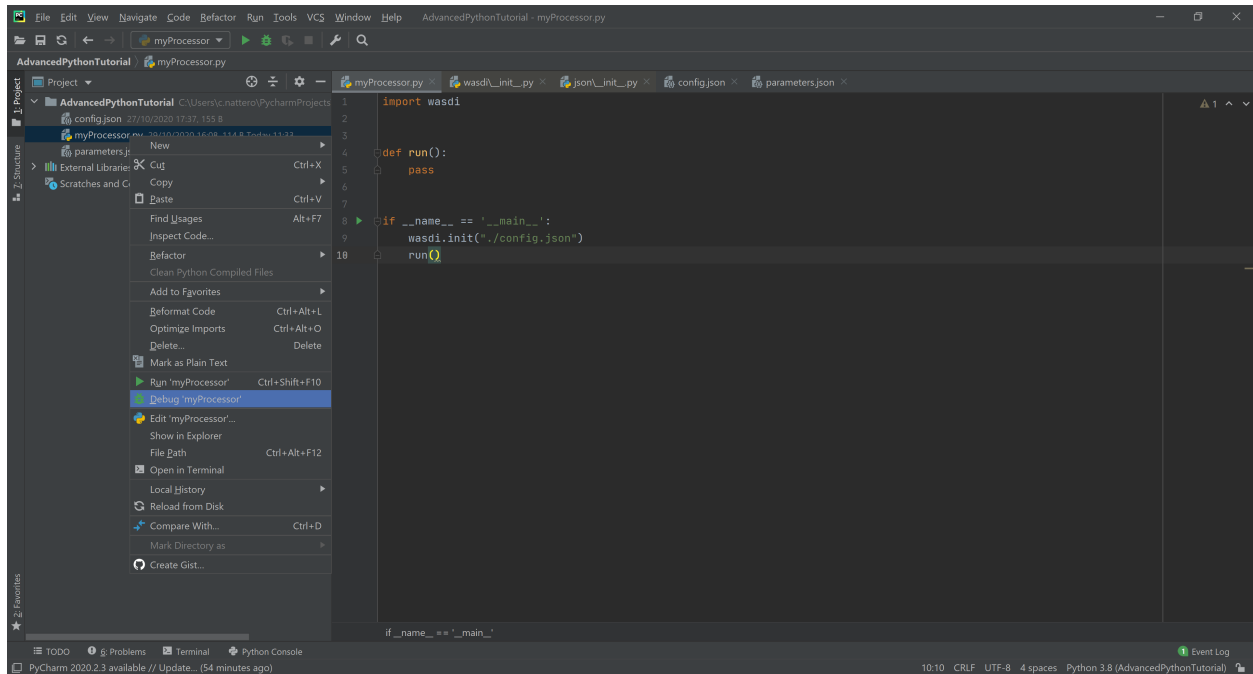
if __name__ == '__main__':
    wasdi.init("./config.json")
    run()
```

As you can see, we call wasdi.init and pass the relative path of the config file to it.



Let's debug to see the effects of this.

Note: If a file `main.py` was created automatically for you, remember to define another debug configuration. The easiest way to do so is by right clicking on your code and select Debug 'myProcessor.py'.



If the setup is correct so far, we should see the output from the wasdi library that shows the initialization has gone well. Something like this:

```
[INFO] _loadParams: wasdi could not load param file. That is fine, you can still load it.
↳ later, don't worry
[INFO] waspy.init: returned session is: 0d3f3ef1-f4c3-4202-9015-6ca17fc21cc7
[INFO] waspy.init: WASPY successfully initiated :-)
[INFO] waspy.printStatus: user: username@email.address
[INFO] waspy.printStatus: password: *****
[INFO] waspy.printStatus: session id: 0d3f3ef1-f4c3-4202-9015-6ca17fc21cc7
[INFO] waspy.printStatus: active workspace: 4f541d2c-4b29-445b-9869-9c8d185932ce
[INFO] waspy.printStatus: workspace owner: username@email.address
[INFO] waspy.printStatus: parameters file path: [...]params.json
[INFO] waspy.printStatus: base path: C:\Users\username\.wasdi\
[INFO] waspy.printStatus: download active: True
[INFO] waspy.printStatus: upload active: True
[INFO] waspy.printStatus: verbose: True
[INFO] waspy.printStatus: param dict: {'BANDS': ['B5', 'B4'], 'RESOLUTION': '30', 'L8FILE
↳ ': 'LC08_L1GT_196029_20211227_20211227_01_RT.zip'}
[INFO] waspy.printStatus: proc id:
[INFO] waspy.printStatus: base url: http://www.wasdi.net/wasdiwebserver/rest
[INFO] waspy.printStatus: is on server: False
[INFO] waspy.printStatus: workspace base url: http://www.wasdi.net/wasdiwebserver/rest
[INFO] waspy.printStatus: session is valid :-)
```

If you have the same situation, we are configured and ready to start!!

4.3.6 Extract Bands

The first step of our processor will be to extract the bands from the L8 image. WASDI ingest L8 images as a .zip file. Each .zip file contains different .tif images, one for each band, and some other files. We want to implement a function that takes as input the name of the L8 zip file, a list of bands, a resolution and that then creates a new .tif file with only the extracted bands at the desired resolution. The L8 bands are:

- B1 - Coastal aerosol 30m
- B2 - Blue 30m
- B3 - Green 30m
- B4 - Red 30m
- B5 - Near Infrared (NIR) 30m
- B6 - SWIR 1 30m
- B7 - SWIR 2 30m
- B8 - Panchromatic 15m
- B9 - Cirrus 30m
- B10 - Thermal Infrared (TIRS) 1 100m
- B11 - Thermal Infrared (TIRS) 2 100m

Our function is implemented like this:

```
def extractBands(sFile, asBands, sResolution="30"):
    """
    Extracts some bands from the L8 zip file into a multiband tiff file at the specified
    resolution
    Bands are
    B1 - Coastal aerosol 30m
    B2 - Blue 30m
    B3 - Green 30m
    B4 - Red 30m
    B5 - Near Infrared (NIR) 30m
    B6 - SWIR 1 30m
    B7 - SWIR 2 30m
    B8 - Panchromatic 15m
    B9 - Cirrus 30m
    B10 - Thermal Infrared (TIRS) 1 100m
    B11 - Thermal Infrared (TIRS) 2 100m

    :param sFile: name of the Landsat 8 file
    :param asBands: array of string with the names of the bands to extract
    :param sResolution: resolution as a string is in meteres
    :return Returns the name of the new tiff file
    """

    # Output File Name that will be returned
    sOutputTiffFile = ""

    try:
        # Prepare the name a .vrt file that will be used to extract bands from the zip
```

(continues on next page)

(continued from previous page)

```

sOutputVrtFile = sFile.replace(".zip", ".vrt")
# Prepare the name of the output tif file
sOutputTiffFile = sFile.replace(".zip", ".tif")

# Get the Local Path of the input Landsat file
sLocalFilePath = wasdi.getPath(sFile)

# Get the path of the output files
sOutputVrtPath = wasdi.getPath(sOutputVrtFile)
sOutputTiffPath = wasdi.getPath(sOutputTiffFile)

# Prepare an array of bands called BXX.TIF
asBandsTiff = [b + '.TIF' for b in asBands]

# Open the zip file
with zipfile.ZipFile(sLocalFilePath, 'r') as zf:
    # Get all the files in the zip
    asZipNameList = zf.namelist()
    # Take from the files in the zip, the ones that match the BXX.TIF naming
    ↪ schema we are searching
    asBandsL8 = [name for name in asZipNameList for band in asBandsTiff if band
    ↪ in name]

    # Create the zip path of the files we want to extract
    asBandsZip = ['/vsizip/' + sLocalFilePath + '/' + band for band in asBandsL8]

    # Create an array that has the names of the files to extract in the order
    ↪ required by the asBands array in input
    asOrderedZipBands = []

    for sBand in asBands:
        for sZipBand in asBandsZip:
            if sBand in sZipBand:
                asOrderedZipBands.append(sZipBand)
                break

    # Let gdal build a virtual file with our bands
    gdal.BuildVRT(sOutputVrtPath, asOrderedZipBands, separate=True)

    # Convert the vrt in tif with option -tr sResolution sResolution to have all
    ↪ bands at the same res (ie -tr 30 30 to have at 30 meters)
    ↪ + " " + sResolution)
    gdal.Translate(sOutputTiffPath, sOutputVrtPath, options="-tr " + sResolution

    # we can remove the vrt file
    os.remove(sOutputVrtPath)
except Exception as oEx:
    wasdi.wasdiLog("extractBands EXCEPTION")
    wasdi.wasdiLog(repr(oEx))
    wasdi.wasdiLog(traceback.format_exc())
except:
    wasdi.wasdiLog("extractBands generic EXCEPTION")

```

(continues on next page)

(continued from previous page)

```
# Return the output file name
return sOutputTiffFile
```

4.3.7 Compute NDVI

The second step is to compute the NDVI starting for our extracted Tif file. To compute NDVI we need to access the NIR and RED bands and compute the formula: $NDVI = NIR - RED / NIR + RED$

```
def computeNDVI(sTiffFile, sNDVIOutputFile):
    """
    Compute ndvi assuming that in sTiffPath there is as band 1 NIR and band 2 RED
    :param sTiffFile: name of the input tiff file
    :param sNDVIOutputFile: name of the ouput file with ndvi
    :return: full path of sNDVIOutputFile
    """

    # Open the tiff file: we assume it has two bands
    oDataset = gdal.Open(wasdi.getPath(sTiffFile))

    if not oDataset:
        wasdi.wasdiLog("Impossible to get Dataset from " + sTiffFile)
        return ""

    # Get the dimension of the bands in input
    [iCols, iRows] = oDataset.GetRasterBand(1).ReadAsArray().shape
    # Create gdal GeoTiff driver
    oDriver = gdal.GetDriverByName("GTiff")
    # Create a new Ouput file, same dimension of the input, compressed and with type_
    ↪ float32.
    oOutDataFile = oDriver.Create(wasdi.getPath(sNDVIOutputFile), iRows, iCols, 1, gdal.
    ↪ GDT_Float32, ['COMPRESS=LZW', 'BIGTIFF=YES'])

    # set to the output same geotransform as input
    oOutDataFile.SetGeoTransform(oDataset.GetGeoTransform())
    # set to the output same projection as input
    oOutDataFile.SetProjection(oDataset.GetProjection())

    # We assume NIR = band1, RED = band2
    oNIR = oDataset.GetRasterBand(1)
    oRED = oDataset.GetRasterBand(2)

    # Convert the band values in a numpy array
    adNIRBandArray = numpy.array(oNIR.ReadAsArray())
    adREDBandArray = numpy.array(oRED.ReadAsArray())
    # Force data to be float
    adNIRBandArray = adNIRBandArray.astype(float)
    adREDBandArray = adREDBandArray.astype(float)
    # Compute NDVI formula, where is not nan
    adNDVIBandArray = numpy.where((adNIRBandArray + adREDBandArray!=0), (adNIRBandArray-
    ↪ adREDBandArray)/(adNIRBandArray+adREDBandArray), 0)
```

(continues on next page)

(continued from previous page)

```

# Write the new calculated NDVI to output file band 1
oOutDataFile.GetRasterBand(1).WriteArray(adNDVIBandArray)
# We assume 0 as no data
oOutDataFile.GetRasterBand(1).SetNoDataValue(0)

# saves to disk!!
oOutDataFile.FlushCache()
wasdi.wasdiLog("Saved " + sNDVIOutputFile)

# Clean memory
oNIR = None
oRED = None

# Return the name of our NDVI create file
return sNDVIOutputFile

```

This tutorial shows an NDVI as a sample, but is clear that with this technique you can manipulate L8 data to fit your needs.

4.3.8 Main Function

Now the main operations are ready, we just need to put it all together.

```

def run():
    wasdi.wasdiLog("Landsat tutorial v.1.0")

    # Read from params the bands we want to extract and the resolution
    asBands = wasdi.getParameter("BANDS", ["B5", "B4"])
    sResolution = wasdi.getParameter("RESOLUTION", "30")
    sL8File = wasdi.getParameter("L8FILE", "LC08_L1GT_196029_20211227_20211227_01_RT.zip
↪")

    # Call extract bands
    sTiffFile = extractBands(sL8File, asBands, sResolution)

    # Prepare the output NDVI name
    sNDVIFile = sTiffFile.replace(".tif", "_NDVI.tif")

    # Call compute NDVI
    computeNDVI(sTiffFile, sNDVIFile)

    # Add the file to the WASDI workspace
    wasdi.addFileToWASDI(sNDVIFile, "NDVI")

```

You can now test your processor. Remember that, at the first time you will debug it locally, WASDI will take some time to download for you the L8 file you are using. All is done automatically and only once, when needed.

In the same way, when you add the file to WASDI, the lib will upload for your result to the cloud:

```

[INFO] waspy._internalAddFileToWASDI( LC08_L1GT_196029_20211227_20211227_01_RT_NDVI.tif, ↪
↪False )

```

(continues on next page)

(continued from previous page)

```
[INFO] waspy._internalAddFileToWASDI: remote file is missing, uploading
upload LC08_L1GT_196029_20211227_20211227_01_RT_NDVI.tif
uploadFile: uploading file to wasdi...
uploadFile: upload complete :-)
[INFO] waspy._internalAddFileToWASDI: file uploaded, keep on working!
[INFO] Running Locally, will not update status on server
```

Now that the core of our processor is done, lets make it a little bit more WASDI-integrated. We want to give some feedback to the user while the app is running and we do this using:

- wasdi.wasdiLog: locally just a print to console, when on the server, it sends the logs to the web user interface
- wasdi.updateProgressPerc: when on the server, updates the progress bar of the processor
- wasdi.setPayload: allows to save a user-defined object associated to the processor run

```
def run():
    wasdi.wasdiLog("Landsat tutorial v.1.0")

    # Read from params the bands we want to extract and the resolution
    asBands = wasdi.getParameter("BANDS", ["B5", "B4"])
    sResolution = wasdi.getParameter("RESOLUTION", "30")
    sL8File = wasdi.getParameter("L8FILE", "LC08_L1GT_196029_20211227_20211227_01_RT.zip
→")

    wasdi.wasdiLog("Calling extract bands")
    # Call extract bands
    sTiffFile = extractBands(sL8File, asBands, sResolution)

    wasdi.updateProgressPerc(30)
    wasdi.wasdiLog("Calculating NDVI")

    # Prepare the output NDVI name
    sNDVIFile = sTiffFile.replace(".tif", "_NDVI.tif")

    # Call compute NDVI
    computeNDVI(sTiffFile, sNDVIFile)
    wasdi.updateProgressPerc(80)

    wasdi.wasdiLog("Adding " + sNDVIFile + " to the workspace")
    # Add the file to the WASDI workspace
    wasdi.addFileToWASDI(sNDVIFile, "NDVI")

    # Create the payload object
    aoPayload = {}
    # Save the inputs that we received
    aoPayload["inputs"] = wasdi.getParametersDict()
    # Save the output we created
    aoPayload["output"] = sNDVIFile
    # Save the payload
    wasdi.setPayload(aoPayload)

    # Close the process setting the status to DONE
    wasdi.updateStatus("DONE", 100)
```

Welcome to Space, Have fun!

4.4 Search and Import EO Images

This tutorial has to goal to show the functionalities available in WASDI Libraries to search and import EO Images.

4.4.1 Introduction

WASDI is a multi-cloud multi-data-provider platform.

Multi-Cloud means that WASDI run on different cloud environments; your workspace can be hosted on CreoDIAS, ONDA, AdwaisEO, EDOC or other clouds. Generic users are routed by WASDI in the shared nodes, Premium users can have dedicated computing nodes and they can also decide to have a specific cloud, depending the real world needs. This allow us to make always the more optimized choice, and also to be able to have the system up and running also when a specific cloud has a planned or unplanned mantainance or worst problem.

Multi-Data-Provider means that WASDI is able to fetch (or import, or download, it depends) different kind of Data from different Providers. At the moment we are writing this tutorial WASDI is able to search and fecth these data:

- Sentinel-1
- Sentinel-2
- Sentinel-3
- Sentinel-5P
- VIIRS Nasa Flood Composite
- PROBA-V
- ENVI
- LANDSAT8
- ERA5 Copernicus Data
- CMES Copernicus Marine Data
- PLANET Commercial Images

WASDI is connected with these data providers:

- LSA Data Center
- ESA Sentinel Hub
- CREODIAS
- ONDA
- SOBLOO
- EODC
- CDS (Copernicus Data Science)
- NOAA
- Terrascope

The Platforms supported and the Data Providers connected are continuously growing.

In this tutorial we will see how to search and import images in WASDI.

4.4.2 Search EO Images

The function to search EO Images is searchEOImages.

It is a single function with different options.

```
def searchEOImages(sPlatform, sDateFrom, sDateTo,
                   FULLat=None, FULLon=None, fLRLat=None, fLRLon=None,
                   sProductType=None, iOrbitNumber=None,
                   sSensorOperationalMode=None, sCloudCoverage=None,
                   sProvider=None, oBoundingBox=None):
    """
    Search EO images

    :param sPlatform: satellite platform: (S1|S2|S3|S5P|VIIRS|L8|ENVI|ERA5)

    :param sDateFrom: initial date YYYY-MM-DD

    :param sDateTo: final date YYYY-MM-DD

    :param FULLat: Latitude of Upper-Left corner

    :param FULLon: Longitude of Upper-Left corner

    :param fLRLat: Latitude of Lower-Right corner

    :param fLRLon: Longitude of Lower-Right corner

    :param sProductType: type of EO product; Can be null. FOR "S1" -> "SLC","GRD",
    ↪ "OCN". FOR "S2" -> "S2MSI1C","S2MSI2Ap","S2MSI2A". FOR "VIIRS" -> "VIIRS_1d_composite",
    ↪ "VIIRS_5d_composite". FOR "L8" -> "L1T","L1G","L1GT","L1GS","L1TP". For "ENVI" -> "ASA_
    ↪ IM__0P", "ASA_WS__0P"

    :param iOrbitNumber: orbit number

    :param sSensorOperationalMode: sensor operational mode

    :param sCloudCoverage: interval of allowed cloud coverage, e.g. "[0 TO 22.5]"

    :param sProvider: WASDI Data Provider to query
    ↪ (AUTO|LSA|ONDA|CREODIAS|SOBLOO|VIIRS|SENTINEL). None means default node provider =
    ↪ AUTO.

    :param oBoundingBox: alternative to the float lat-lon corners: an object
    ↪ expected to have these attributes: oBoundingBox["northEast"]["lat"], oBoundingBox[
    ↪ "southWest"]["lng"], oBoundingBox["southWest"]["lat"], oBoundingBox["northEast"]["lng"]

    :return: a list of results represented as a Dictionary with many properties. The
    ↪ dictionary has the "fileName" and "relativeOrbit" properties among the others
    """
```

The only mandatory params are:

- sPlatform: a string with the code of the platform. Each search is done for a single platform. S1|S2|S3|S5P|VIIRS|L8|ENVI|ERA5 are the currently supported platforms

- sDateFrom: start date of the search. It is a string in the format YYYY-MM-DD (ie “2021-12-15”)
- sDateTo: end date of the search. It is a string in the format YYYY-MM-DD (ie “2021-12-15”)

The other highly recommended parameter is the bounding box. WASDI accepts only rectangle bounding boxes. This method supports two ways to specify the rectangle:

- fULLat, fULLon, fLRLat, fLRLon: four float numbers indicating Upper Left Latitude (North), Upper Left Longitude (West), Lower Right Latitude (South), Lower Right Longitude (East)
- oBoundingBox: an alternative that is an object that has these attributes: oBoundingBox[“northEast”][“lat”], oBoundingBox[“southWest”][“lng”], oBoundingBox[“southWest”][“lat”], oBoundingBox[“northEast”][“lng”]

sProductType is not mandatory. Can be specified as the “level of processing” of the Platform. Product Types supported are:

- **S1**
 1. SLC
 2. GRD
 3. OCN
- **S2**
 1. S2MSI1C
 2. S2MSI2Ap
 3. S2MSI2A
- **S3**
 1. SR_1_SRA___
 2. SR_1_SRA_A
 3. SR_1_SRA_BS
 4. SR_2_LAN___
- **S5P**
 1. L1B_IR_SIR
 2. L1B_IR_UVN
 3. L1B_RA_BD1
 4. L1B_RA_BD2
 5. L1B_RA_BD3
 6. L1B_RA_BD4
 7. L1B_RA_BD5
 8. L1B_RA_BD6
 9. L1B_RA_BD7
 10. L1B_RA_BD8
 11. L2__AER_AI
 12. L2__AER_LH
 13. L2__CH4___

14. L2__CLOUD__
15. L2__CO____
16. L2__HCHO__
17. L2__NO2____
18. L2__NP_BD3
19. L2__NP_BD6
20. L2__NP_BD7
21. L2__O3_TCL
22. L2__O3____
23. L2__SO2____
24. AUX_CTMFCT
25. AUX_CTMANA

- **VIIRS**

1. VIIRS_1d_composite
2. VIIRS_5d_composite

- **L8**

1. L1T
2. L1G
3. L1GT
4. L1GS
5. L1TP

- **ENVI**

1. ASA_IM__OP
2. ASA_WS__OP

- **PROBAV**

1. [urn:ogc:def:EOP:VITO:PROBAV_S1-TOC_333M_V001](#)
2. [urn:ogc:def:EOP:VITO:PROBAV_S10-TOC_333M_V001](#)

- **ERA5**

1. reanalysis
2. ensemble_mean
3. ensemble_members
4. ensemble_spread

- **PLANET**

1. PSScene
2. PSScene3Band
3. PSScene4Band

4. PSOrthoTile
5. REOrthoTile
6. REScene
7. SkySatScene
8. SkySatCollect
9. SkySatVideo

iOrbitNumber is supported by Sentinel-1 files and is used to filter the relative Orbit.

sSensorOperationalMode is also supported by Sentinel-1 files and can be:

- SM
- EW
- IW
- WV

sCloudCover is supported by Sentinel-2, Landsat8 and PROBA-V. It is the accepted cloud cover percentage. It is a string in this format: [MIN TO MAX]. For example “[0 TO 30]” means a maximum of 30% of cloud cover. “[0 TO 50]” means a maximum of 50% of cloud cover in the image.

sProvider is the provider to use to search and import images: the user can select one of the providers supported by wasdi: AUTO|LSA|ONDA|CREODIAS|SOBLOO|VIIRS|SENTINEL. By default, if it is left to null, WASDI will use the automatic data provider.

4.4.3 Automatic Data Provider

WASDI implements the Automatic Data Provider that is strongly suggested if you do not have a clear need to work with a specific Data Provider. As we have seen, WASDI supports many Platforms and many Data Provider.

In general, data of one Platform, can be obtained by one or more Data Providers.

The goal of WASDI is always: move the processor near to the data. We say, more realistically, minimize the data transfer.

Each Cloud has its own data policy: some has a full archive of some Platform, some have more platforms but with a Long Term Archive policy for file olders than some days, some allows only download, some the file access... it is a complex and varied scenario.

The Automatic Data Provider of WASDI knows the data you are searching and where your code is running: using this info, **WASDI makes a smart choice of the best Data Provider for you**. This functionality can be used both for search and import. Indeed, it is used by default if you do not specify a specific provider.

The Automatic Data Provider has also the advantage to try to get the image from another provider if, for any reason there is a problem to reach the best one. And this is done with all the providers that supports the platform you are searching, making WASDI very resilient to the external problems that may happen.

4.4.4 searchEOImages Output

The output of searchEOImages, is a list of Dictionary Objects with many properties.

The main important keys of the dictionary are “**fileName**” and “**link**”. This means you can access the file name with this code:

```
aoFound = wasdi.searchEOImages("S1", sDateFrom="2021-02-01", sDateTo="2021-02-02",
↪sProductType="GRD", fULLat=44.5, fULLon=8.5, fLRLat=44.0, fLRLon=9.0)
if len(aoFound) > 0:
    wasdi.wasdiLog("Image 0 name: " + aoFound[0]["fileName"])
```

Usually, you can use directly that object for what you need (ie to import the image/images), but as we have seen you can directly access the properties you need. In the returnet object, you can find also many other properties: these properties depends by the Platform and the selected Data Provider, and are easy to explore with a print or in a debug session.

4.4.5 Search Sample Code

In this code we just make different search, of different data types.

Note: This tutorial goes streight to the point: if you need help how to setup a project, add a parameter files and run your WASDI Application please refer to Python Tutorial.

First of all fill your params.json file:

```
{
  "bbox": {
    "northEast": {
      "lat": 30.0,
      "lng": -96.0
    },
    "southWest": {
      "lat": 29.5,
      "lng": -96.5
    }
  },
  "date": "2020-10-09",
  "searchdays": 10,
  "provider": "AUTO",
  "maxCloud": 30,
  "s1Type": "GRD"
}
```

We are defining a bounding box, a reference date, the number of days to search back and the Data Provider. We also add a Max Cloud for S2 data and the product type for S1.

The full code is here:

```
import wasdi
import sys
from datetime import datetime
from datetime import timedelta
```

(continues on next page)

(continued from previous page)

```

def run():

    try:
        # Read the bbox
        oBbox = wasdi.getParameter("bbox", None)
        # Read the reference Date
        sDate = wasdi.getParameter("date")
        # Read the provider
        sProvider = wasdi.getParameter("provider")
        # Read the number of days we want to search back from reference date
        iDays = wasdi.getParameter("searchdays", 10)
        # Cloud Cover
        iMaxCloud = wasdi.getParameter("maxCloud", 30)
        # S1 Product Type
        sS1ProductType = wasdi.getParameter("s1Type", "GRD")

        # A bounding box is really needed
        if oBbox is None:
            wasdi.wasdiLog("Bounding Box is null. The world is still too big.
→")

            wasdi.updateStatus("ERROR")
            sys.exit(1)

        # Initialize a safe date
        oEventDay = datetime.today()

        # Convert date from YYYY-MM-DD to a valid python date
        try:
            oEventDay = datetime.strptime(sDate, '%Y-%m-%d')
        except:
            wasdi.wasdiLog('Date not valid, assuming today')

        # Now we want to go back of iDays day
        oTimeDelta = timedelta(days=iDays)
        # Ok this is the start date
        oStartDay = oEventDay - oTimeDelta
        # And this is the end date
        oEndDay = oEventDay

        # Get back the date in string format
        sStartDate = oStartDay.strftime("%Y-%m-%d")
        sEndDate = oEndDay.strftime("%Y-%m-%d")

        # We start searching Sentinel 1 Data: here we use also product type
        aoFound = wasdi.searchEOImages("S1", sDateFrom=sStartDate,
→sDateTo=sEndDate, sProductType=sS1ProductType, sProvider=sProvider, oBoundingBox=oBbox)

        # Log how many images we found
        wasdi.wasdiLog("S1 found " + str(len(aoFound)))

        # This will be used to log but not too much
        iCount = 0

```

(continues on next page)

(continued from previous page)

```

# For each image
for oImage in aoFound:
    # Log the file name
    wasdi.wasdiLog(" " + oImage["fileName"] + " Orbit " + str(oImage[
↪"relativeOrbit"]))

    # Increment the counter
    iCount = iCount + 1
    if iCount>5:
        # Ok, understood the concept, now lets go on
        wasdi.wasdiLog("Break")
        break

# Search S2 Data
sCloudCoverage = "[0 TO " + str(iMaxCloud) + "]"
aoFound = wasdi.searchEOImages("S2", sDateFrom=sStartDate, ↪
↪sDateTo=sEndDate, sCloudCoverage=sCloudCoverage, sProvider=sProvider, ↪
↪oBoundingBox=oBbox)

# Log results, as before
wasdi.wasdiLog("S2 found " + str(len(aoFound)))

iCount = 0

for oImage in aoFound:
    wasdi.wasdiLog(" " + oImage["fileName"])
    iCount = iCount + 1
    if iCount>5:
        wasdi.wasdiLog("Break")
        break

# Search S3 Data
aoFound = wasdi.searchEOImages("S3", sDateFrom=sStartDate, ↪
↪sDateTo=sEndDate, sProvider=sProvider, oBoundingBox=oBbox)

# Log results, as before
wasdi.wasdiLog("S3 found " + str(len(aoFound)))

iCount = 0

for oImage in aoFound:
    wasdi.wasdiLog(" " + oImage["fileName"])
    iCount = iCount + 1
    if iCount>5:
        wasdi.wasdiLog("Break")
        break

# Search SSP Data
aoFound = wasdi.searchEOImages("SSP", sDateFrom=sStartDate, ↪
↪sDateTo=sEndDate, sProvider=sProvider, oBoundingBox=oBbox)

# Log results, as before

```

(continues on next page)

(continued from previous page)

```

wasdi.wasdiLog("S5P found " + str(len(aoFound)))

iCount = 0

for oImage in aoFound:
    wasdi.wasdiLog(" " + oImage["fileName"])
    iCount = iCount + 1
    if iCount>5:
        wasdi.wasdiLog("Break")
        break

# Search L8 Data
aoFound = wasdi.searchEOImages("L8", sDateFrom=sStartDate,
↪sDateTo=sEndDate, sProvider=sProvider, oBoundingBox=oBbox)

wasdi.wasdiLog("L8 found " + str(len(aoFound)))

iCount = 0

# For each image
for oImage in aoFound:
    wasdi.wasdiLog(" " + oImage["fileName"])
    iCount = iCount + 1
    if iCount>5:
        wasdi.wasdiLog("Break")
        break

#Search ENVI Data
aoFound = wasdi.searchEOImages("ENVI", sDateFrom=sStartDate,
↪sDateTo=sEndDate, sProvider=sProvider, oBoundingBox=oBbox)

wasdi.wasdiLog("ENVI found " + str(len(aoFound)))

iCount = 0

# For each image
for oImage in aoFound:
    wasdi.wasdiLog(" " + oImage["fileName"])
    iCount = iCount + 1
    if iCount>5:
        wasdi.wasdiLog("Break")
        break

# Search VIIRS Data
aoFound = wasdi.searchEOImages("VIIRS", sDateFrom=sStartDate,
↪sDateTo=sEndDate, sProvider=sProvider, oBoundingBox=oBbox)

wasdi.wasdiLog("VIIRS found " + str(len(aoFound)))

iCount = 0

# For each image

```

(continues on next page)

(continued from previous page)

```

        for oImage in aoFound:
            wasdi.wasdiLog(" " + oImage["fileName"])
            iCount = iCount + 1
            if iCount > 5:
                wasdi.wasdiLog("Break")
                break

    except Exception as oE:
        wasdi.wasdiLog("Error " + str(oE))
        wasdi.updateStatus('ERROR')
        sys.exit(1)

    wasdi.wasdiLog('Done bye bye')
    wasdi.updateStatus('DONE', 100)

if __name__ == '__main__':
    wasdi.init('./config.json')
    run()

```

The output of this code depends by the params you are using: for example, to find ENVI images, you have to go in the past. L8 images at the moment are found only in Europe. The cloud coverage can influence the S2 results. We suggest to play a little bit with the params to see different results.

4.4.6 Import functionalities

Once you found your images, usually you need to import one or more of the results in your workspace to continue your work. To do this, the lib gives you different options

- `importProductByFileUrl`: import a single product using directly file name and url. Almost a legacy method, but left for advanced use.
- `importProduct`: import a single product. Takes in input one of the objects returned by `searchEOImages`.
- `importProductList`: import a list of products. Takes in input an array of objects as returned by `searchEOImages`.

These are the synch version. Synch means that the function will not exit until the import is done. All methods returns the status of the operation, a string that can be:

- `DONE`: operation done with success
- `ERROR`: operation not done with an error
- `STOPPED`: operation stopped, by the user or by a timeout

A more advanced use of WASDI can bring you to use the async version of these methods. Async means that the method will return not the status but the `processId` of the import operation. This id can be used to query or wait the status of the import with `waitProcess`, `waitProcesses`, or `getProcessStatus`.

- `asyncImportProductByFileUrl`: import a single product using directly file name and url. Almost a legacy method, but left for advanced use.
- `asyncImportProduct`: import a single product. Takes in input one of the objects returned by `searchEOImages`.
- `asyncImportProductList`: import a list of products. Takes in input an array of objects as returned by `searchEOImages`.

The last option, is an optimized way to import a list of products and apply to them a specific SNAP Workflow. It can be a S1 search that, after the import, run a workflow to calibrate and geo-reference the image or a S2 that after the import run a workflow to run an Index like NDVI o many others.

- `importAndPreprocess`: Imports in WASDI and apply a SNAP Workflow to an array of EO Images as returned by `searchEOImages`. Takes in input the array of images, the name of the workflow to run, and the suffix to add to input files to create workflow output files.

4.4.7 Import Sample Code

The following python app make a search of S1 images and import the results in synch mode. It uses the same params used for the search sample.

```
import wasdi
import sys
from datetime import datetime
from datetime import timedelta

def run():

    try:
        # Read the bbox
        oBbox = wasdi.getParameter("bbox", None)
        # Read the reference Date
        sDate = wasdi.getParameter("date")
        # Read the provider
        sProvider = wasdi.getParameter("provider")
        # Read the number of days we want to search back from reference date
        iDays = wasdi.getParameter("searchdays", 10)
        # Cloud Cover
        iMaxCloud = wasdi.getParameter("maxCloud", 30)
        # S1 Product Type
        sS1ProductType = wasdi.getParameter("s1Type", "GRD")

        # A boundig box is really needed
        if oBbox is None:
            wasdi.wasdiLog("Boundig Box is null. The world is still too big.
↪")

            wasdi.updateStatus("ERROR")
            sys.exit(1)

        # Initialize a safe date
        oEventDay = datetime.today()

        # Convert date from YYYY-MM-DD to a valid python date
        try:
            oEventDay = datetime.strptime(sDate, '%Y-%m-%d')
        except:
            wasdi.wasdiLog('Date not valid, assuming today')

        # Now we want to go back of iDays day
        oTimeDelta = timedelta(days=iDays)
        # Ok this is the start date
```

(continues on next page)

(continued from previous page)

```

oStartDay = oEventDay - oTimeDelta
# And this is the end date
oEndDay = oEventDay

# Get back the date in string format
sStartDate = oStartDay.strftime("%Y-%m-%d")
sEndDate = oEndDay.strftime("%Y-%m-%d")

# We start searching Sentinel 1 Data: here we use also product type
aoFound = wasdi.searchEOImages("S1", sDateFrom=sStartDate,
↪sDateTo=sEndDate, sProductType=sS1ProductType, sProvider=sProvider, oBoundingBox=oBbox)

# Log how many images we found
wasdi.wasdiLog("S1 found " + str(len(aoFound)))

# Take the current product list
asCurrentFiles = wasdi.getProductsByActiveWorkspace()

if asCurrentFiles is not None:
    wasdi.wasdiLog("Products in the workspace before the import: " +
↪+ str(len(asCurrentFiles)))

# Import products, it may take time...
wasdi.importProductList(aoFound)

# Refresh the list
asCurrentFiles = wasdi.getProductsByActiveWorkspace()

if asCurrentFiles is not None:
    wasdi.wasdiLog("Products in the workspace after the import: " +
↪str(len(asCurrentFiles)))

except Exception as oE:
    wasdi.wasdiLog("Error " + str(oE))
    wasdi.updateStatus('ERROR')
    sys.exit(1)

wasdi.wasdiLog('Done bye bye')
wasdi.updateStatus('DONE', 100)

if __name__ == '__main__':
    wasdi.init('./config.json')
    run()

```

The same work can be done in an asynch way:

```

import wasdi
import sys
from datetime import datetime
from datetime import timedelta

def run():

```

(continues on next page)

(continued from previous page)

```

try:
    # Read the bbox
    oBbox = wasdi.getParameter("bbox", None)
    # Read the reference Date
    sDate = wasdi.getParameter("date")
    # Read the provider
    sProvider = wasdi.getParameter("provider")
    # Read the number of days we want to search back from reference date
    iDays = wasdi.getParameter("searchdays", 10)
    # Cloud Cover
    iMaxCloud = wasdi.getParameter("maxCloud", 30)
    # S1 Product Type
    sS1ProductType = wasdi.getParameter("s1Type", "GRD")

    # A bounding box is really needed
    if oBbox is None:
        wasdi.wasdiLog("Bounding Box is null. The world is still too big.
→")

        wasdi.updateStatus("ERROR")
        sys.exit(1)

    # Initialize a safe date
    oEventDay = datetime.today()

    # Convert date from YYYY-MM-DD to a valid python date
    try:
        oEventDay = datetime.strptime(sDate, '%Y-%m-%d')
    except:
        wasdi.wasdiLog('Date not valid, assuming today')

    # Now we want to go back of iDays day
    oTimeDelta = timedelta(days=iDays)
    # Ok this is the start date
    oStartDay = oEventDay - oTimeDelta
    # And this is the end date
    oEndDay = oEventDay

    # Get back the date in string format
    sStartDate = oStartDay.strftime("%Y-%m-%d")
    sEndDate = oEndDay.strftime("%Y-%m-%d")

    # We start searching Sentinel 1 Data: here we use also product type
    aoFound = wasdi.searchEOImages("S1", sDateFrom=sStartDate,
→sDateTo=sEndDate, sProductType=sS1ProductType, sProvider=sProvider, oBoundingBox=oBbox)

    # Log how many images we found
    wasdi.wasdiLog("S1 found " + str(len(aoFound)))

    # Take the current product list
    asCurrentFiles = wasdi.getProductsByActiveWorkspace()

```

(continues on next page)

(continued from previous page)

```

        if asCurrentFiles is not None:
            wasdi.wasdiLog("Products in the workspace before the import: " +
↪+ str(len(asCurrentFiles)))

        # Import products, in an async mode
        asProcIds = wasdi.asyncImportProductList(aoFound)

        # Do something else in the meanwhile: maybe smarter than this
        wasdi.wasdiLog("Here we are, while is working")
        iSampleNumber = 1
        iSampleNumber = iSampleNumber * 100
        wasdi.wasdiLog("We made something useless, in the meantime: " +
↪str(iSampleNumber))

        # Ok now wait for WASDI to finish
        wasdi.waitProcesses(asProcIds)

        wasdi.wasdiLog("Imports done")

        # Refresh the list
        asCurrentFiles = wasdi.getProductsByActiveWorkspace()

        if asCurrentFiles is not None:
            wasdi.wasdiLog("Products in the workspace after the import: " +
↪str(len(asCurrentFiles)))

    except Exception as oE:
        wasdi.wasdiLog("Error " + str(oE))
        wasdi.updateStatus('ERROR')
        sys.exit(1)

    wasdi.wasdiLog('Done bye bye')
    wasdi.updateStatus('DONE', 100)

if __name__ == '__main__':
    wasdi.init('./config.json')
    run()

```

Note that, while you are importing images, if you open the workspace on WASDI, you will see your operations on going.

Welcome to space.

4.4.8 Search Parameter Documentation

Sentinel-1 Parameters

.filename: This is the Satellite Platform. It can be of type “S1A” or “S1B”.

.producttype: This is the Product Type. It can be of type “SLC”, “GRD” or “OCN”.

.polarisationmode: This is the Polarisation. It can be of type “HH”, “VV”, “HV”, “VH”, “HH+HV”, “VV+HH”.

.sensoroperationalmode: This is the Sensor Mode. It can be of type “SM”, “IW”, “EW”, or “WV”.

.relativeorbitnumber: This is the Relative Orbit Number. It can be an integer from 1 to 175.

****swathidentifier:** ** This is the Swath.

Sentinel-2 Parameters

.filename: this is the Satellite Platform. It can be of type “S2A” or “S2B”.

.producttype: this is the product type. It can be of type “S2MSI1C”, “S2MSI2Ap” or “S2MSI2A”.

.cloudcoverpercentage: this is the Cloud Coverage Percentage. It can be an integer range from 0 to 100 - including float numbers (e.g., 0 to 9.4).

Sentinel-3 Parameters

.productlevel: this is the product level. It can be of type “L1” or “L2”.

.insturment: This is the Insturment. It can only be of type “SRAL”.

.producttype: This is the Product Type. Its value is determined by the following conditions:

- If .productlevel is set to “L1” it can be of type “SR_1_SRA___”, “**SR_1_SRA_A_**” or “SR_1_SRA_BS”.
- If .productlevel is set to “L2” it can be of type “SR_2_LAN___”

.timeliness: This is the Timeliness. It can be of value “Near Real Time”, “Short Time Critical”, or “Non Time Critical”.

.realtiveorbitstart: this is the Relative Orbit Start. The value can be an integer from 1 to 385.

Sentinel-5P Parameters

.productlevel: This is the product level. It can be of value “LEVEL1B” or “LEVEL2”.

.producttype: This is the Product Type. Its value is determined by the following conditions:

- If .productlevel is set to “LEVEL1B”, the value can be “L1B_IR_SIR”, “L1B_IR_UVN”, “L1B_RA_BD1”, “L1B_RA_BD2”, “L1B_RA_BD3”, “L1B_RA_BD4”, “L1B_RA_BD5”, “L1B_RA_BD6”, “L1B_RA_BD7”, “L1B_RA_BD8”, or “AUX_CTMFCT”, “AUX_CTMANA”
- If .productlevel is set to “LEVEL2”, then the value of .producttype can be: “L2__AER_AI”, “L2__AER_LH”, “L2__CH4___”, “L2__CLOUD_”, “L2__CO___”, “L2__HCHO_”, “L2__NO2___”, “L2__NP_BD3”, “L2__NP_BD6”, “L2__NP_BD7”, “L2__O3_TCL”, “L2__O3___”, “L2__SO2___”, “AUX_CTMFCT”, or “AUX_CTMANA”

.timeliness: This is the Timeliness. It can be of value “Offline”, “Near real time”, or “Reprocessing”.

.absoluteorbit: This is the Absolute Orbit Number.

PROBA-V Parameters

.collection: This is the Collection. The value can be “urn:ogc:def:EOP:VITO:PROBAV_S1-TOC_333M_V001”, “urn:ogc:def:EOP:VITO:PROBAV_S10-TOC_333M_V001”, or “urn:ogc:def:EOP:VITO:PROBAV_S5-TOC_100M_V001”

.cloudcoverpercentage: This is the Cloud Coverage Percentage. It can be set to a range of 0 to 100 (e.g., 0 to 9.4).

.snowcoverpercentage: This is the snowcoverpercentage. It can be set to a range of 0 to 100 (e.g., 0 to 9.4).

.productref: This is the product ref.

.cameraId: This is the Camera Id.

.productID: This is the Product ID.

.year: This is the year.

.Insturment: This is the Insturment. It can be of value “VG1” or “VG2”.

Envisat Parameters

.name: This is the Type. It can be of value “ASA_IM__0P”, or “ASA_WS__0P”.

.orbitDirection: This is the Orbit Direction. It can be of value “ASCENDING” or “DESCENDING”

Landsat8 Parameters

.name: This is the Type. It can be of value “L1T”, “L1G”, “L1GT”, “L1GS”, “L1TP”.

.cloudcoverpercentage: This is the Cloud Cover Percentage. It can be set to a range of 0 to 100 (e.g., 0 to 94).

VIIRS Parameters

.producttype: This is the Product. It can be of value “VIIRS_1d_composite” or “VIIRS_5d_composite”.

ERA5 Parameters

.dataset: This is the Dataset. It can be of value “reanalysis-era5-pressure-levels” or “reanalysis-era5-single-levels”.

.productType: This is the Product Type.

- If .dataset is set to “reanalysis-era5-pressure-levels” then the .producttype can be of value “reanalysis”, “ensemble_mean”, “ensemble_members”, or “ensemble_spread”.
- If .dataset is set to “reanalysis-era5-single-levels” then the .producttype can be of value “reanalysis”, “ensemble_mean”, “ensemble_members”, or “ensemble_spread”.

.pressureLevels: This is the Pressure Levels in hPa. If .dataset is set to “reanalysis-era5-pressure-levels” then the value of .pressureLevels can be “1”, “2”, “1+2”, or “1000”.

.variables: This is the Variables.

- If .dataset is set to “reanalysis-era5-pressure-levels” then the value of .variables can be “RH”, “U”, “V”, or “RH+U+V”.
- If .dataset is set to “reanalysis-era5-single-levels” then the value of .variables can be “SST+SP+ST” or “10U+10V+2DT+2T+SP”.

.format: This is the Format. It can be of value “grib” or “netcdf”.

CAMS Parameters

.dataset: This is the Dataset and can be of value “cams-global-atmospheric-composition-forecasts”.

.type: This is the Type. If the .dataset is set to “cams-global-atmospheric-composition-forecasts” then the value can be “forecast”.

.variables: This is the Variables. If the .dataset is set to “cams-global-atmospheric-composition-forecasts” then the value will be set to “dust_aerosol_optical_depth_550nm”.

.format: This is the Format. It can be of value “grib” or “netcdf_zip”.

PLANET Parameters

.producttype: This is the Planet Item Type. It can be of value “PSScene”, “PSScene3Band”, “PSScene4Band”, “PSOrthoTile”, “REOrthoTile”, “REScene”, “SkySatScene”, “SkySatCollect”, or “SkySatVideo”.

DEM Parameters

.dataset: This is the Dataset. It can be of value “DEM_30M” or “DEM_90M”.

WorldCover Parameters

.dataset: This is the Dataset. It can be of value “10m_2020_V1”.

StaticFiles Parameters

.producttype: This is the Product Type. It can be of either value “ESA_CCI_LAND_COVER_2015” or “ESACCI-Ocean-Land-Map-150m-P13Y-2000”.

IMERG Parameters

.latency: This is the Latency. It can be of value “Early” or “Late”.

.duration: This is the Duration. Its value is based on the following conditions:

- If .latency is set to “Early” .duration can have a of value “HHR”.
- If .latency is set to “Late” .duration can have a value of “HHR”, “DAY”, or “MO”

.accumulation: This is the Accumulation. Its value is based on the following conditions:

- If .latency is set to “Early” and .duration is set to “HHR” then .accumulation can have a value of “30min”, “1day”, “3hr”, or “All”.
- If .latency is set to “Late” and .duration is set to “HHR” then .accumulation can have a value of “30min”, “1day”, “3day”, “7day”, “3hr” or “All”.

CM Parameters

.producttype: This is the Product Type. It can have a value of

- “OCEANCOLOUR_MED_CHL_L4_NRT_OBSERVATIONS_009_041-TDS”,
- “OCEANCOLOUR_GLO_BGC_L4_MY_009_108-TDS”,
- “SST_MED_SST_L3S_NRT_OBSERVATIONS_010_012-TDS”,
- “SST_MED_SST_L4_NRT_OBSERVATIONS_010_004-TDS”,
- “GLOBAL_ANALYSIS_FORECAST_WAV_001_027-TDS”,
- “INSITU_MED_NRT_OBSERVATIONS_013_035”, or
- “OCEANCOLOUR_GLO_CHL_L3_NRT_OBSERVATIONS_009_032-TDS”

.protocol: This is the Protocol. If .producttype has a value matching any from the list below, then .protocol can have a value of either “SUBS” or “FTP”.

- OCEANCOLOUR_MED_CHL_L4_NRT_OBSERVATIONS_009_041-TDS
- OCEANCOLOUR_GLO_BGC_L4_MY_009_108-TDS
- SST_MED_SST_L3S_NRT_OBSERVATIONS_010_012-TDS
- SST_MED_SST_L4_NRT_OBSERVATIONS_010_004-TDS
- GLOBAL_ANALYSIS_FORECAST_WAV_001_027-TDS
- OCEANCOLOUR_GLO_CHL_L3_NRT_OBSERVATIONS_009_032-TDS

If .product type is set to the value “INSITU_MED_NRT_OBSERVATIONS_013_035” then .protocol can have a value of “FTP”.

.dataset: This is the Dataset. Its possible value is determined by these following conditions:

- If the .producttype has a value of “OCEANCOLOUR_MED_CHL_L4_NRT_OBSERVATIONS_009_041-TDS” then .dataset can have a value of one of the following:
 - “dataset-oc-med-chl-multi-l4-chl_1km_monthly-rt-v02”
 - “dataset-oc-med-chl-multi-l4-interp_1km_daily-rt-v02”
 - “dataset-oc-med-chl-olci_a-l4-chl_1km_monthly-rt-v02” or
 - “dataset-oc-med-chl-olci-l4-chl_300m_monthly-rt”
- If the .producttype has a value of “SST_MED_SST_L3S_NRT_OBSERVATIONS_010_012-TDS” then .dataset can have a value of either:
 - “SST_MED_SST_L3S_NRT_OBSERVATIONS_010_012_a” or
 - “SST_MED_SST_L3S_NRT_OBSERVATIONS_010_012_b”
- If .producttype is set to “SST_MED_SST_L4_NRT_OBSERVATIONS_010_004-TDS” then .dataset can have a value of:
 - “SST_MED_SST_L4_NRT_OBSERVATIONS_010_004_a_V2”
 - “SST_MED_SST_L4_NRT_OBSERVATIONS_010_004_c_V2”
 - “SST_MED_SSTA_L4_NRT_OBSERVATIONS_010_004_b” or
 - “SST_MED_SSTA_L4_NRT_OBSERVATIONS_010_004_d”
- If the .producttype has a value of “GLOBAL_ANALYSIS_FORECAST_WAV_001_027-TDS” and .protocol is set to “SUBS” then .dataset can have a value of “global-analysis-forecast-wav-001-027”.

- If the .producttype is set to “GLOBAL_ANALYSIS_FORECAST_WAV_001_027-TDS” and .protocol is set to “FTP” then .dataset can have a value of:
 - “global-analysis-forecast-wav-001-027” or
 - “global-analysis-forecast-wav-001-027-statics”
- If the .producttype is “INSITU_MED_NRT_OBSERVATIONS_013_035” and the .protocol is “FTP” then the value of .dataset can be “med_multiparameter_nrt”.
- If .producttype is set to “OCEANCOLOUR_GLO_CHL_L3_NRT_OBSERVATIONS_009_032-TD” then the value of .dataset can be “dataset-oc-glo-bio-multi-l3-chl_300m_daily-rt”.

.variables: This is the Variables. The possible values of .variables is determined by the following conditions:

- If .protocol is set to “SUBS” AND .dataset is set to one of the following, then the value of variables can be “CHL+CHL_count+CHL_error”, “CHL”, “CHL_count” or “CHL_error”.
 - “dataset-oc-med-chl-multi-l4-interp_1km_daily-rt-v02” and .protocol is set to “SUBS” then the value of .variables can be “CHL”.
- If .dataset is set to “c3s_obs-oc_glo_bgc-plankton_my_l4-multi-4km_P1M” and .protocol is set to “SUBS” then the value of .variables can be “CHL+CHL_count+CHL_error+grid_mapping”, “CHL”, “CHL_count”, “CHL_error”, or “grid_mapping”.
- If .protocol is set to “SUBS” and .dataset is set either “SST_MED_SST_L3S_NRT_OBSERVATIONS_010_012_a” or “SST_MED_SST_L3S_NRT_OBSERVATIONS_010_012_b” then variables can have a value of one of the following:
 - “adjusted_sea_surface_temperature+quality_level+sea_surface_temperature+source_of_sst”,
 - “adjusted_sea_surface_temperature|quality_level”
 - “sea_surface_temperature”, or
 - “source_of_salt”
- If .protocol is set to “SUBS” and .dataset is set to either “SST_MED_SST_L4_NRT_OBSERVATIONS_010_004_a_V2” or “SST_MED_SST_L4_NRT_OBSERVATIONS_010_004_c_V2” then .variables can have a value of one of the following:
 - “analysed_sst+analysis_error”
 - “analysed_sst”, or
 - “analysis_error”:
- If .protocol is set to “SUBS” and .dataset is set to either “SST_MED_SSTA_L4_NRT_OBSERVATIONS_010_004_b” or “SST_MED_SSTA_L4_NRT_OBSERVATIONS_010_004_d” then .variables can have a value of “sst_anomaly”.
- If .dataset is set to “global-analysis-forecast-wav-001-027” and .protocol is set to “SUBS” then .variables can have a value of one of the following:
 - **VHM0+VHM0_SW1+VHM0_SW2+VHM0_WW+VMDR+VMDR_SW1+VMDR_SW2+VMDR_WW+VPED+VSDX+VSDY+VTM01_SW1+VTM01_SW2+VTM01_WW+VTM02+VTM10+VTPK**
 - VHM0
 - VHM0_SW1
 - VHM0_SW2
 - VHM0_WW
 - VHDR

- VMDR_SW1
 - VMDR_SW2
 - VMDR_WW
 - VPED
 - VSDX
 - VSDY
 - VTM01_SW1
 - VTM01_SW2
 - VTM01_WW
 - VTM02
 - VTM10
 - VTPK
- If `.dataset` has a value of “dataset-oc-glo-bio-multi-l3-chl_300m_daily-rt” and `.protocol` is set to “SUBS” then `.variables` can have a value of one of the following:
 - HL+CHL_error+CHL_flags
 - CHL
 - CHL_error
 - CHL_flags

ECOSTRESS Parameters

.dataset: This is the Dataset. It can have a value of one of the following:

- EEHCM
- EEHSEBS
- EEHSTIC
- EEHSW
- EEHTES
- EEHTSEB

.relativeorbitnumber: This is the Orbit Range. It can have a value of an integer from 1 to 19999.

.parameteName: This is the Parameter. It can have a value of either “Day” or “Night”.

.dayNightFlag: This is the DayNightFlag. It can have a value of either “L1B_RAD” or “Night”.

4.5 Configuration tutorial

4.5.1 Introduction

The present Tutorial showcase all the feature available in the *WASDI configuration file*. The config file is required to connect the running instance of your code with WASDI services.

Note: This page reports all the features with complete description, if you just need a quick reference or a starting point please refer to *config quickstart*

This tutorial showcase all the available features with working example, explaining all the available fields and their usage. As a reference library, waspy python library is used but the same concepts applies for all wasdi libraries.

Note: The configuration file contains your credentials and some additional information to get WASDI started: never share it with others! It is required only for developing on your PC, so do not upload it to WASDI when deploying or updating an application

4.5.2 Login parameters

WASDI allows a simple transition from local execution on your machine to the execution in the cloud, leveraging powerful dedicated cloud resources and faster interactions with data.

In order to start a new session with WASDI web services users can login to WASDI web services by invoking the **login** method. If no parameters are passed the library will ask for credentials on the console where the code is executed.

Another possibility is to rely on the *config.json* file. This file can be read by using the **init** method. Upon initialization, the library search for credentials inside the configuration passed and proceed to create a new session for the user.

Note: Please remember that the config file follows the *JSON* syntax specification. Please check the syntax to adhere such specification.

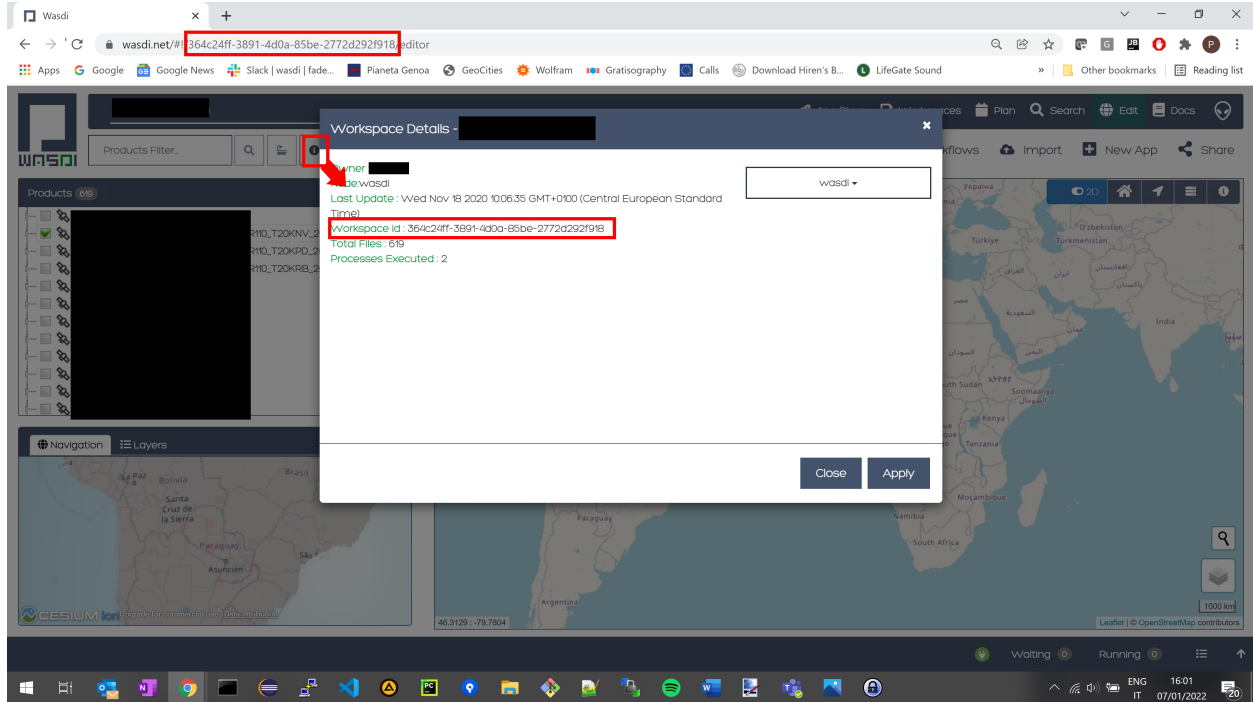
4.5.3 Workspace initialization

After login, config.json file can be used to open a specific workspace and start on working there. The reference can be done by using the workspace name OR the workspace Id.

Here's an example of the field included in a config file:

```
{
  "WORKSPACE" : "workspace Name",
  "WORKSPACEID" : "workspaceId"
}
```

The workspace id can be found in WASDI application both on navigation bar or inside workspace details window.



Note: In case both parameters, name and id, are specified only **workspace name** is taken in consideration.

4.5.4 Parameters dictionary

In order to make your WASDI application completely configurable, it is possible to embed a **custom dictionary** for the execution. This can be very flexible and useful, by instance, to parametrize some part of your algorithms and checks the results on different setup.

The dictionary use itself the JSON syntax and can contains any number of custom field. Such parameters can be stored on a separated JSON file and in order to be loaded, can be specified the path to the particular file.

To load a parameter dictionary the config file can specify the **PARAMETERSFILEPATH** field

```
{
  "PARAMETERSFILEPATH" : "./params.json"
}
```

In this example we are using file named **params.json** in the same folder of the python script with the following content:

```
{
  "First_Param" : "First_Value",
  "Second_Param" : 42
}
```

In the following of your code such values can be used by invoking, for example the **getParameter()** method:

```
print(wasdi.getParameter("First_Param"))  
  
print(wasdi.getParameter("Second_Param"))
```

The execution of the above code will simply print on the command line the values specified from the params.json file.

4.5.5 Download/Upload activation

In local environment it is possible to enable or disable both the upload and download of resources. To control this aspect please check the following field of the config file:

```
{  
  "DOWNLOADACTIVE" : true  
  "UPLOADACTIVE" : false  
}
```

Both parameters are boolean so please check the syntax for JSON format.

4.5.6 Logs verbosity

Upon execution of the code it is possible to control the verbosity.

Ideally during debugging of your scripts it is useful to have more details about the status of the running instance. Instead, when the code is effectively deployed, the logs can be disabled by setting this field to false.

```
{  
  "VERBOSE" : false  
}
```

Ideally during debugging of your scripts it is useful

to have more details about the status of the running instance. Instead, when the code is effectively deployed, the logs can be disabled by setting this field to false.

4.5.7 Base path

This parameter control where the running instance of your application stores the workspaces and their relative assets downloaded from WASDI.

If no path is specified this defaults to

- Linux: /home/[your user]/.wasdi
- Windows: C:\Users\[your user]\.wasdi

```
{  
  "BASEPATH" : "[Custom Directory]"  
}
```

4.5.8 Advanced settings

The following parameters represents functionalities for an advanced usage of WASDI. Each parameters is hereby described.

```
{
  "BASEURL" : "baseUrl",
  "REQUESTSTIMEOUT" : "requestsTimeout",
  "SESSIONID" : "sessionId",
  "MYPROCID" : "myProcId",
  "ENABLECHECKSUM" : "enableChecksum"
}
```

BASEURL Controls the base address used to contact the WASDI services. In brief WASDI is multicloud and can be executed in several different cloud environments: this parameter allows the developer contact various instances of WASDI deployed on several different URLs.

REQUESTSTIMEOUT Allows the developer to setup the time, *in seconds*, before the call to WASDI api must respond before the connection is dropped.

SESSIONID This field allows to connect directly to WASDI services skipping the login process and supplying directly a session id

MYPROCID Allows to force the Process ID within the workspace. All processes have a dedicated ID. In general this id is a GUID generated at each launch. This field of the settings gives the possibility to the developer to force a custom value.

ENABLECHECKSUM When dealing with EO product and in particular with their transfer between environments, it can be useful to relies the checksum to check data integrity. When enabling this field the MD5 check sum is requested to the WASDI instance and verified against the downloaded files.

4.5.9 Complete config.json reference

In the following a complete config.json file example is reported, showcasing all the possible parameters readable by the libraries.

```
{
  "USER" : "user",
  "PASSWORD" : "password",
  "WORKSPACE" : "workspace",
  "WORKSPACEID" : "workspaceId",
  "BASEPATH" : "basePath",
  "PARAMETERSFILEPATH" : "parametersFilePath",
  "DOWNLOADACTIVE" : true,
```

(continues on next page)

(continued from previous page)

```
"UPLOADACTIVE" : true,

"VERBOSE" : true,

"BASEURL" : "baseUrl",

"REQUESTSTIMEOUT" : "requestsTimeout",

"SESSIONID" : "sessionId",

"MYPROCID" : "myProcId",

"ENABLECHECKSUM" : true
}
```

4.6 Working with Workspaces and Products

This tutorial has to goal to intruduce the main functionality of the WASDI Libraries to work with workspaces (see Wasdi Libraries Concepts for the main concepts).

4.6.1 Introduction

Workspaces are the space where you can import files, run applications, run workflows, run your code, create and add your own files.

Each workspace has a Name, an Id (guid) and a owner.

Each workspace is hosted on a WASDI Computing node: this can be a shared node for generic users or a dedicated Computing node for premium users.

Each workspace can be shared with other WASDI Users.

Developers can access their own workspaces or the workspaces other users shared with them.

4.6.2 Workspace functionalities

Using the libraries, a developer can:

- Get the id of the active workspace
- Get the name of a workspace from the Id or viceversa
- Open another workspace
- Get the list of user workspaces

4.6.3 Workspaces Sample Code

The following python app make some sample of what you can do with workspaces:

```
import wasdi

def run():

    # Get The Active Workspace Id
    sWorkspaceId = wasdi.getActiveWorkspaceId()
    wasdi.wasdiLog("WorkspaceId is: " + sWorkspaceId)

    # Get the name of a workspace from the id
    sWorkspaceName = wasdi.getWorkspaceNameById(sWorkspaceId)
    wasdi.wasdiLog("WorkspaceName is: " + sWorkspaceName)

    # Get all the user workspaces
    aoWorkspaces = wasdi.getWorkspaces()
    wasdi.wasdiLog("User has " + str(len(aoWorkspaces)) + " Workspaces")

    # get the last workspace (we have at least one, this one!)
    oLastWorkspace = aoWorkspaces[len(aoWorkspaces)-1]
    wasdi.wasdiLog("Last Workspace is: " + oLastWorkspace["workspaceName"])

    # Open the last workspace
    sNewWorkspaceId = wasdi.openWorkspace(oLastWorkspace["workspaceName"])
    if sNewWorkspaceId == "":
        wasdi.wasdiLog("Error opening the last workspace")
    else:
        wasdi.wasdiLog("Now active workspace is " + wasdi.getActiveWorkspaceId())

    # Re open the original workspace
    wasdi.openWorkspaceById(sWorkspaceId)
    wasdi.wasdiLog("Re Opened the original workspace " + wasdi.
    ↪getWorkspaceNameById(wasdi.getActiveWorkspaceId()))

if __name__ == '__main__':
    wasdi.init('./config.json')
    run()
```

At the beginning we read the Id of the active workspace (**getActiveWorkspaceId**). This is defined in the config file and is the workspace where your code is running. Then we get the name of this workspace (**getWorkspaceNameById**).

We want next to get the list of the users' workspaces (**getWorkspaces**): this method returns a list of dictionaries: each object has these properties

```
"ownerUserId": STRING,
"sharedUsers": [STRING],
"workspaceId": STRING,
"workspaceName": STRING
```

Next step is to open another workspace (**openWorkspace**): this method returns the workspaceId if ok, an empty string in case of error. Finally, we come back to our original workspace using the id we collected before (**openWorkspaceById**) and verify using its name (**getWorkspaceNameById**).

The output will be something similar to this: .. code-block:

```
WorkspaceId is: a5dc8f79-3e89-46b5-8d39-169e9ecb0a98
WorkspaceName is: TutorialWorkspace
User has 108 Workspaces
Last Workspace is: S3_Day_ActiveFire
Now active workspace is ab34e55b-d233-466b-983e-223b42915869
Re Opened the original workspace TutorialWorkspace
```

4.6.4 Products functionalities

The functionalities to work with products are:

- get the list of products in a workspace
- check if a product is in the workspace or not
- get the local path of the product
- add a new product to the workspace

4.6.5 Products Sample Code

The following python app make some sample of what you can do with products.

To make it run, you should create a workspace and put there at least one file using the WASDI Search web user interface or the upload.

Please note that this code can take some time to be executed the first time you run it beacuse it shows how to access file locally (so download) and to upload results in WASDI.

Note: The goal of this tutorial is not to manipulate files so, the “new” file, is created just making a copy of an existing one with a different name.

```
import wasdi
import os
from shutil import copyfile

def run():

    # Get the list of file names
    aoProducts = wasdi.getProductsByActiveWorkspace()
    wasdi.wasdiLog("In the workspace we have " + str(len(aoProducts)))

    # Make sure we have at least one
    if len(aoProducts)>0:
        # Double check
        bCheck = wasdi.fileExistsOnWasdi(aoProducts[0])
        wasdi.wasdiLog("Product " + aoProducts[0] + " is on workspace? " +
↪str(bCheck))

        # This line will return the local path: it assume you need it to open
↪the image, so the first time will automatically download the image
```

(continues on next page)

(continued from previous page)

```

sLocalPath = wasdi.getPath(aoProducts[0])

# Generate the name of a new file, not existing yet: start taking the
↳original file without extension
sCopyLocalPath = os.path.splitext(sLocalPath)[0]
# add _copy and re-put extension
sCopyLocalPath = sCopyLocalPath + "_copy" + os.path.
↳splitext(sLocalPath)[1]
# Make a local copy, as it was another file
copyfile(sLocalPath, sCopyLocalPath)

# Get only the file name
sCopiedFileName = os.path.basename(sCopyLocalPath)
wasdi.wasdiLog("We 'created' a second new file: " + sCopiedFileName)
# Add the file to wasdi: this will upload the new file to the cloud
wasdi.addFileToWASDI(sCopiedFileName)

wasdi.wasdiLog("Tutorial Done!")

if __name__ == '__main__':
    wasdi.init('./config.json')
    run()

```

The code starts taking a list of the products in the workspace (**getProductsByActiveWorkspace**). Just to show the functionality, it then checks if the first file is really available on WASDI (**fileExistsOnWasdi**).

The next step is to simulate a local file access: to open a file, you need a full local path: this must be requested to WASDI (**getPath**).

The same function can be used also to obtain a path to use to save your own file: our code just makes a copy of a file in a workspace with another name, using again **getPath** to have to path to use to save the file.

This copy is a new file for WASDI: to add it to the workspace use **addFileToWASDI**: please note that add file to WASDI takes as input only the file name and not the full path.

4.7 Synchronous and Asynchronous WASDI programming

4.7.1 Introduction

This tutorial is designed to explain in detail the Processes life cycle of WASDI, the meaning and difference between synch and asynch programming.

In this tutorial we use PyCharm as a free Python Development tool, but the code can be ran on every different Python environment.

It is out of scope of this Tutorial how to set up the environment and start programming with WASDI. For this you can refer to:

- Wasdi Libraries Concepts
- Python Tutorial

4.7.2 WASDI Processes

WASDI supports different operations. Each operation is called Process Workspace, since it is a process that is executed in a Workspace.

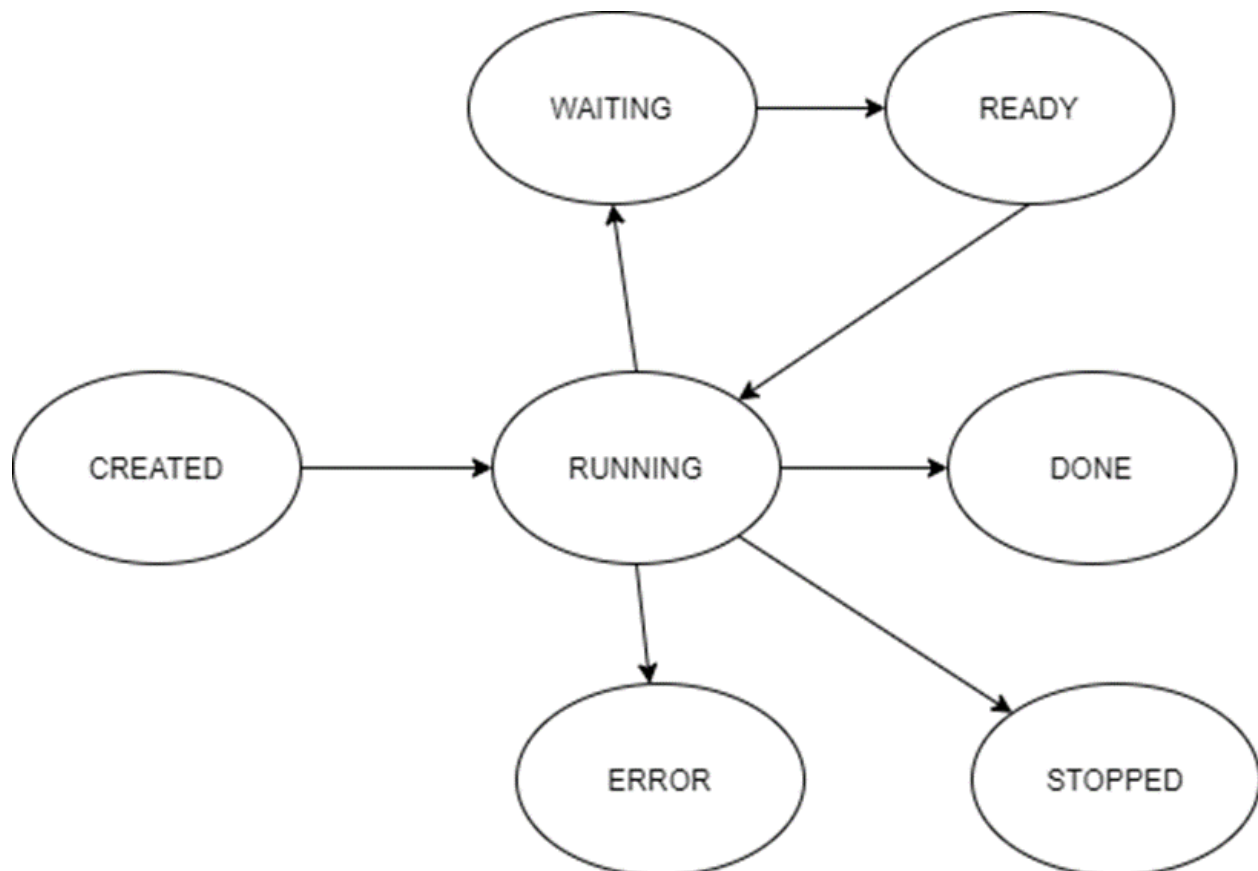
Operations are a sort of Class, while Process Workspaces are Objects; the operations are what WASDI can do, a Process Workspace is an instance of one operation in a Workspace.

WASDI Operations are (the list is indicative):

- Download: more in general fetch of a Product (Image) in WASDI. The image can be really downloaded or accessed using the file system, depending by the Data Type and the cloud that host the computing node
- Ingest: this is the import of an image uploaded by the user or generated by an application
- Run Processor: with a sub type for each language, is the operation that runs a user supplied application
- Graph: executes a SNAP graph on the cloud
- Mosaic: mosaic different products in a single new product
- Multisubset: subset one single image in one or more subsets (smaller parts of the original image)
- Sen2Core: executes the SNAP sen2core processor on the cloud

There are other operations that are more useful for the platform that for the developers and are out of the scope of this tutorial.

Each WASDI process has a state. The processes states are:



- **CREATED:** each process workspace, when is triggered, goes in the CREATED state waiting for the scheduler to run it

- **RUNNING**: the process is up and running
- **WAITING**: the process is waiting for some other operation (process workspace) to finish
- **READY**: the process finished to wait for other operation and now is waiting for the scheduler to restart it
- **DONE**: the process finished with success
- **STOPPED**: the process has been stopped by the user or by some timeout
- **ERROR**: the process ended in error

These states are handled like a String, in all the libraries. Some methods can return the status of an operation or set it; in both cases the state is a string.

We can divide these states in three categories:

- Pre Execution: CREATED is the only pre-execution state
- During Execution: RUNNING, WAITING and READY are the three states that means that the operation is executing
- Post Execution: DONE, STOPPED and ERROR are the three states that means that the operation is finished

4.7.3 Synchronous vs Asynchronous

Synchronous and Asynchronous are concepts that are valid in all the IT development world, not only in WASDI.

With Synchronous, we mean that process A starts operation B and the execution of process A does not proceed until operation B is finished.

With Asynchronous, we mean that process A starts operation B and the execution of process A proceed immediately without waiting that operation B is finished.

These concepts are very important to optimize applications, specially in a cloud environment like WASDI where, **your application, if it is 'well written' in terms of synch or asynch operations, can scale up very quickly and use the full power of the cloud.**

In general, you need to use synch operations if the result of the operation you are running is needed to proceed and you can not do nothing else in the meanwhile.

You are suggested to use asynch operations when you do not need immediately the result of the operation to proceed while in the meanwhile you can do something else.

In general, in WASDI, almost all the operations have two version: Operation and asynchOperation. For example we can use:

```
#Synch Version
wasdi.importProduct(oProduct)
```

or

```
#Asynch Version
wasdi.asynchImportProduct(oProduct)
```

All the synch versions, returns a string (or an array of strings) with the output status of the requested operation.

All the asynch versions, returns a string (or an array of strings) with ProcessWorkspaceId of the triggered operation.

Since each rule is done to be broke, All BUT ONE: executeProcessor, to run another wasdi app from your code, is ALWAYS an asynch call.

There are two methods in the library that can be used to re-synchronize the execution after the use of asynch operation:

```
#Wait a single process: takes in input a string with the procId of the process to wait
wasdi.waitForProcess(sProcessId)
```

```
#Wait a list processes: takes in input an array of strings with the procId of the
→ processes to wait
wasdi.waitForProcess(arrayOfProcessIds)
```

Both functions returns the state of the processes in input: this will be one of the Post Execution States (“DONE”, “ERROR” or “STOPPED”).

4.7.4 Download Sample

Lets imagine that our application needs to import some images and apply some algorithm. We can imagine two situations to show the difference between synch and asynch operations.

Let start with the case where our applications needs only one specific image in input: in this case, the image is needed... we need to import it and then we need to wait. Nothing to do. For this example, we make a search and then we use the first found image.

```
#Search Images
aoFound = wasdi.searchEOImages("S1", sDateFrom="2021-02-01", sDateTo="2021-02-02",
→ sProductType="GRD", fULLat=44.5, fULLon=8.5, fLRLat=44.0, fLRLon=9.0)
# Double check we have one
if len(aoFound) > 0:
    wasdi.wasdiLog("Import Image 0")
    #Import the image
    sState = wasdi.importImage(aoFound[0])
    wasdi.wasdiLog("Import finished with status: " + sState)
```

If you run this snippet of code, you will see that execution breaks at the importImage line; you can check live on the wasdi user interface that in that workspace wasdi will start a download and, when is done, the control of the code will return to the log line.

Lets imagine instead that we need to retrieve the full list of products. In this case, if we use the synch version, WASDI will trigger the execution of one download per time and will not use the ability of the cloud. Instead, if we use an asynch version, we can request all our list of images and then wait for all them to finish: in this case we push WASDI to download in parallel as many images as possible:

```
#Search Images
aoFound = wasdi.searchEOImages("S1", sDateFrom="2021-02-01", sDateTo="2021-02-02",
→ sProductType="GRD", fULLat=44.5, fULLon=8.5, fLRLat=44.0, fLRLon=9.0)

# Here we will keep the list of process id that we started
asOperationsIds = []

# Double check we have one
if len(aoFound) > 0:
    #For all our found images
    for oProduct in aoFound:
        wasdi.wasdiLog("Import Image " + oProduct["fileName"])
        # Start the import the image without waiting
        sOperationId = wasdi.asynchImportImage(oProduct)
        # Add the proc id to the list of the ones we need to wait
```

(continues on next page)

(continued from previous page)

```

asOperationsIds.append(sOperationId)

# This line will be executed without waiting the images
wasdi.wasdiLog("All import triggered, wait images")
# Now, we stop and wait
wasdi.waitProcesses(asOperationsIds)
# This line will be executed when all are imported
wasdi.wasdiLog("All images imported")

```

This code snippet is for didacting reason only: this functionality is already implemented in WASDI in a single function of the lib.

```

#Search Images
aoFound = wasdi.searchEOImages("S1", sDateFrom="2021-02-01", sDateTo="2021-02-02",
↪sProductType="GRD", fULLat=44.5, fULLon=8.5, fLRLat=44.0, fLRLon=9.0)

# Import all the images using the max power of the cloud
wasdi.importProductList(aoFound)

# This line will be executed when all are imported
wasdi.wasdiLog("All images imported")

```

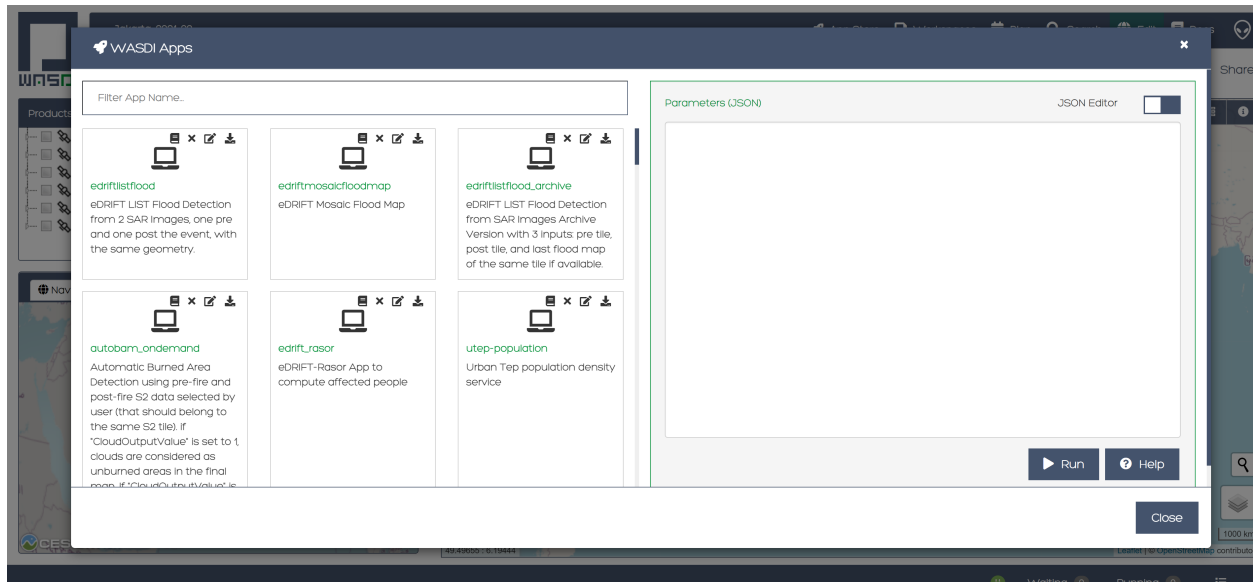
4.7.5 Start Other Applications

One of the most powerful feature of WASDI is the ability from one application to call another one. First of all, this means that in WASDI we have a full language interoperability: it does not matter in which language you are developing, you can call apps made in IDL, Java, Python or Matlab with the same syntax and same results.

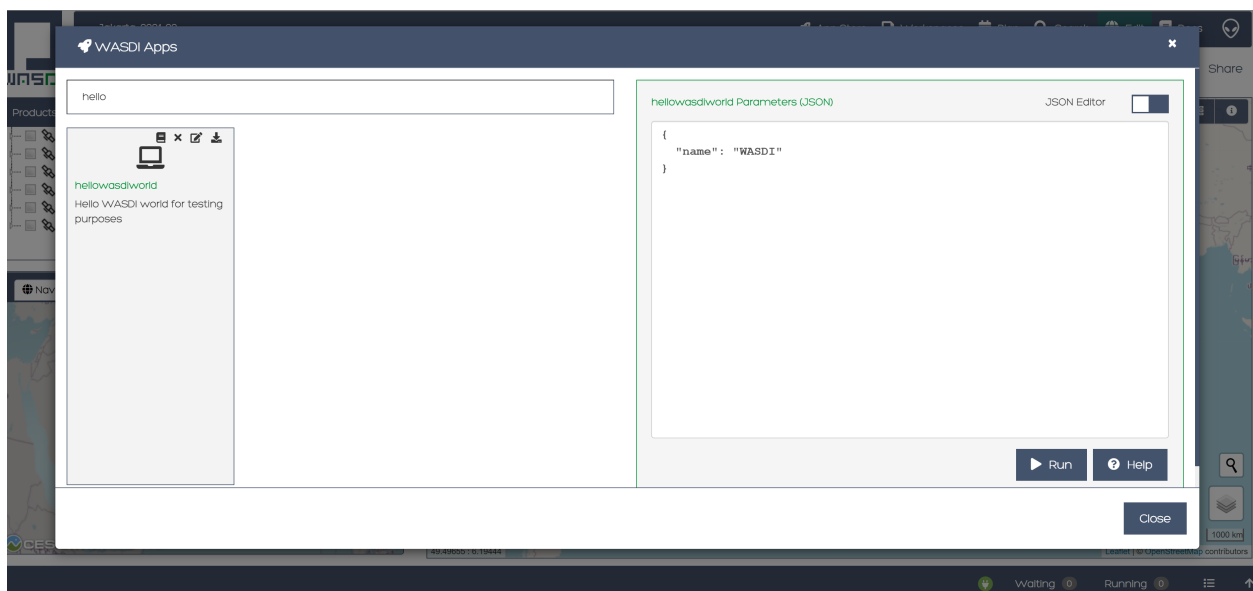
As for your application the input is represented by your params.json, the same is for the others.

So to call another application in WASDI, you have to prepare a Dictionary that has a key for each parameter of the application you want to call, and assign the relative desired value.

Applications has two view in WASDI: * App Store: is the Space Market of WASDI, desinged for end users * Advanced App view: from the Edit section (just open a workspace), you can access the Apps view



From the apps view usually, each developer declares the sample json input required:



In the image, the hellowasdiworld app shows that takes a NAME parameter as input.

Usually, developers also add an help file to their applications where they declare the different parameters.

These are the info you need to call another app.

As it has been stated before, to execute another application is always an asynch operation.

```
#Prepare Params
helloParams = {}
helloParams["NAME"]="Synch Asynch Tutorial"

#Call the hellowasdiworld application
sProcessId = wasdi.executeProcessor("hellowasdiworld", helloParams)
```

(continues on next page)

(continued from previous page)

```
#Here you can do anything else!  
  
#Wait for the application to finish  
wasdi.waitProcess(sProcessId)  
  
wasdi.wasdiLog("Hello WASDI World finshed")
```

This snippet of code is the core for the optimization of your application.

4.7.6 Suggested WASDI App Organization

In our experience the best way to develop your wasdi application is an old advise: “Dividi et Impera”.

Usually WASDI applications are developed to manipulate satellite data to obtain a value added product in output.

To obtain the value added product, your algorithm may need to take in input on single image, or a pair, or a list of images. In any case, usually, you can individuate the base brick of your algorithm that takes in input only the images needed and produces one or more output.

The suggestion is to start writing this first base processor: it can be developed and tested manually, using WASDI web interface to search, upload, import, preprocess or whatever is needed to prepare the input for you. The params.json should declares the images needed in input and any other specific param of your algorithm.

Once this processor is ready, an automation processor can be build upon it: usually, the automation processor, takes in input date an bounding box and not the exact image to use: in this wrapper-processor you can search EO Images, filter results, apply workflows, mosaic, subsets, conversions, whatever is needed to run your base processor.

When the data is ready, you can start in parallel as many instances as possibile of your base processor and then wait for all the different instances to finish and, maybe, if needed, mosaic or summarize the results that you will find in the workspace.

Using this technique usually let you take the best advantage from the execution of your code in the cloud. Also, it ensure you to have your code more portable: the base application takes in input only file names and can be quickly adapted to other systems or platforms, and you can isolate your dependency by WASDI only in the automation code.

Welcome to Space, Have fun!

4.8 C# Tutorial

4.8.1 Prerequisites

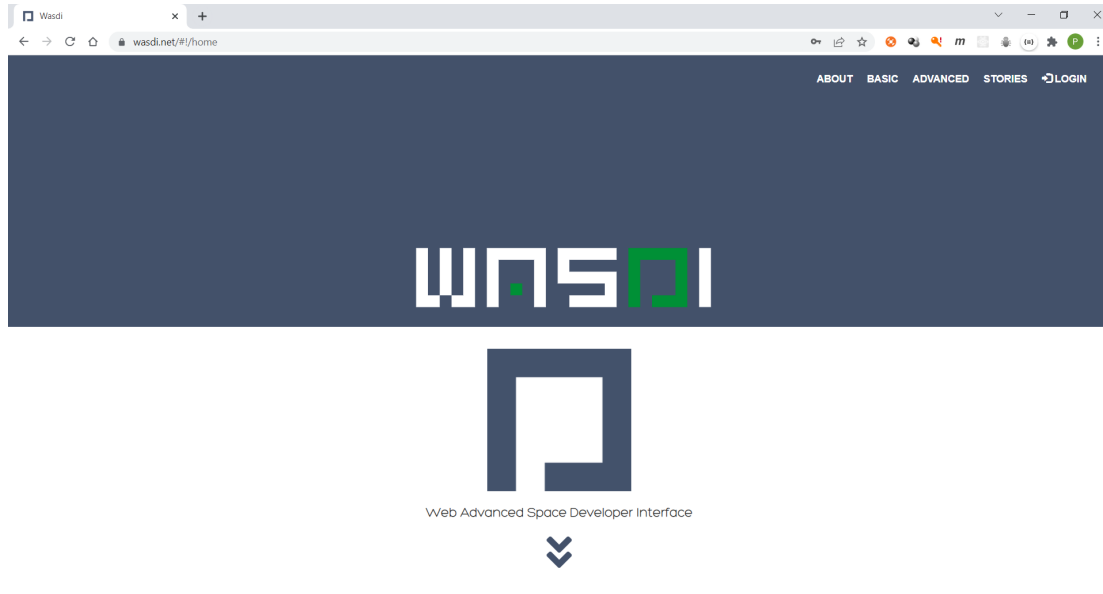
Note: To make the most of this tutorial, prior experience with the WASDI platform is required.

For new users, it is highly recommended to follow the [Wasdi Web Platform access and basic usage](#) tutorial before continuing.

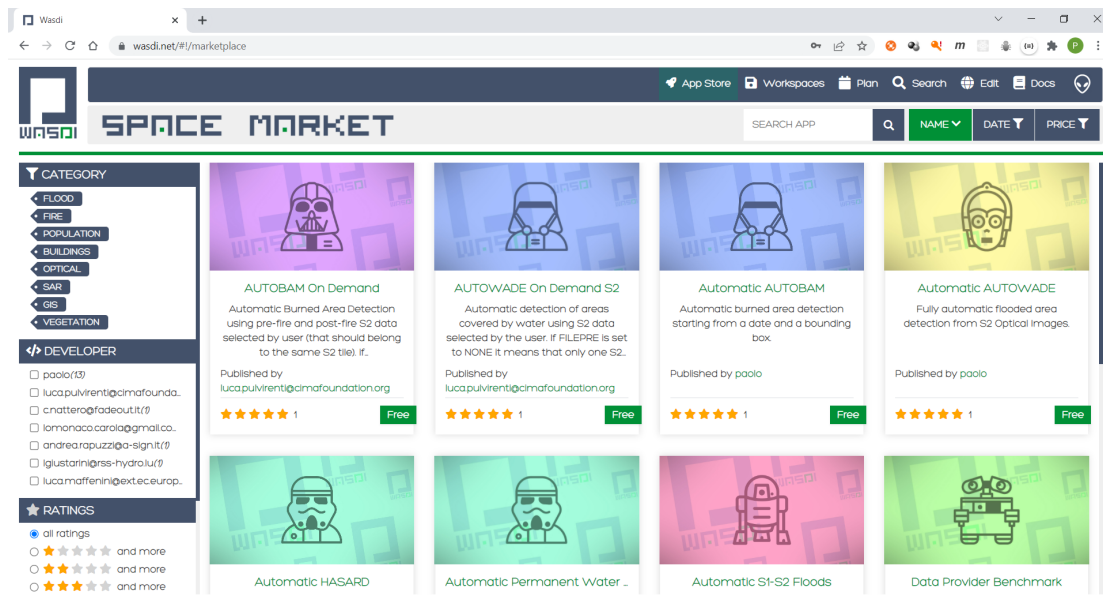
4.8.2 Setup

Setup on Wasdi web-app side

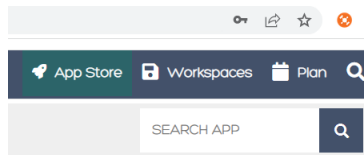
Open the Wasdi web-application in a browser by typing <https://www.wasdi.net/>.



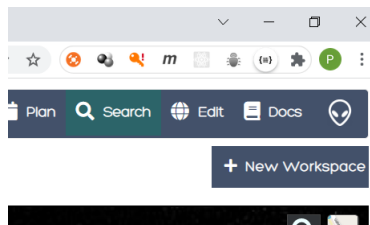
On the main page, the marketplace is shown (Space Market).



Go to the Workspaces page by pressing the **Workspaces** button.

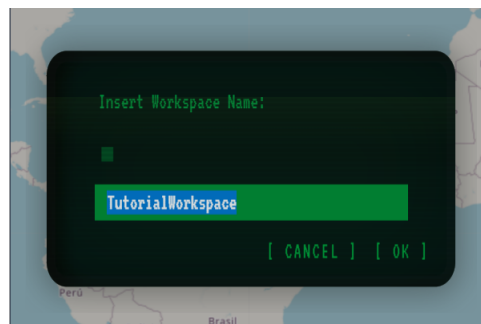


Create a new workspace by pressing the **New Workspace** button.

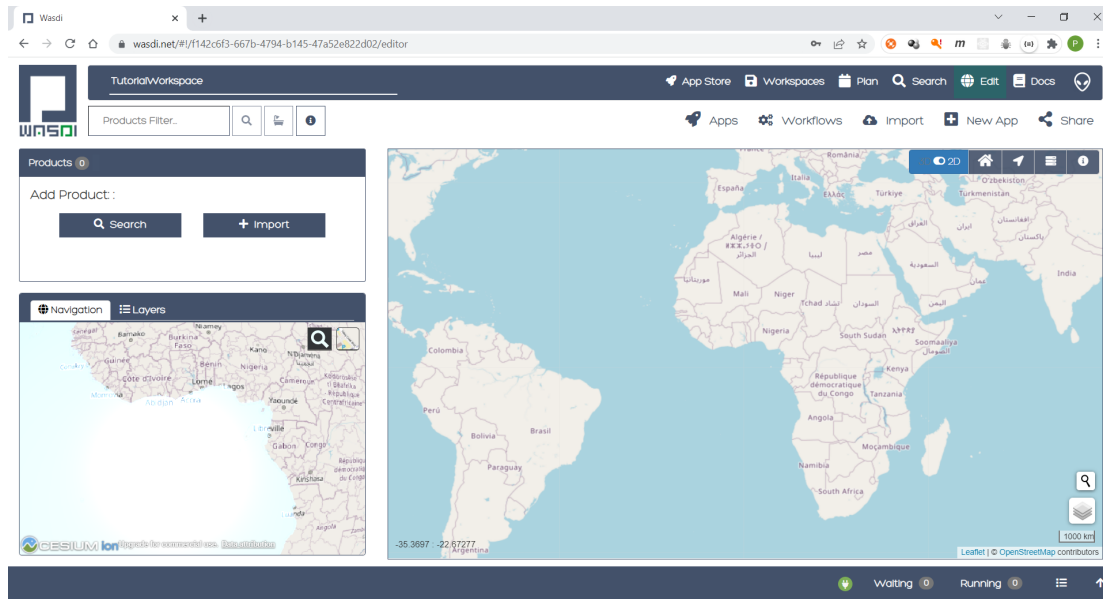


On the pop-up window, specify the name of the new workspace and press **OK**. For this tutorial, I will choose **TutorialWorkspace**.

Note: Although it is not mandatory, it is strongly recommended to use the indicated name. This would allow the code to be copy/pasted without adjustments.

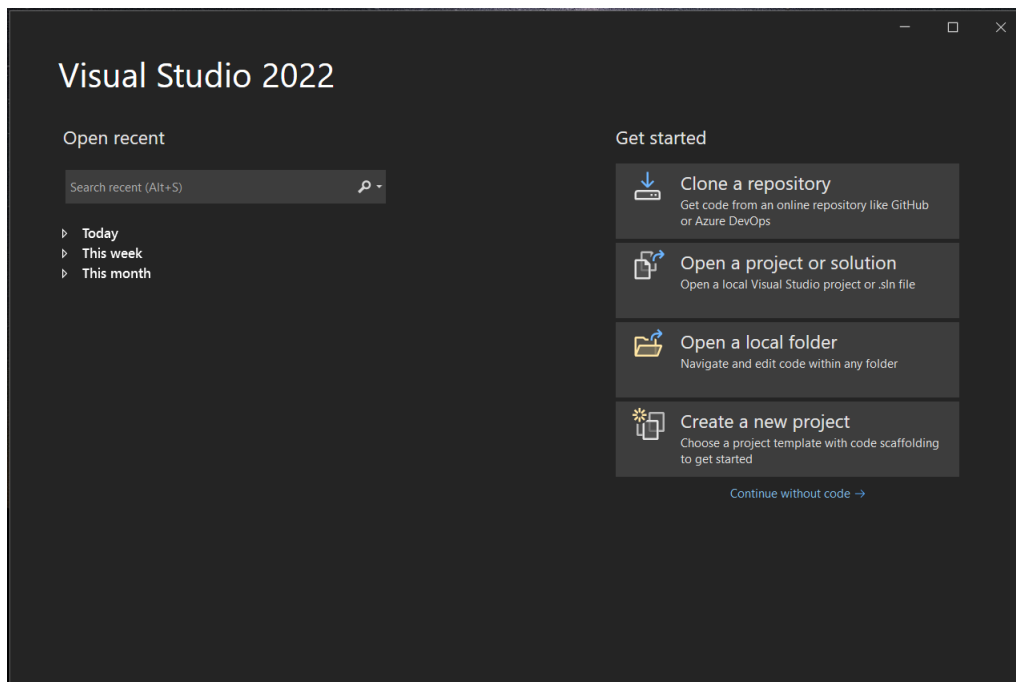


The newly created workspace is shown on the main page.

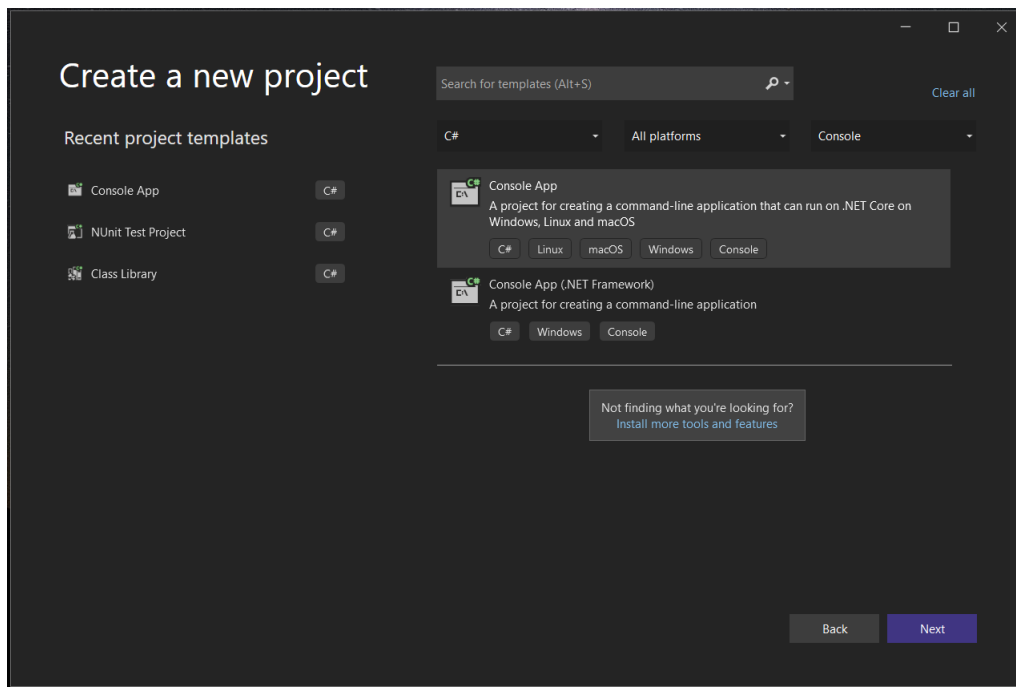


Setup on Microsoft's Visual Studio side

Open Microsoft's Visual Studio



Create a new project (Console App).



For this tutorial, choose **TutorialSeeSharpApp**.

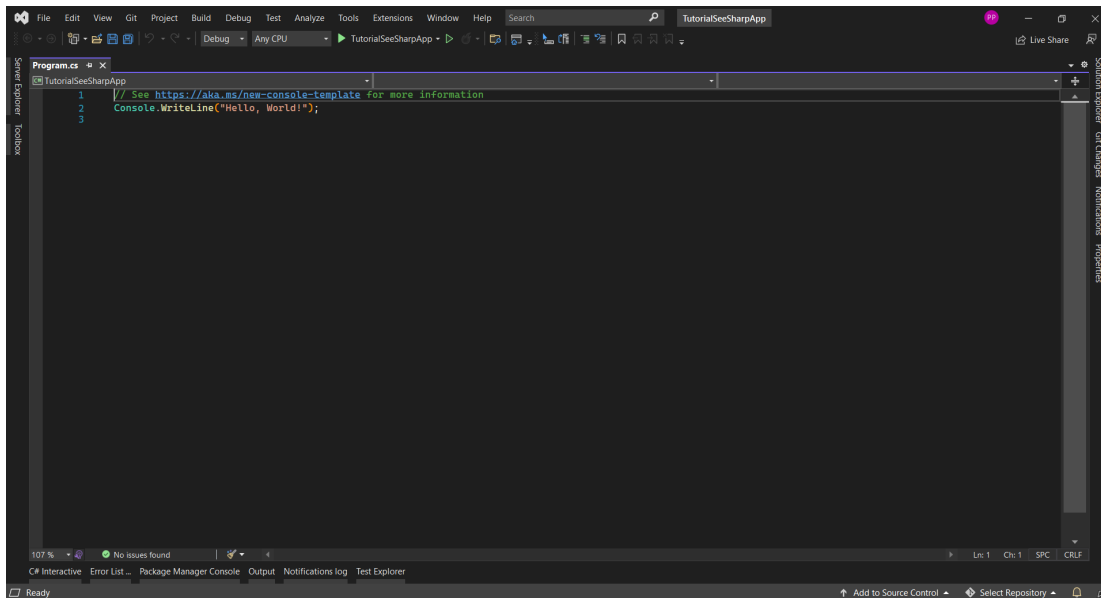
Note: Although it is not mandatory, it is strongly recommended to use the indicated name. This would allow the code to be copy/pasted without adjustments.

The screenshot shows the 'Configure your new project' window. At the top, it says 'Configure your new project'. Below that, there are tabs for 'Console App', 'C#', 'Linux', 'macOS', 'Windows', and 'Console'. The 'Console App' tab is selected. Under 'Project name', the text 'TutorialSeeSharpApp' is entered. Under 'Location', the text 'C:\work\dev\' is entered. Under 'Solution name', the text 'TutorialSeeSharpApp' is entered. There is a checkbox labeled 'Place solution and project in the same directory' which is checked. At the bottom right, there are 'Back' and 'Next' buttons.

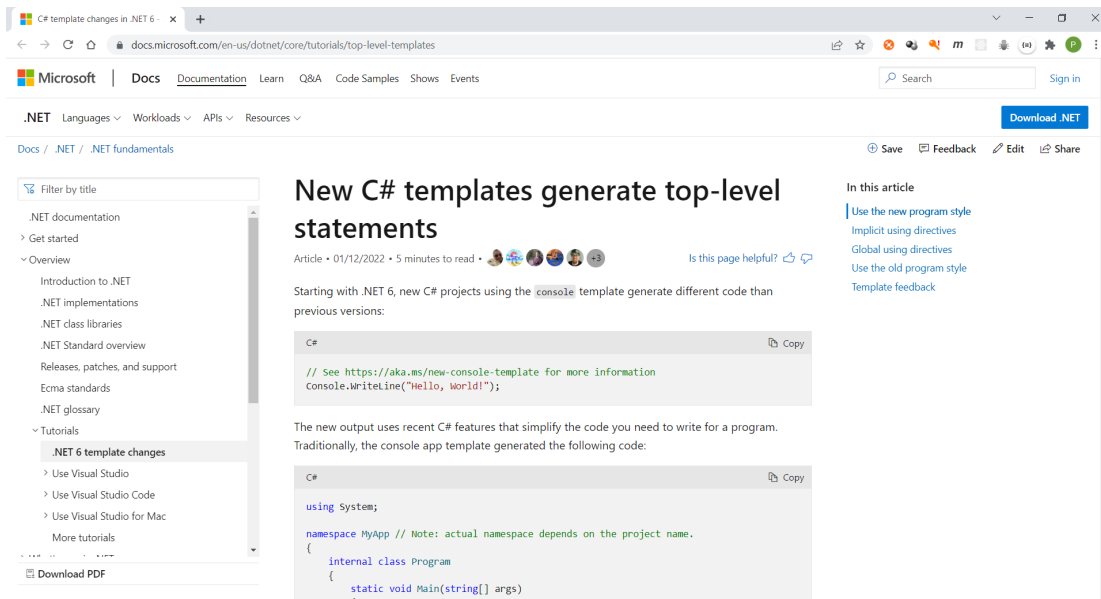
Choose the desired framework. I will accepted the default option (.NET 6.0).

The screenshot shows the 'Additional information' window. At the top, it says 'Additional information'. Below that, there are tabs for 'Console App', 'C#', 'Linux', 'macOS', 'Windows', and 'Console'. The 'Console App' tab is selected. Under 'Framework', the text '.NET 6.0 (Long-term support)' is entered. At the bottom right, there are 'Back' and 'Create' buttons.

Pressing the **Create** button, as the setup is complete, the MS Visual Studio will open the project.



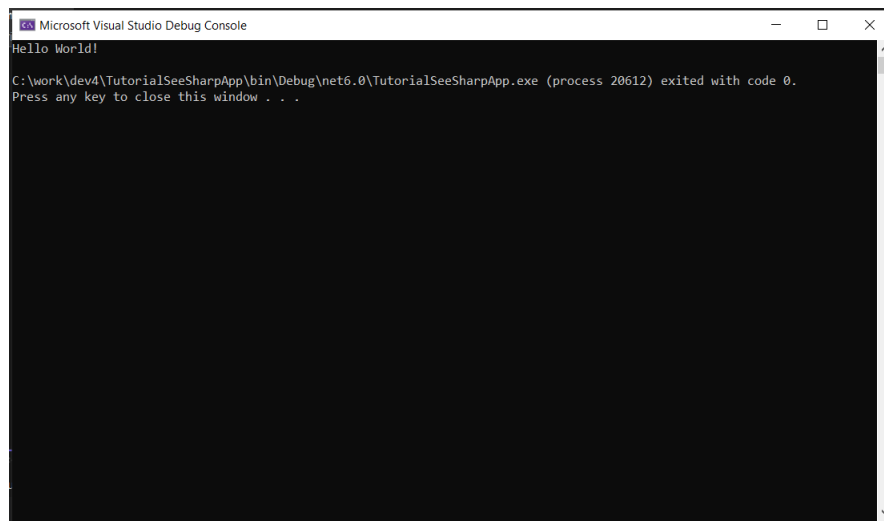
Create a full-fledged main class. Use the URL provided to obtain the code. Copy and paste it to replace the generated stub.



Change the name name of the namespace to match the name of the project (**TutorialSeeSharpApp**).

```
namespace TutorialSeeSharpApp
{
    internal class Program
    {
        static void Main(string[] args)
        {
            Console.WriteLine("Hello World!");
        }
    }
}
```

Run the program to verify that everything is fine.

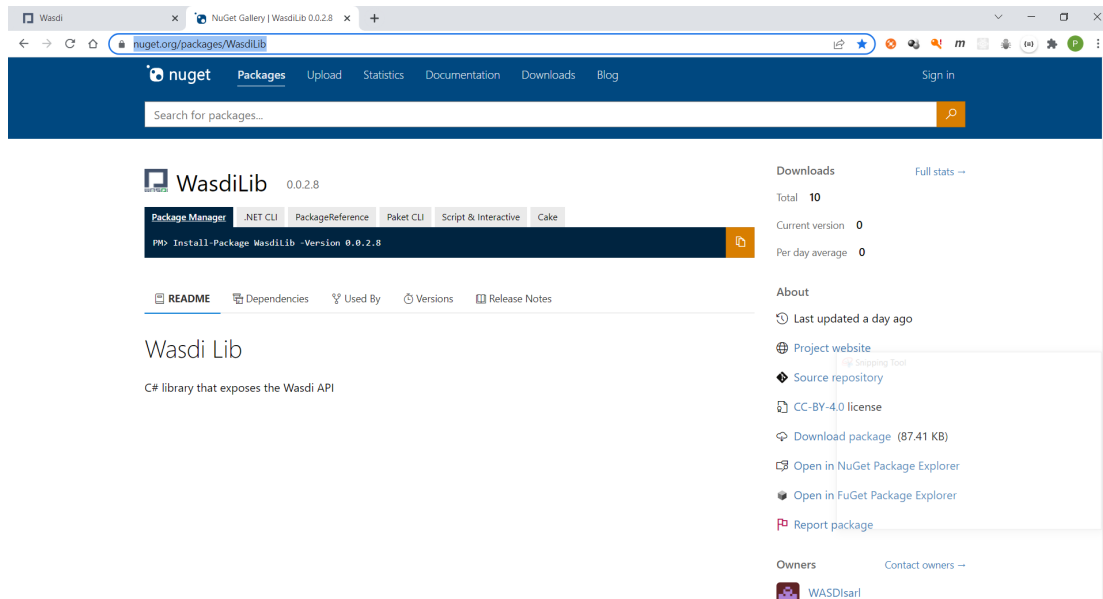


4.8.3 Work with WASDI

Add the WasdiLib dependency to your application

Note: The code showed in this section can also be found on the dedicated public GitHub repository: <https://github.com/wasdi-cloud/TutorialSeeSharpApp>.

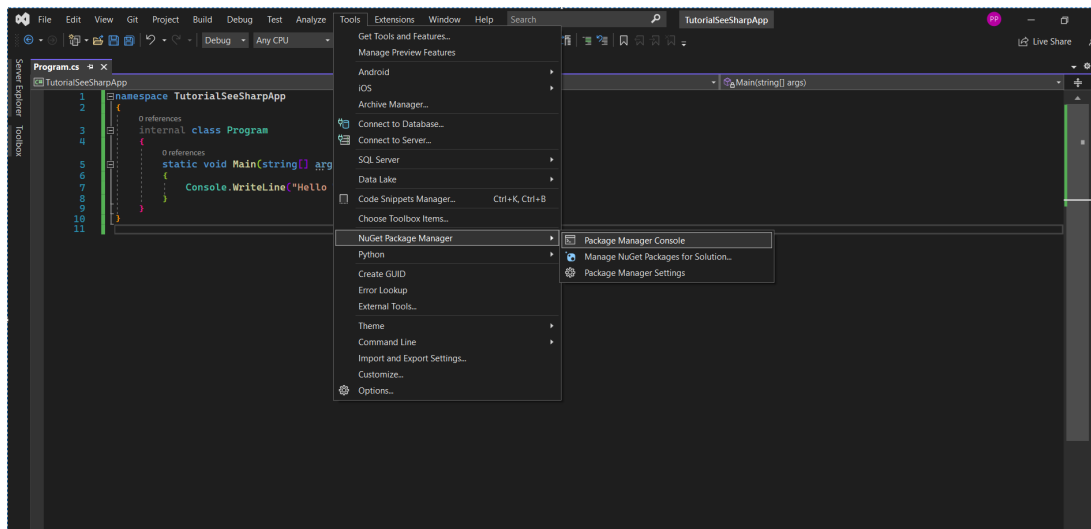
Find the WasdiLib on NuGet. Open a page in a browser and navigate to <https://www.nuget.org/packages/WasdiLib>.



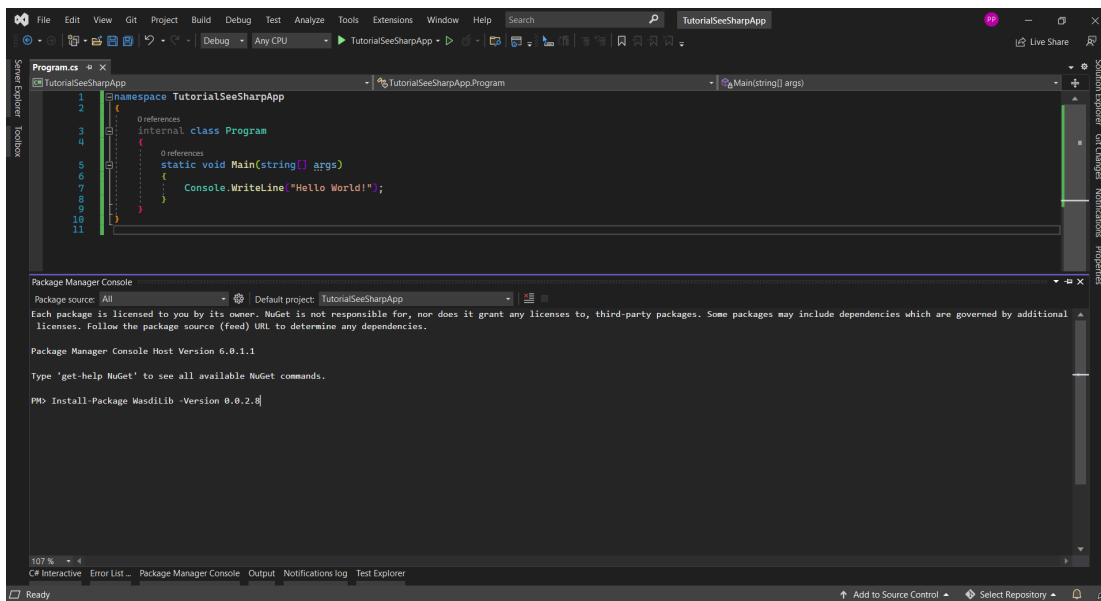
Copy the installation command for the latest version (by pressing the orange button):

```
Install-Package WasdiLib -Version 0.0.3.5
```

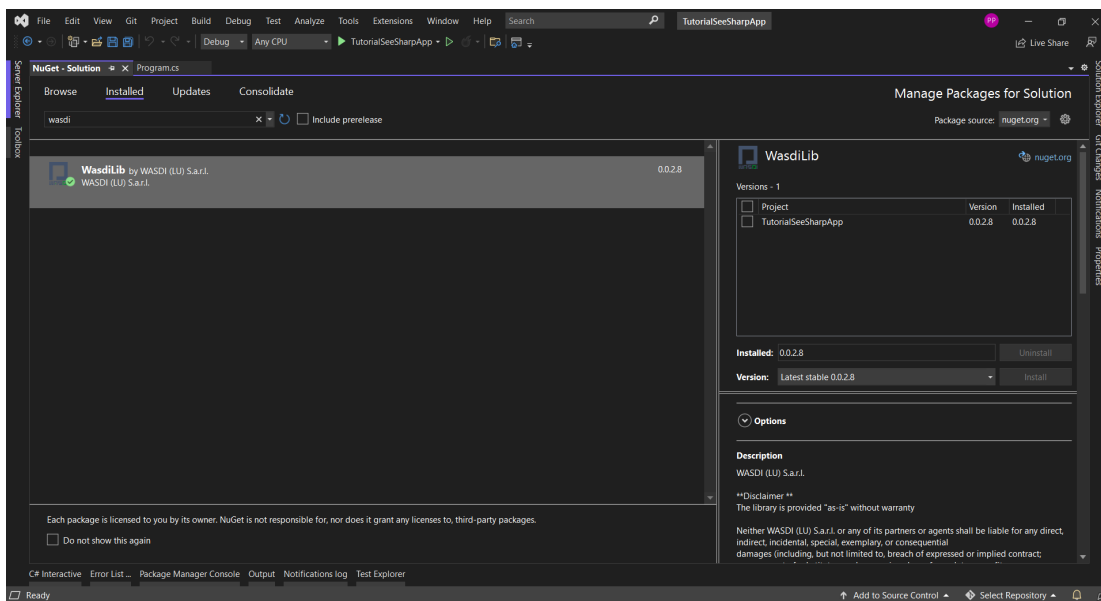
Open the NuGet Package Manager console (Tools > NuGet Package Manager) and paste in the command just copied.



Install the WasdiLib as a dependency of your console application.



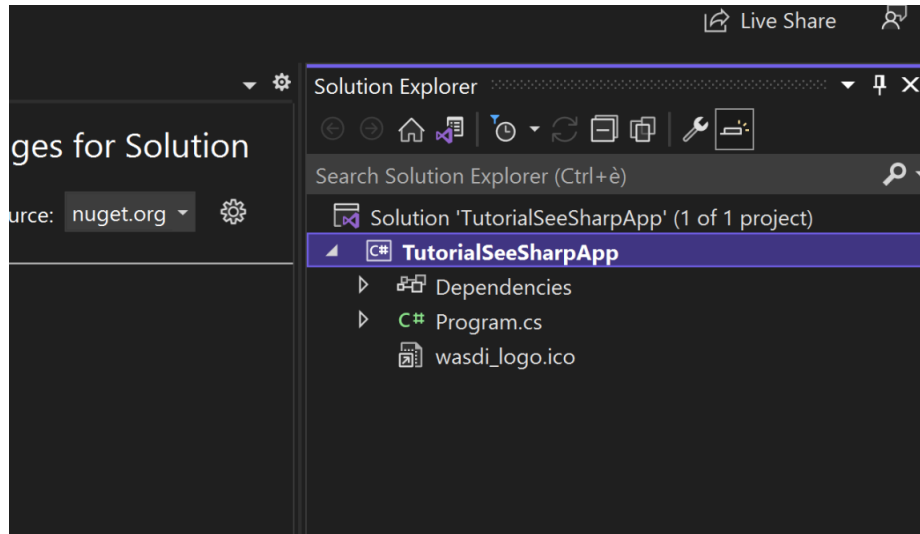
Alternatively, the WasdiLib can be installed through NuGet Package Manager (Tools > Manage NuGet Packages for Solution).



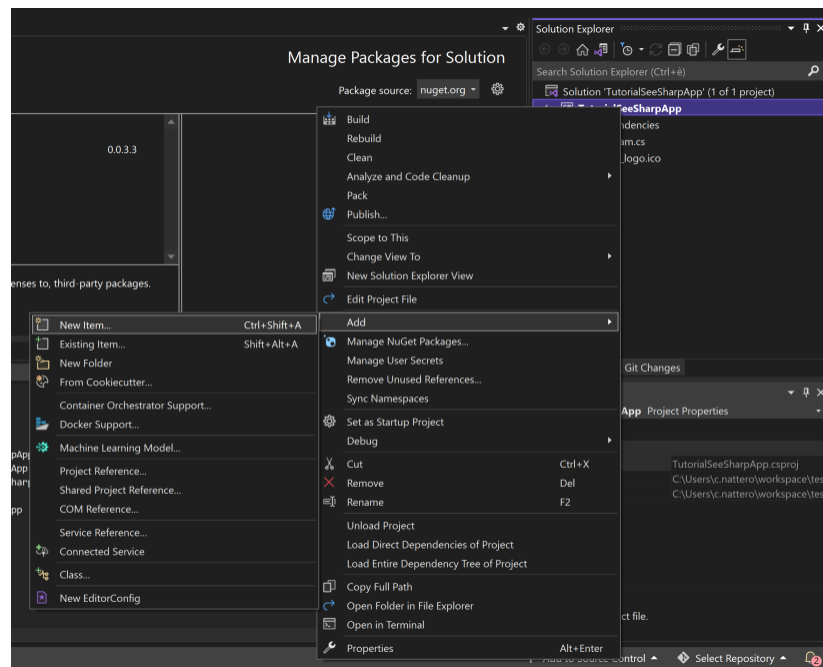
Create configuration files

Create appsettings.json

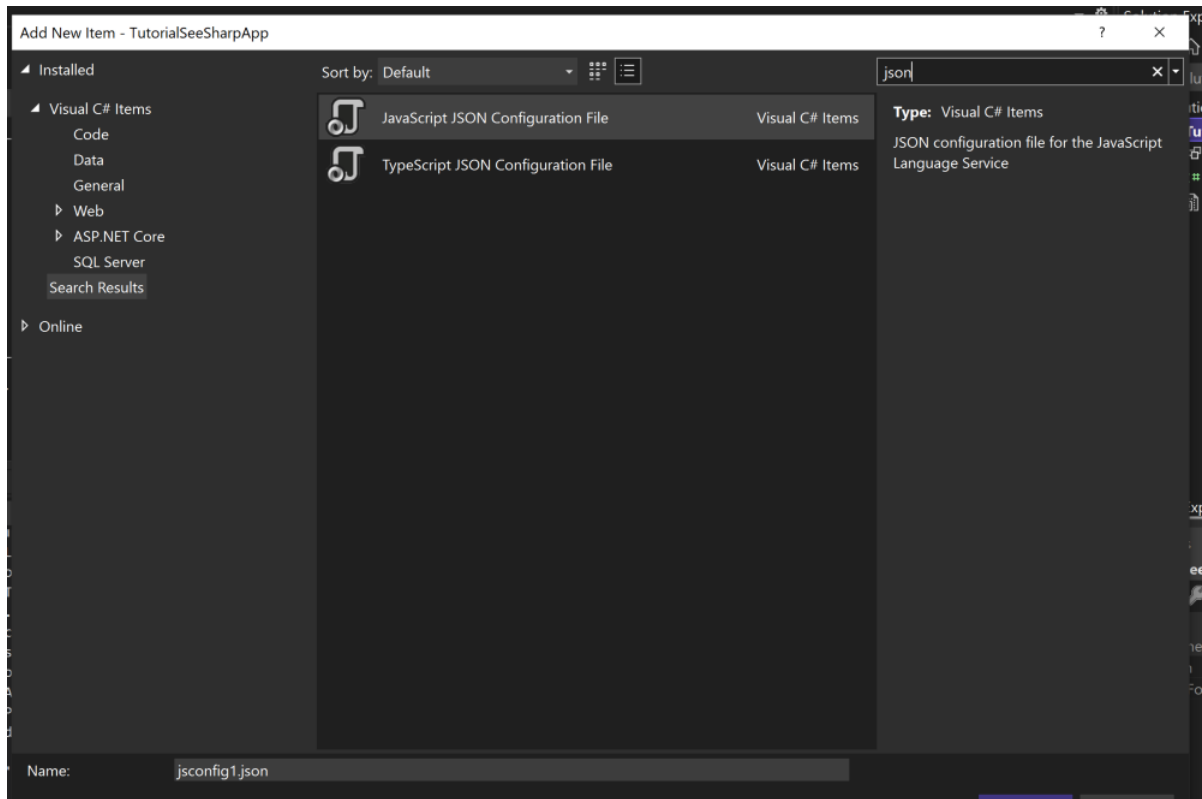
To add a file to the application, right click on the **TutorialSeeSharpApp**.



Select **Add** and then **New Item**.



Select the type of file and input the name.



The **appsettings.json** file contains the information required to connect to the Wasdi server. In absence of such information, the library cannot connect to the server, in development mode. Once the application is deployed on the Wasdi server, it will obtain the required information from the user session. Therefore, for development use, please do not forget to input your credentials on the **appsettings.json** file.

```
{
  "USER": "your_username",
  "PASSWORD": "your_password",
  "BASEPATH=": "C:/temp/wasdi/",
  "BASEURL": "https://www.wasdi.net/wasdiwebserver/rest",
  "WORKSPACE": "TutorialWorkspace",
  "PARAMETERSFILEPATH": "./parameters.json"
}
```

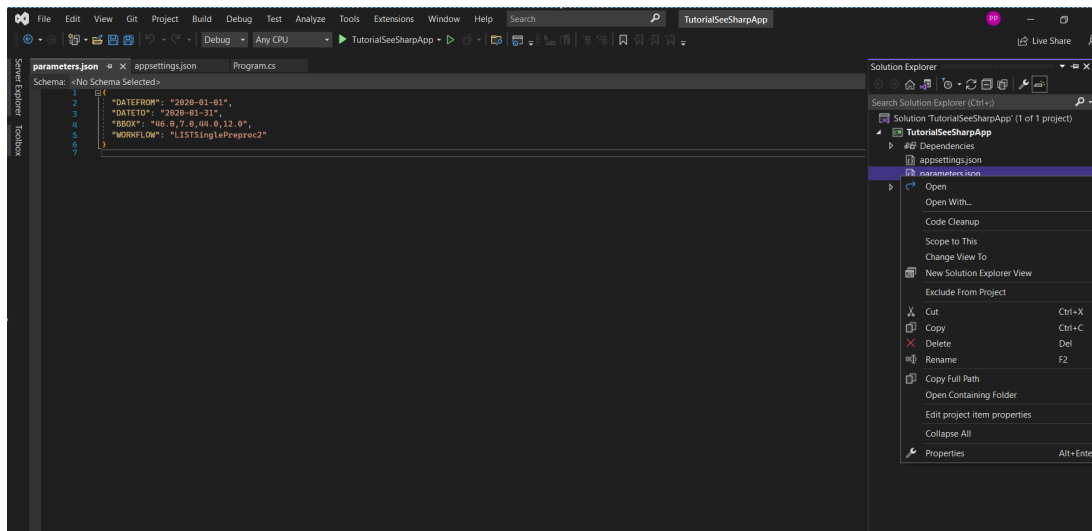
Create parameters.json

The **parameters.json** file contains the information related to the operation conducted on the Wasdi server.

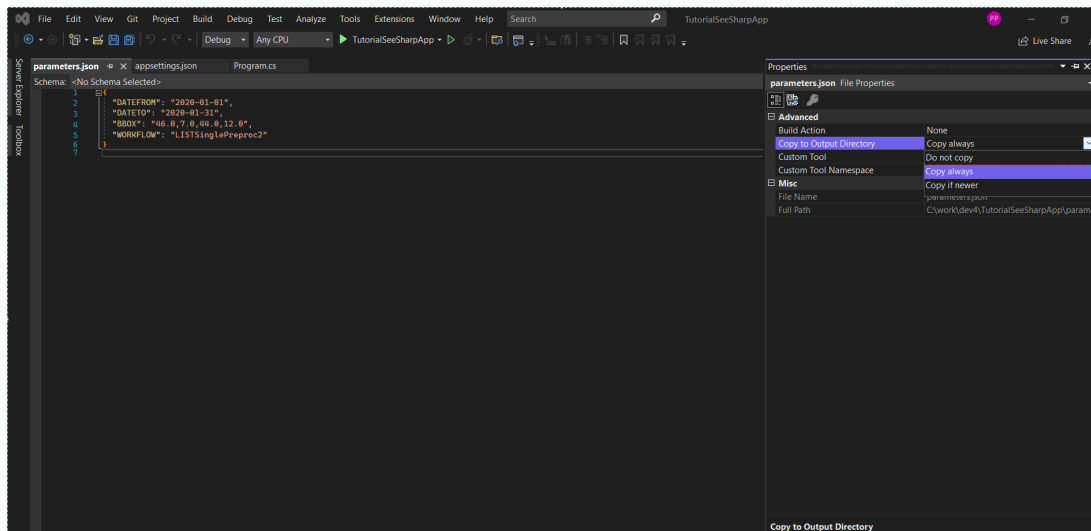
```
{
  "DATEFROM": "2020-01-01",
  "DATETO": "2020-01-31",
  "BBOX": "46.0,7.0,44.0,12.0",
  "WORKFLOW": "LISTSinglePreproc2"
}
```

Note: The properties of both these configuration files should be changed to trigger their copy each time the project is built.

Repeat this procedure for both files (appsettings.json and parameters.json): On the right side-bar, click on **Solution Explorer**. Select the file, right-click on it, Select the last option, **Properties**.



On the **Advanced** section, change the value of the property **Copy to output directory** to **Copy always**.



Verify the setup

Call the /hello endpoint

The application can run locally as a stand-alone application (with a **Main** method). However, in order for the application to run on the Wasdi platform, the class must meet two conditions:

- implement the **IWasdiRunnable** interface and override its **Run** method;
- have a no-arg constructor (if the class does not have an explicit constructor, the compiler will add a default no-arg constructor at compile time);

Note: It is strongly recommended for the application to have the structure shown below.

To connect to the Wasdi server through the WasdiLib, an object of type Wasdi must be created and initialized in the **Main** method and passed as an argument to the **Run** method. The verbosity of the logging mechanism could be increased, in order to see on the console the result.

The actual call to the Wasdi object should be done either form inside the **Run** method or from any other method called by **Run**.

```
using WasdiLib;

namespace TutorialSeeSharpApp
{
    internal class Program : IWasdiRunnable
    {
        static void Main(string[] args)
        {
            Wasdi wasdi = new();
            wasdi.Init();
            wasdi.SetVerbose(true);
```

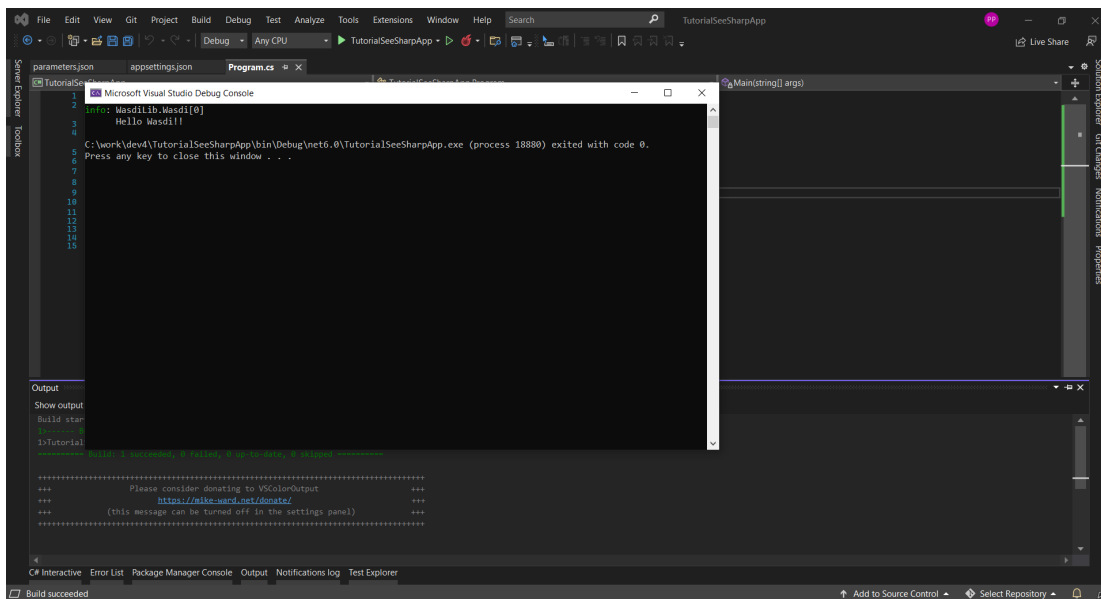
(continues on next page)

(continued from previous page)

```
        Program program = new Program();
        program.Run(wasdi);
    }

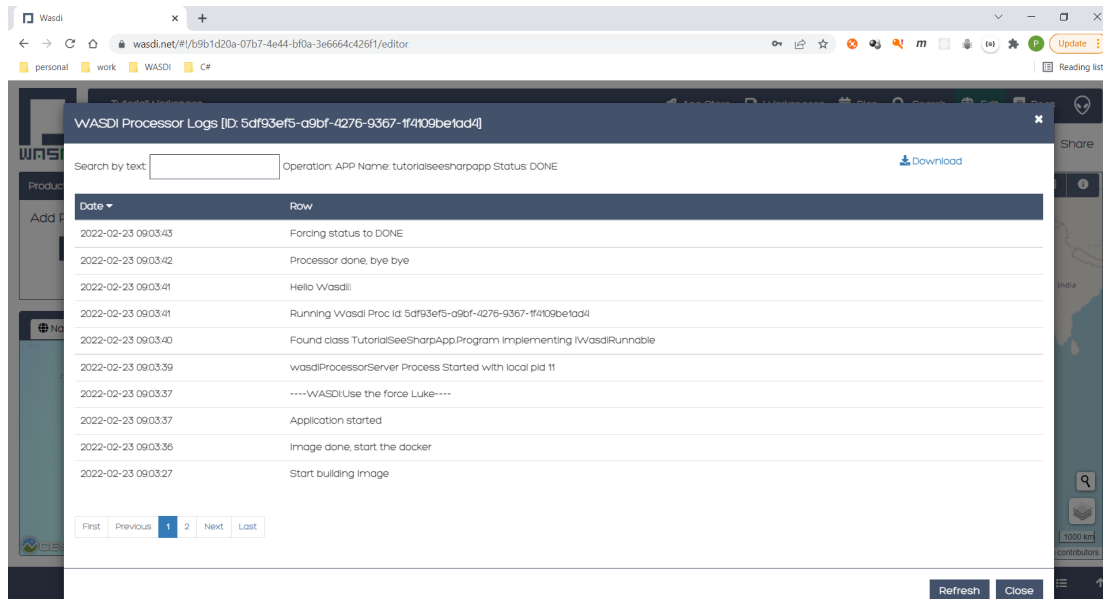
    public void Run(Wasdi wasdi)
    {
        wasdi.WasdiLog(wasdi.Hello());
    }
}
```

The outcome of running the program locally is a console window showing the Wasdi greeting.



Note: The procedure to deploy the application on the WASDI platform is described later in this tutorial.

Running the same program on the Wasdi platform produces the following outcome.



Get the user's workspaces' names

The following program retrieves the names of the workspaces that the user has access to. An user can access a workspace either if the workspace was created by the user or if the workspace was shared by another user.

```
using WasdiLib;

namespace TutorialSeeSharpApp
{
    internal class Program : IWasdiRunnable
    {
        static void Main(string[] args)
        {
            Wasdi wasdi = new();
            wasdi.Init();
            wasdi.SetVerbose(true);

            Program program = new Program();
            program.Run(wasdi);
        }

        public void Run(Wasdi wasdi)
        {
            GetWorkspacesNames(wasdi);
        }

        private static void GetWorkspacesNames(Wasdi wasdi)
        {
            wasdi.WasdiLog("GetWorkspacesNames()");
        }
    }
}
```

(continues on next page)

(continued from previous page)

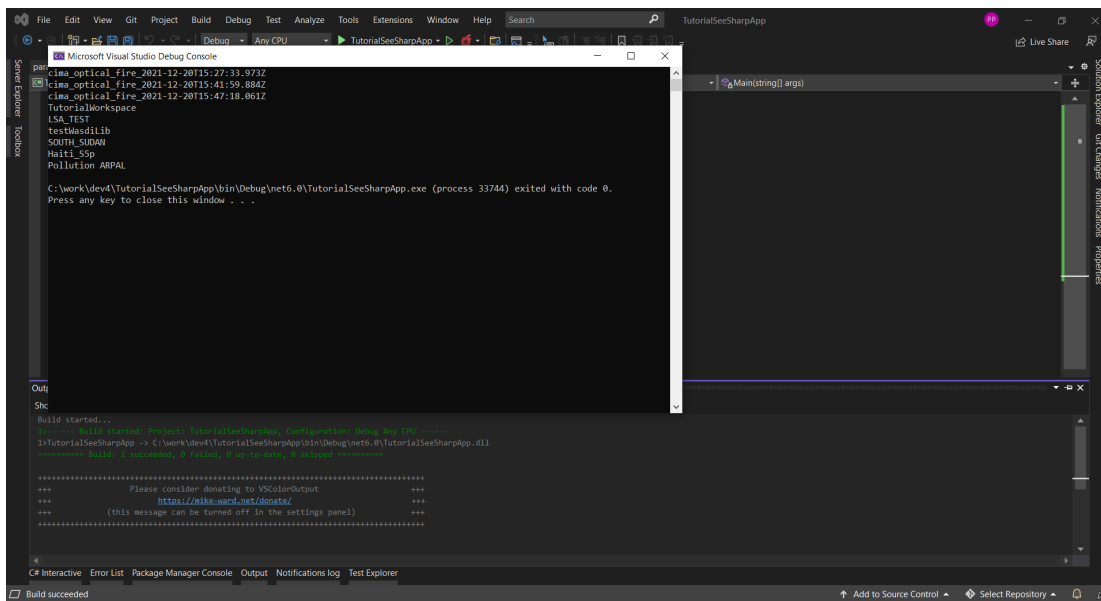
```

        List<string> workspacesNames = wasdi.GetWorkspacesNames();

        foreach (string workspaceName in workspacesNames)
        {
            wasdi.WasdiLog(workspaceName);
        }
    }
}

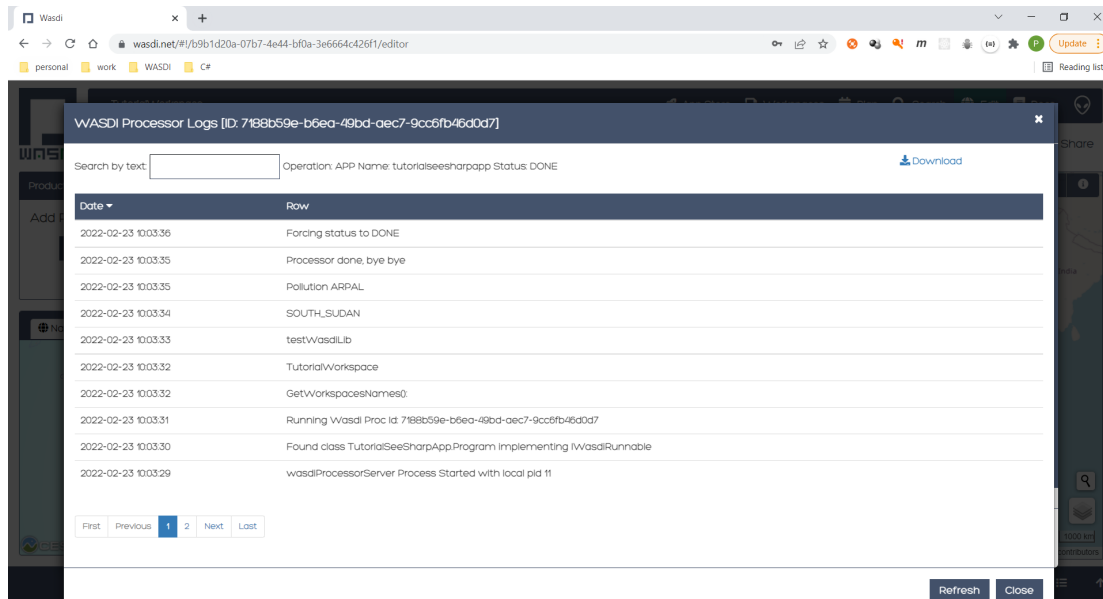
```

Running the program locally should show in the console the list of workspaces' names. At least **TutorialWorkspace** should be present.



Note: The procedure to deploy the application on the WASDI platform is described later in this tutorial.

Running the same program on the Wasdi platform produces the following outcome.



Running the new C# application on Wasdi platform

It's great to have the application running locally but the end-goal is to have the application running on Wasdi server.

Writing the application

In order to see the application producing some effects, two operations are triggered: the execution of an workflow and the execution of a processor.

```
using WasdiLib;
using WasdiLib.Models;

namespace TutorialSeeSharpApp
{
    internal class Program : IWasdiRunnable
    {
        static void Main(string[] args)
        {
            Wasdi wasdi = new();
            wasdi.Init();
            wasdi.SetVerbose(true);

            Program program = new Program();
            program.Run(wasdi);

            UpdateStatus(wasdi);
        }

        public void Run(Wasdi wasdi)
```

(continues on next page)

(continued from previous page)

```

    {
        RunExecuteWorkflow(wasdi);

        RunExecuteProcessor(wasdi);
    }

private static void RunExecuteWorkflow(Wasdi wasdi)
{
    string sStartDate = wasdi.GetParam("DATEFROM");
    string sEndDate = wasdi.GetParam("DATETO");
    string sBbox = wasdi.GetParam("BBOX");
    string sWorkflow = wasdi.GetParam("WORKFLOW");

    double dLatN = 44.0;
    double dLonW = 35.0;
    double dLatS = 45.0;
    double dLonE = 36.0;

    if (sBbox != null)
    {
        String[] asLatLons = sBbox.Split(',');
        dLatN = Double.Parse(asLatLons[0]);
        dLonW = Double.Parse(asLatLons[1]);
        dLatS = Double.Parse(asLatLons[2]);
        dLonE = Double.Parse(asLatLons[3]);
    }

    wasdi.WasdiLog("Start searching images");
    List<QueryResult> aoResults = wasdi.SearchEOImages("S1",
↪sStartDate, sEndDate, dLatN, dLonW, dLatS, dLonE, "GRD", null, null, null);
    wasdi.WasdiLog("Found " + aoResults.Count + " Images");

    if (aoResults.Count > 0)
    {
        wasdi.ImportProduct(aoResults[0]);

        List<string> asInputs = new List<string>();
        asInputs.Add(aoResults[0].Title + ".zip");

        List<string> asOutputs = new List<string>();
        asOutputs.Add("preprocessed.tif");

        wasdi.ExecuteWorkflow(asInputs, asOutputs, sWorkflow);
    }
    wasdi.WasdiLog("FINISHED");
}

private static void RunExecuteProcessor(Wasdi wasdi)
{
    // call another app: HelloWasdiWorld
    Dictionary<string, object> dictionary = new Dictionary<string,

```

(continues on next page)

(continued from previous page)

```

↪object>()

        { { "name", wasdi.GetUser() } };

        wasdi.ExecuteProcessor("HelloWasdiWorld", dictionary);
    }

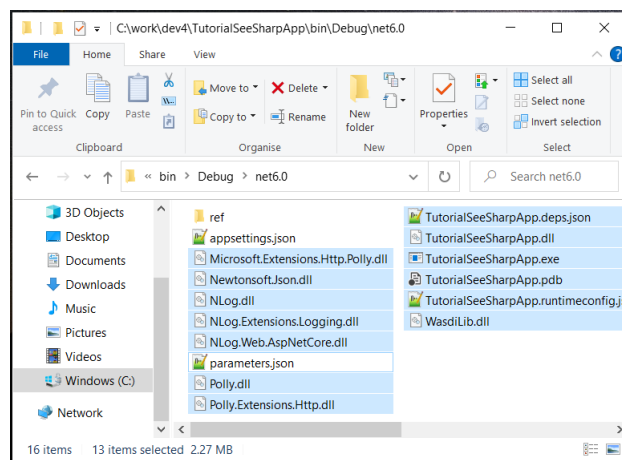
    private static void UpdateStatus(Wasdi wasdi)
    {
        wasdi.WasdiLog("UpdateStatus:");
        string sStatus = "DONE";
        int iPerc = 100;
        wasdi.UpdateStatus(sStatus, iPerc);
    }
}

```

Note: For applications that require heavy processing, it is recommended not to run locally but exclusively on the WASDI platform.

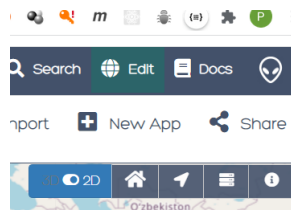
Packaging the application

To export the application, zip the content of the `\bin\Debug\net6.0` directory, except for the configuration files (`appsettings.json` and `parameters.json`) and the `ref` directory. The zip archive should share the name of the application, in my case **TutorialSeeSharpApp.zip**.

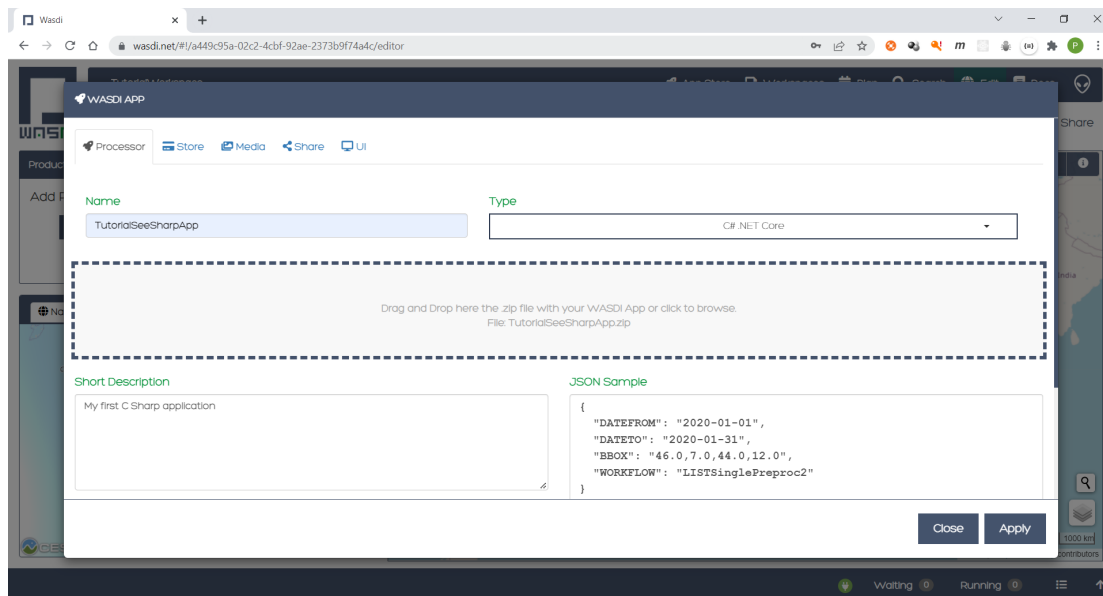


Deploying the application

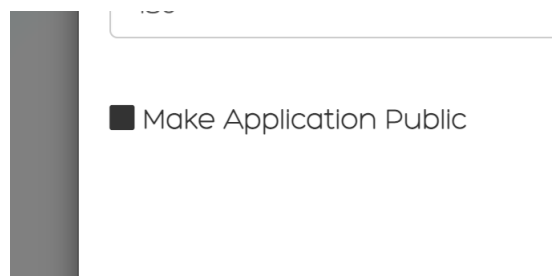
Back on the Wasdi web-application, create a new application by pressing the **New App** button.



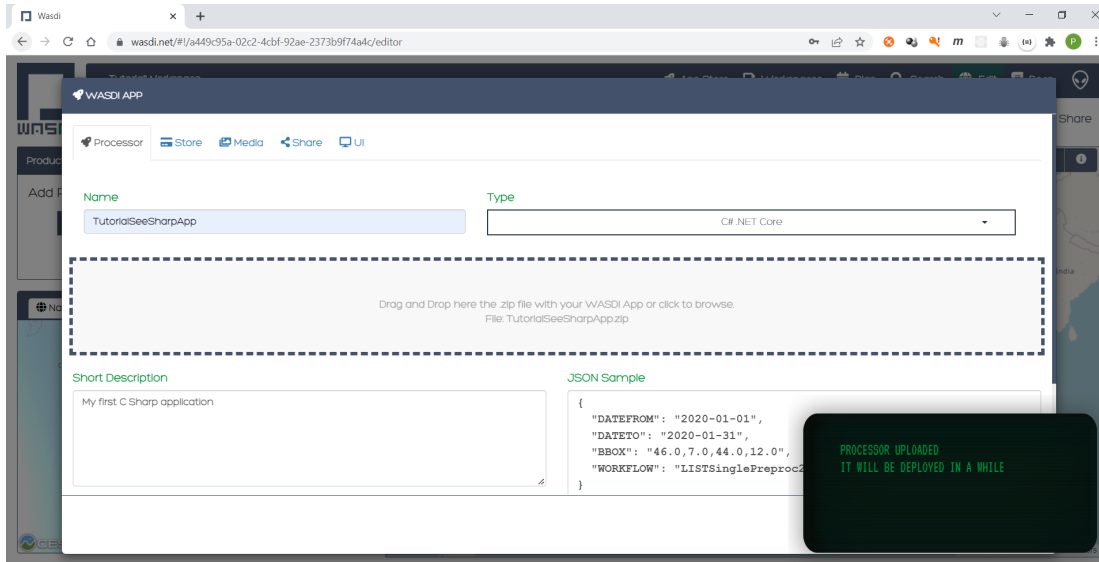
On the page that opens, fill in the details of the application, as shown in the image below.



Until the application is ready to be exposed to the public, the **Make Application Public** checkbox could be unchecked. To find this checkbox, scroll down to the bottom of the page.

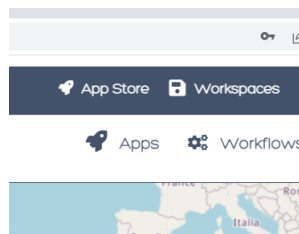


A message will be shown to inform the user that the application (processor) will be deployed shortly.

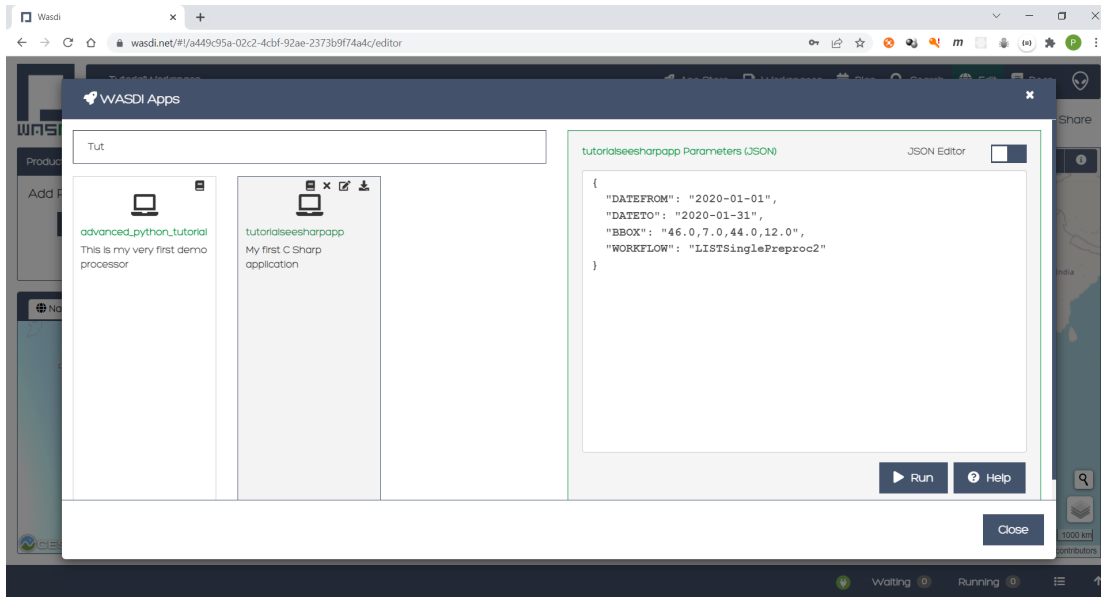


Viewing the application

Navigate to the applications page by pressing the **Apps** button. Search the newly created application by filtering the list.

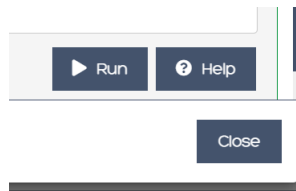


Type **Tutorial** and click on the application's card.

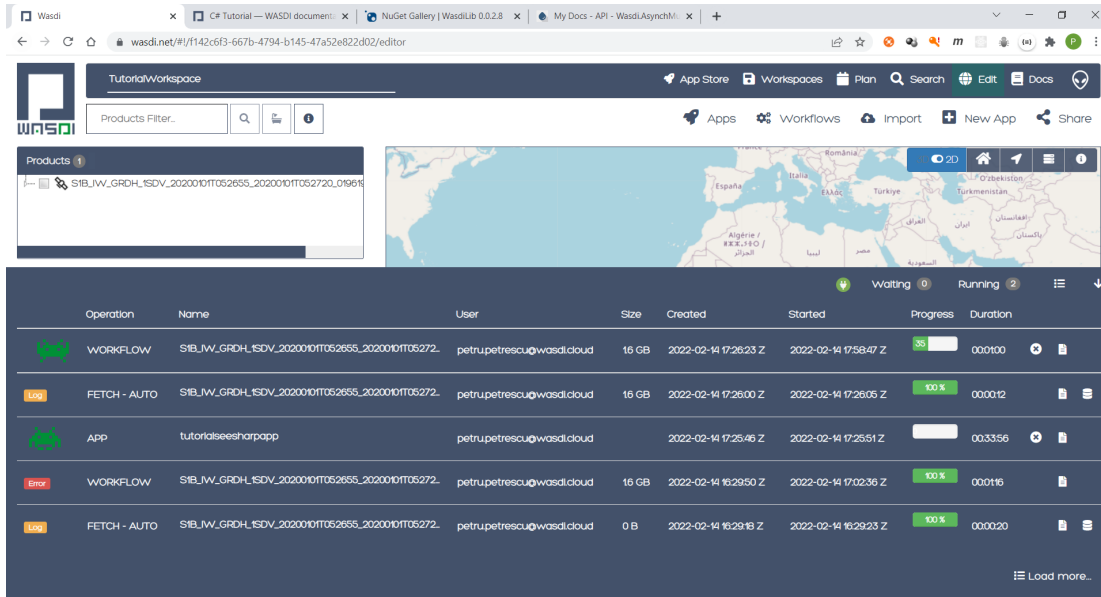


Running the application

Adjust the parameters of the application as needed and press the **Run** button.

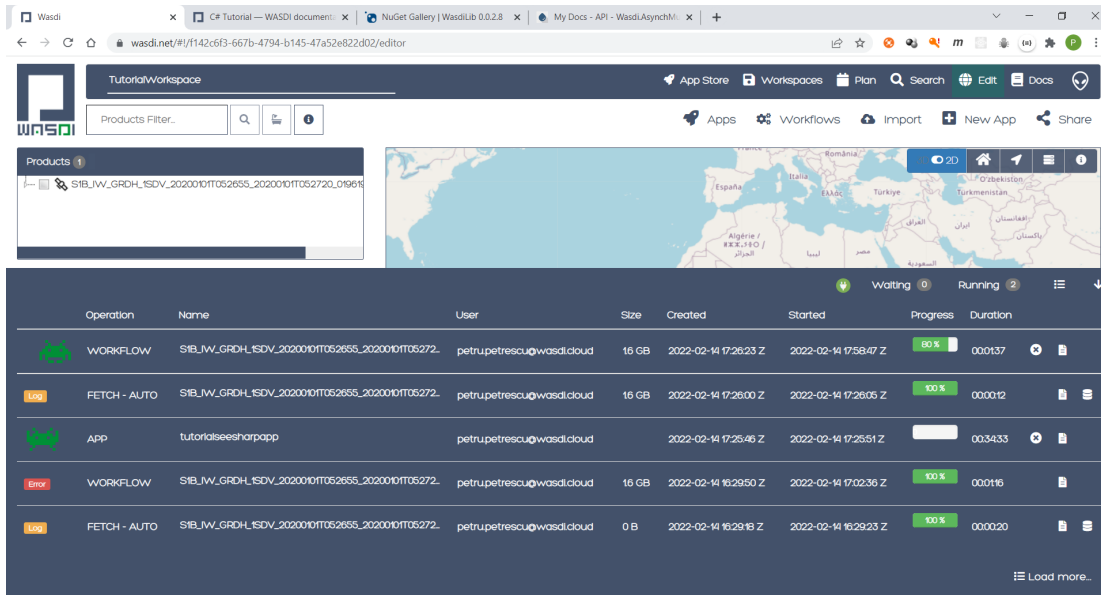


Depending on the load on the server, the deployed application starts executing in second or in minutes.



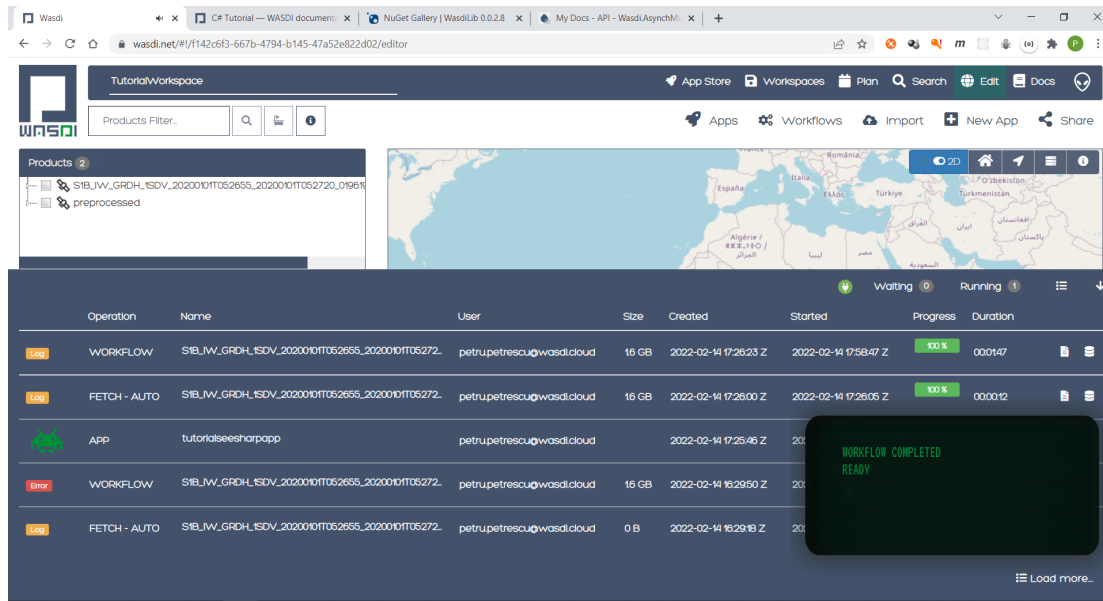
Operation	Name	User	Size	Created	Started	Progress	Duration
WORKFLOW	SIBJW_GRDH_ISDV_2020010T052655_2020010T05272...	petru.petrescu@wasdi.cloud	16 GB	2022-02-14 17:26:23 Z	2022-02-14 17:58:47 Z	33%	00:01:00
FETCH - AUTO	SIBJW_GRDH_ISDV_2020010T052655_2020010T05272...	petru.petrescu@wasdi.cloud	16 GB	2022-02-14 17:26:00 Z	2022-02-14 17:26:05 Z	100%	00:00:12
APP	tutorialseesharpapp	petru.petrescu@wasdi.cloud		2022-02-14 17:25:46 Z	2022-02-14 17:25:51 Z	0%	00:33:56
WORKFLOW	SIBJW_GRDH_ISDV_2020010T052655_2020010T05272...	petru.petrescu@wasdi.cloud	16 GB	2022-02-14 16:29:50 Z	2022-02-14 17:02:36 Z	100%	00:01:16
FETCH - AUTO	SIBJW_GRDH_ISDV_2020010T052655_2020010T05272...	petru.petrescu@wasdi.cloud	0 B	2022-02-14 16:29:18 Z	2022-02-14 16:29:23 Z	100%	00:00:20

Also, the duration of the execution may vary. The bar and the percentage show to the user the progress.

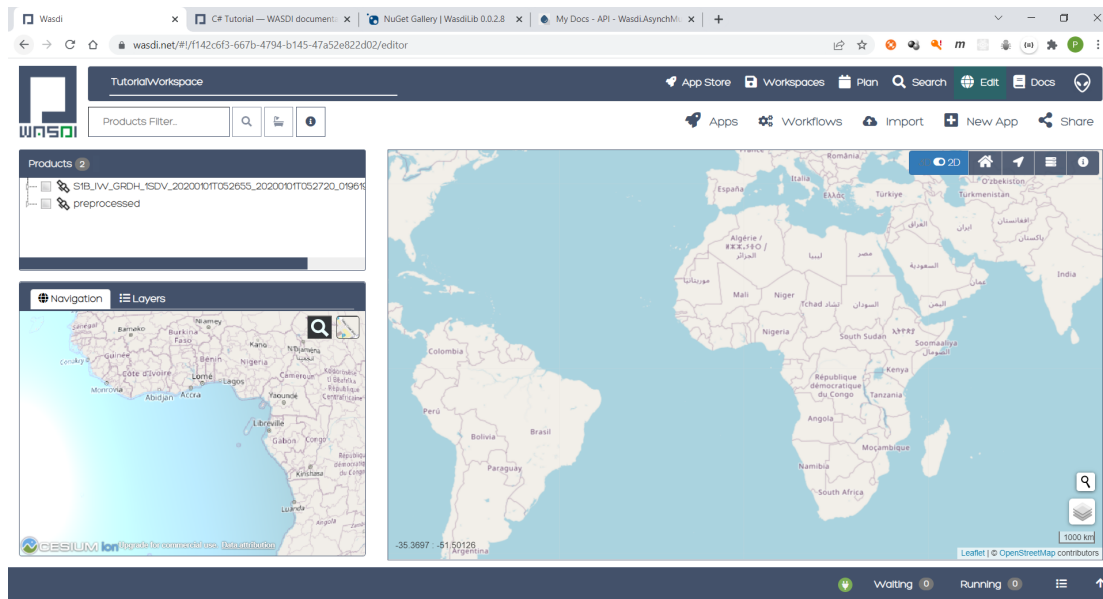


Operation	Name	User	Size	Created	Started	Progress	Duration
WORKFLOW	SIBJW_GRDH_ISDV_2020010T052655_2020010T05272...	petru.petrescu@wasdi.cloud	16 GB	2022-02-14 17:26:23 Z	2022-02-14 17:58:47 Z	60%	00:01:37
FETCH - AUTO	SIBJW_GRDH_ISDV_2020010T052655_2020010T05272...	petru.petrescu@wasdi.cloud	16 GB	2022-02-14 17:26:00 Z	2022-02-14 17:26:05 Z	100%	00:00:12
APP	tutorialseesharpapp	petru.petrescu@wasdi.cloud		2022-02-14 17:25:46 Z	2022-02-14 17:25:51 Z	0%	00:34:33
WORKFLOW	SIBJW_GRDH_ISDV_2020010T052655_2020010T05272...	petru.petrescu@wasdi.cloud	16 GB	2022-02-14 16:29:50 Z	2022-02-14 17:02:36 Z	100%	00:01:16
FETCH - AUTO	SIBJW_GRDH_ISDV_2020010T052655_2020010T05272...	petru.petrescu@wasdi.cloud	0 B	2022-02-14 16:29:18 Z	2022-02-14 16:29:23 Z	100%	00:00:20

As soon as the execution is completed, a message is shown to the user.

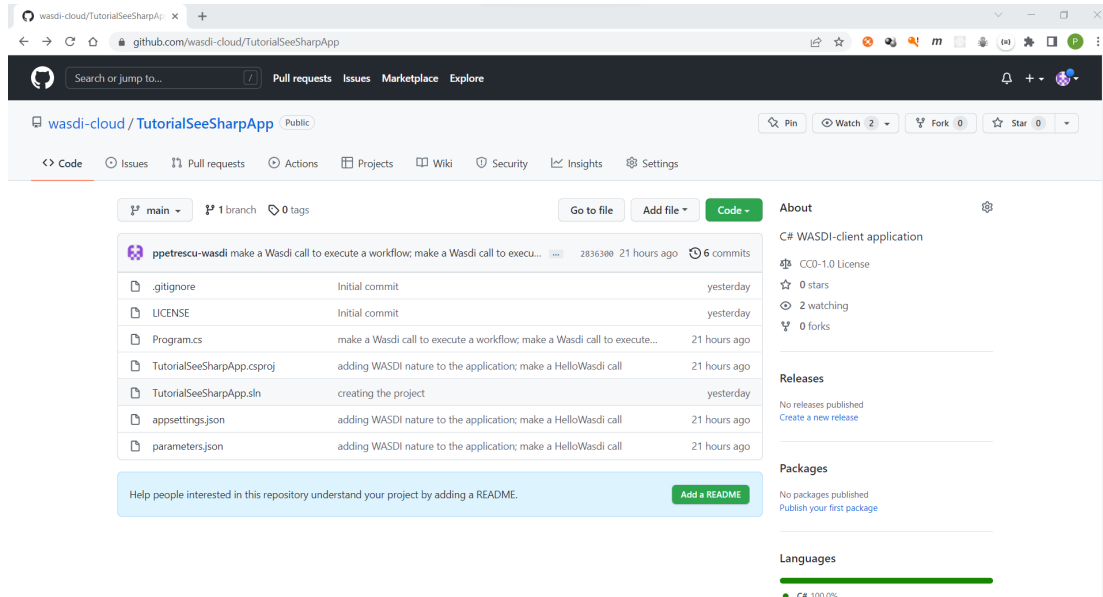


Minimizing the logging panel, the downloaded products become visible on the workspace.



The GitHub repository

The code showed in this tutorial can also be found on the dedicated public GitHub repository: <https://github.com/wasdi-cloud/TutorialSeeSharpApp>.



You can clone the project or download the code as a ZIP archive. The [commits](#) page highlights the steps of this tutorial.

The end

This is the end of the tutorial. Please try to use the WasdiLib to build interesting and powerful applications. More information about the available operations can be found on the [library reference](#) page.

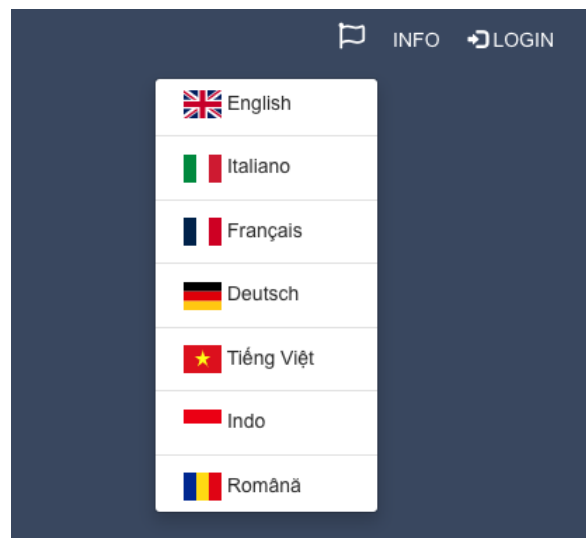
4.9 Site Map

4.9.1 Login Page

To begin a session on WASDI, you must first log on to <https://www.wasdi.net/>. Here you will be welcomed with our homepage.



In the top right-hand corner there is a toolbar.



This toolbar contains three options:

- The flag icon can be used to set the site language. The options are: English, Italian, French, German, Vietnamese, Indonesian, and Romanian.

- The Info button will re-route you to <https://www.wasdi.cloud/> by opening a new tab. This is WASDI's informational site. Here you will find information about the company and platform, links to our social media pages, and other useful resources.
- The final button is a login button. When clicked it will open a login card in the middle of the page.

Logging In

The image shows a login card with the following elements:

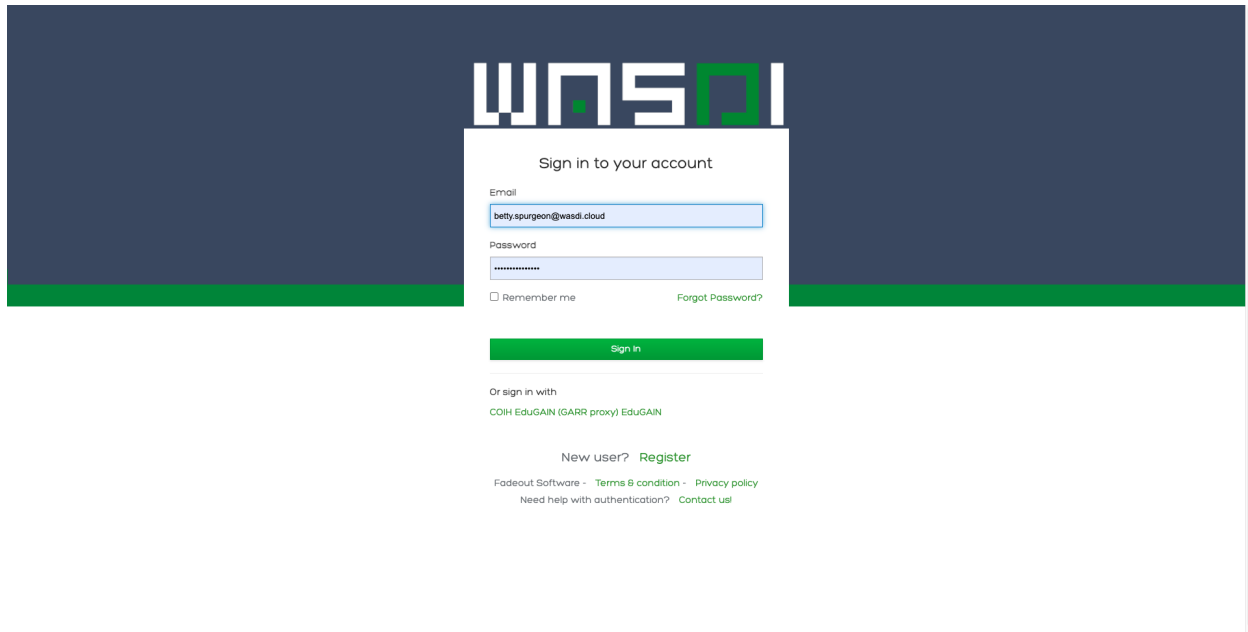
- 1.** Email input field containing `betty.spurgeon@wasdi.cloud` and a password input field with masked characters.
- ☐ Remember me
- 2.** [Forgot your password?](#)
- 3.** [New User? Register here!](#)
- 4.** [Sign in with WASDI Login 2.0](#)

This login card provides you with four options:

1. Log into an existing account.
2. Reset a lost or forgotten password.
3. Register with WASDI and create a new account.
4. Login with WASDI's new log in system.

To Login, simply enter your account credentials (email and password) to the original login card or WASDI's Login 2.0 which can be selected by clicking the bottom-most button from the homepage's login card.

If you decide to use WASDI's Login 2.0 system, you will be redirected to this page:



Here you will find most of the same options as WASDI's original login page, but with the option to sign in with COIH, EduGAIN(GARR proxy), or EduGAIN as well.

You can use the same credentials (email and password) here as on the original login page.

4.10 How to create a User Interface (UI)

4.10.1 Introduction

Note: In this tutorial you can click on all images to see them at full resolution.

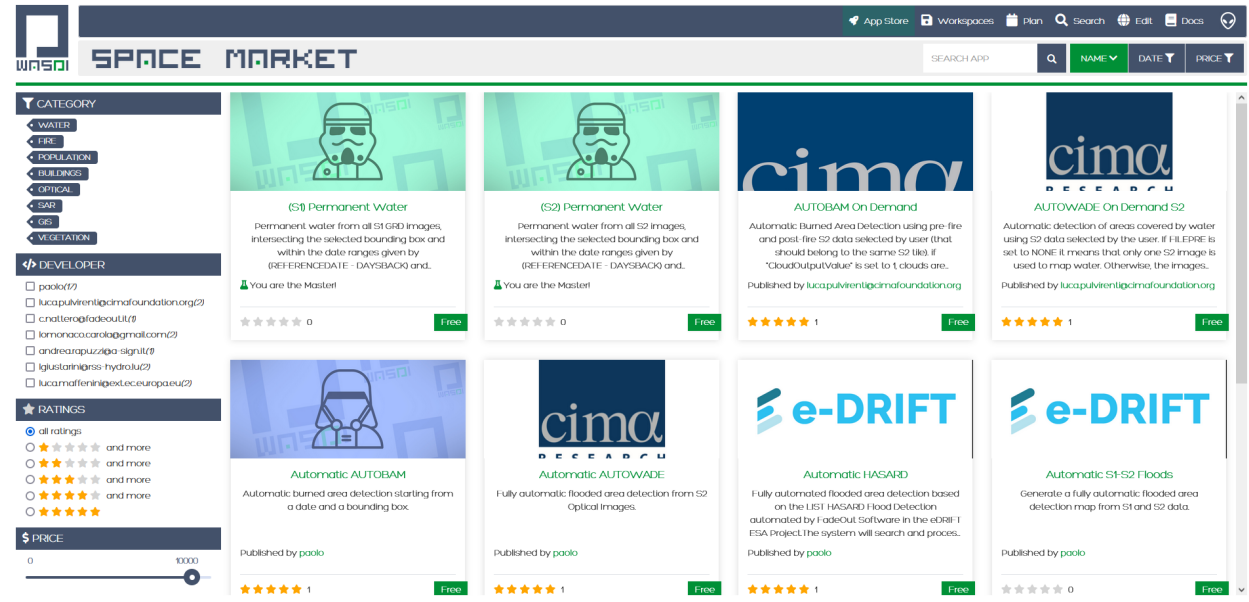
Use back command on your browser to get back to the documentation.

Here we introduce how to create a User Interface (UI) for an app that has already been deployed and made visible on the marketplace.

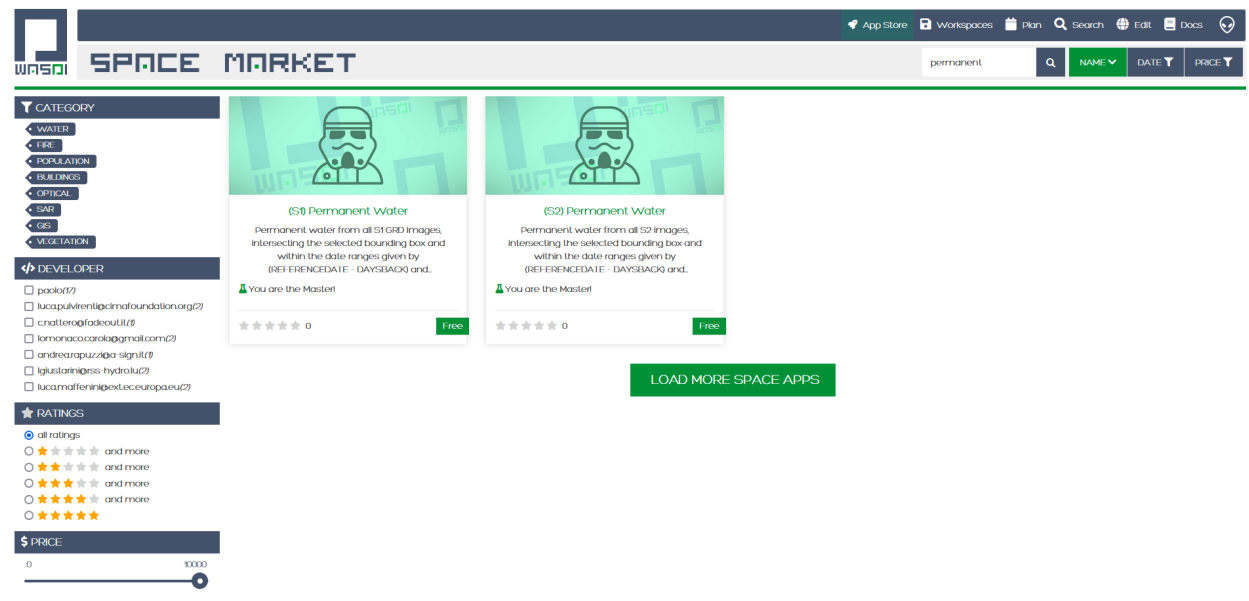
The UI represents a user-friendly way to enter the parameters that otherwise should be defined in the params.json file.

Indeed, the UI maps all parameters of the params.json file to an interface where the user can enter the parameter values, using different pieces of the interface itself that describe what parameter is expected and guide the user in the selection.

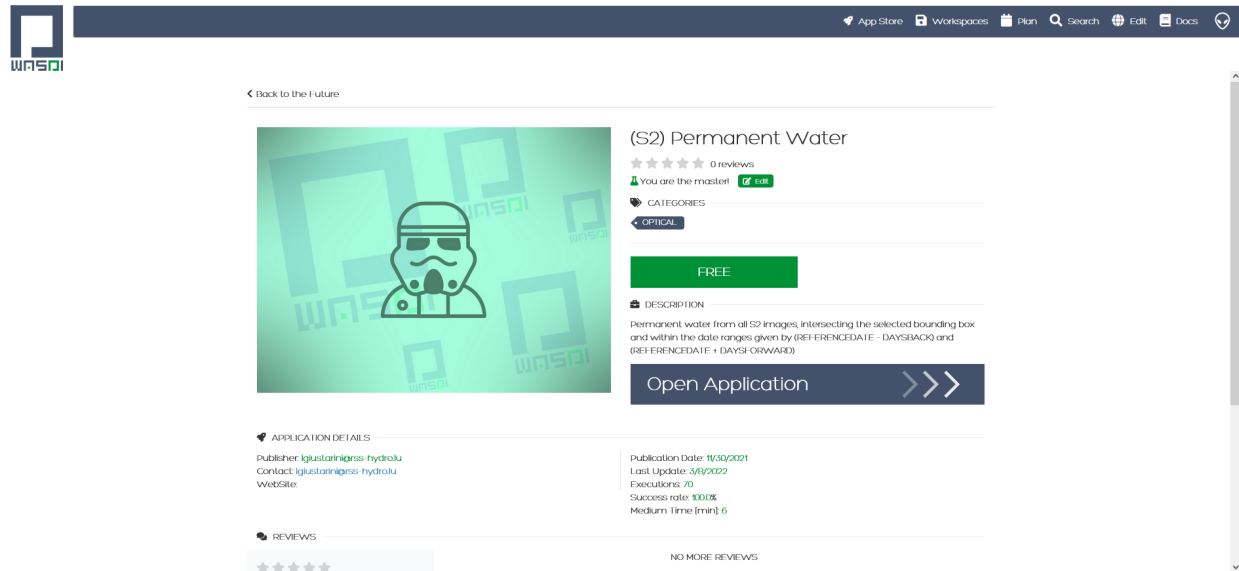
The marketplace displays all visible apps:



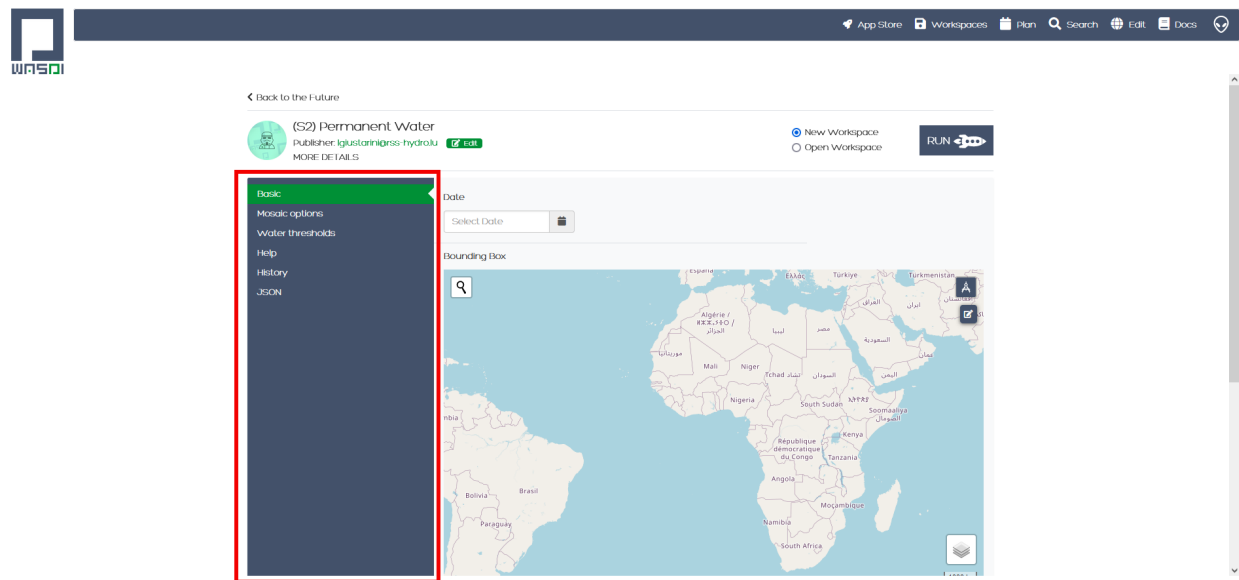
Selecting for example, the app named Automatic Permanent Water (S2):



After clicking on it to select this app:



And after opening the app itself, we are presented with the following:



Each of the elements listed to the left represent a tab.

In this case there are 6 tabs:

- Basic
- Mosaic
- Options
- Water thresholds
- Help
- History
- JSON.

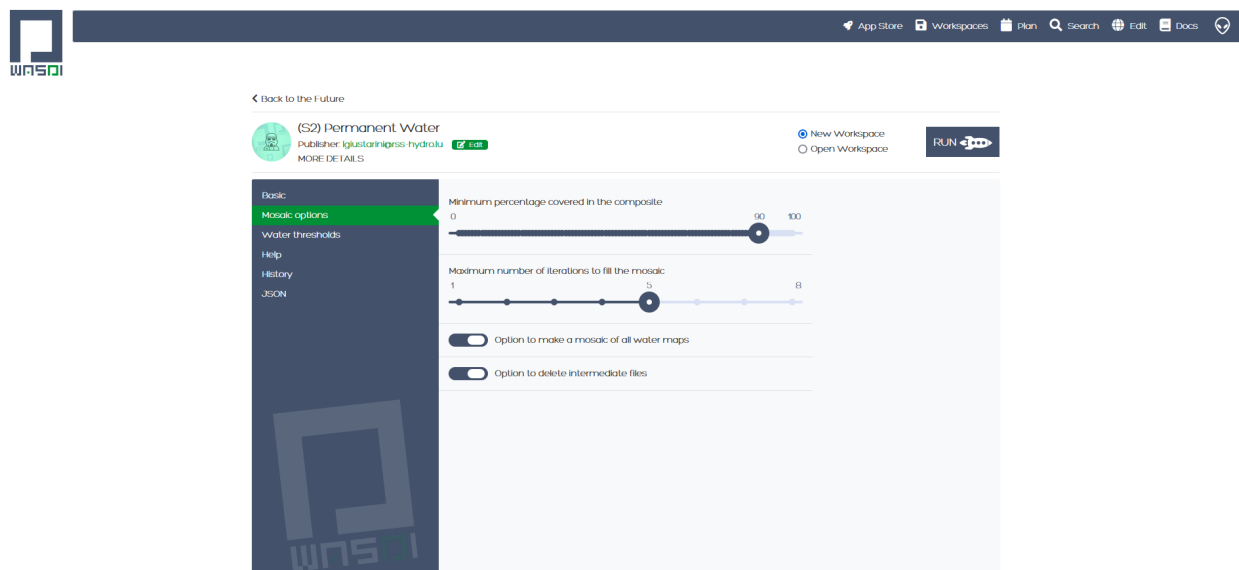
The last 3 tabs, i.e. *Help*, *History* and *JSON* are always created by default by WASDI, for each application shown in the Marketplace.

The figure above shows the controls of the tab named Basic (highlighted in green): in the figure we can see the first 2 controls of this tab, i.e. Date and Bounding Box.

A *control* is a portion of UI to enter a certain parameter. The parameters are the same as defined by the app developer in the `params.json` file.

In our case, the first parameter is a date and its corresponding control is a calendar, while the second parameter is a bounding box and its corresponding control is a map with options to define the bounding box itself.

After clicking on a different tab, for instance Mosaic options, the controls of this specific tab are displayed. In the figure below we can see the 4 controls of this tab, i.e. Minimum percentage covered in the composite, Maximum number of iterations to fill the mosaic, Option to make a mosaic of all water maps, Option to delete intermediate files. In this case, the parameter corresponding to the slider is a range of integers (for the first 2 controls in this tab), while the parameter corresponding to the on/off button is a Boolean variable (for the last 2 controls in this tab).



In general, the following controls are available for different types of parameters:

- *Textbox* to insert a text
- *Calendar* to insert a date
- *Map* to insert an area of interest
- *Slider* to insert an integer number
- *Numeric field* to insert a floating number
- *Switch* to insert a Boolean value
- *Dropdown box* to select a value from a pre-defined set of values
- *Product Dropdown* to let the user select an image in a workspace

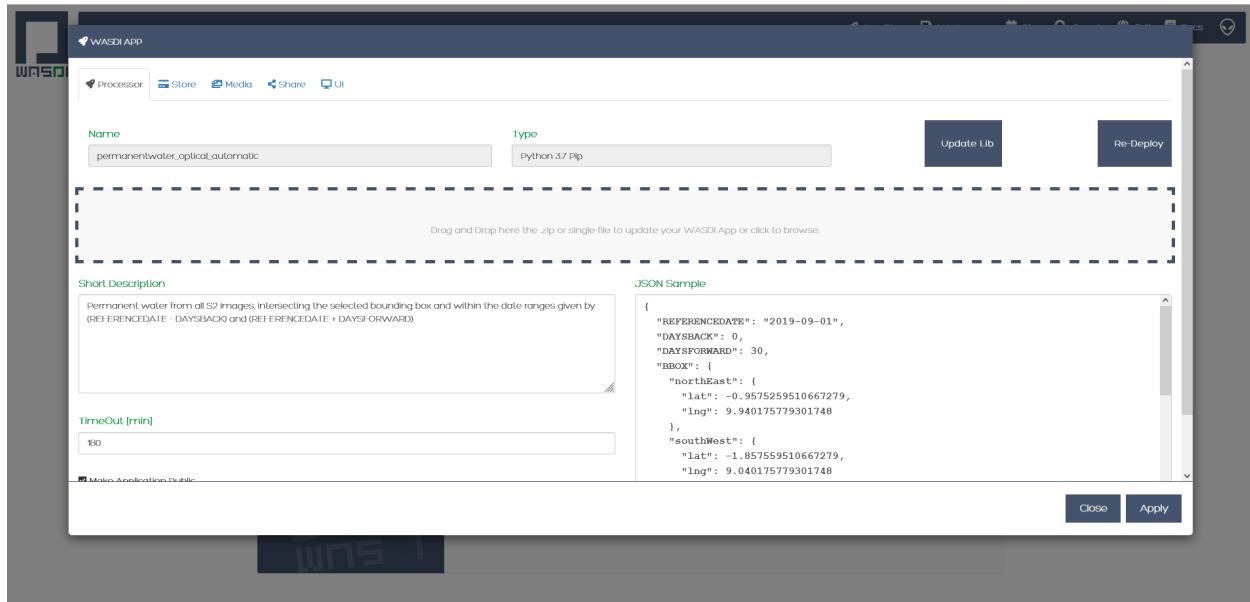
For each control, the developer can set:

- The name associated parameter
- The default value
- If it is mandatory or not

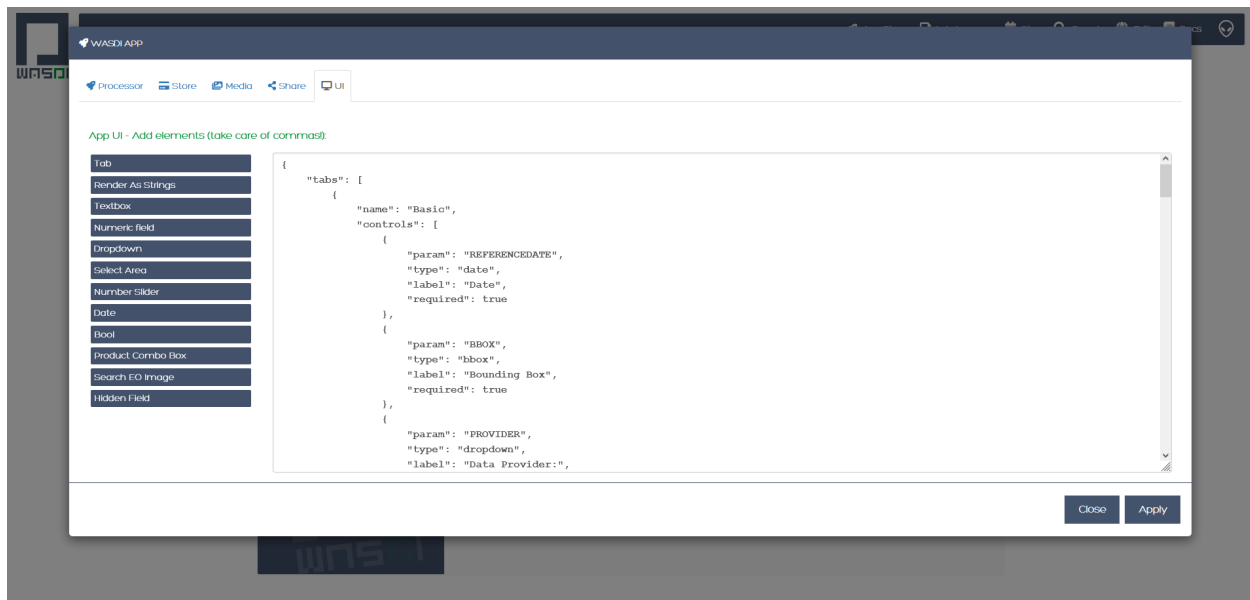
For each type of control, there may be additional options like a min and max value (for a slider), or the max or min area (for a bounding box).

To build the UI, WASDI needs a JSON that describes the number of tabs, their names as they should appear in the UI, the order of the tabs, the controls to be included in each tab, the order of the controls in the tab and the type of control to be attributed to each parameter.

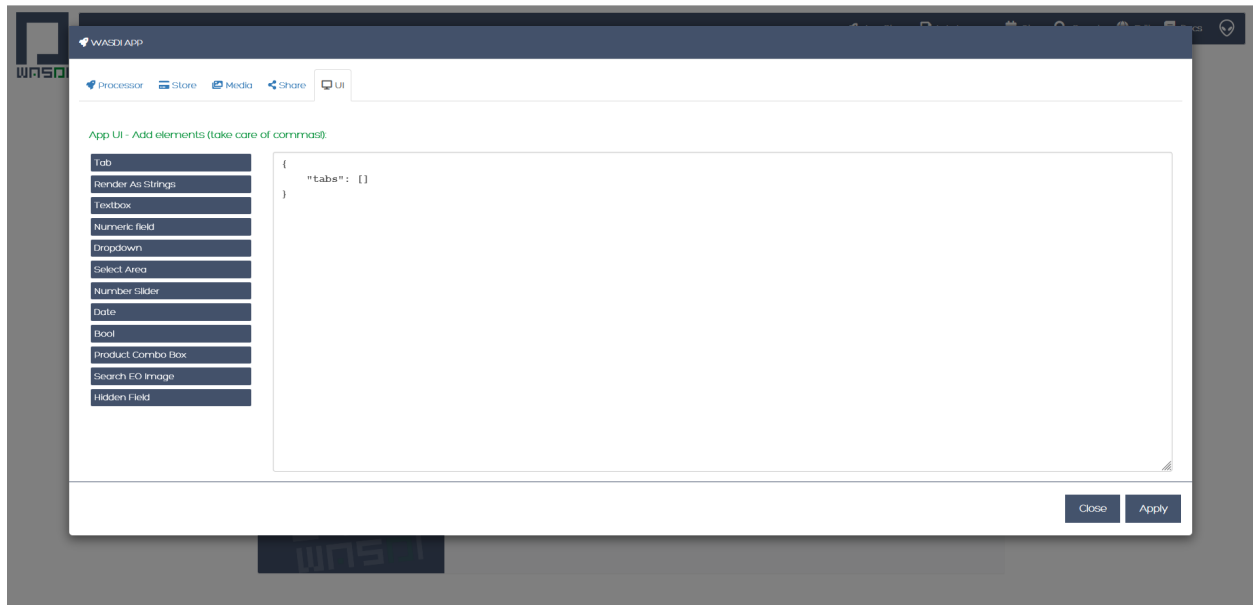
To build the UI, the developer accesses the application edit window:



and goes to the UI section:

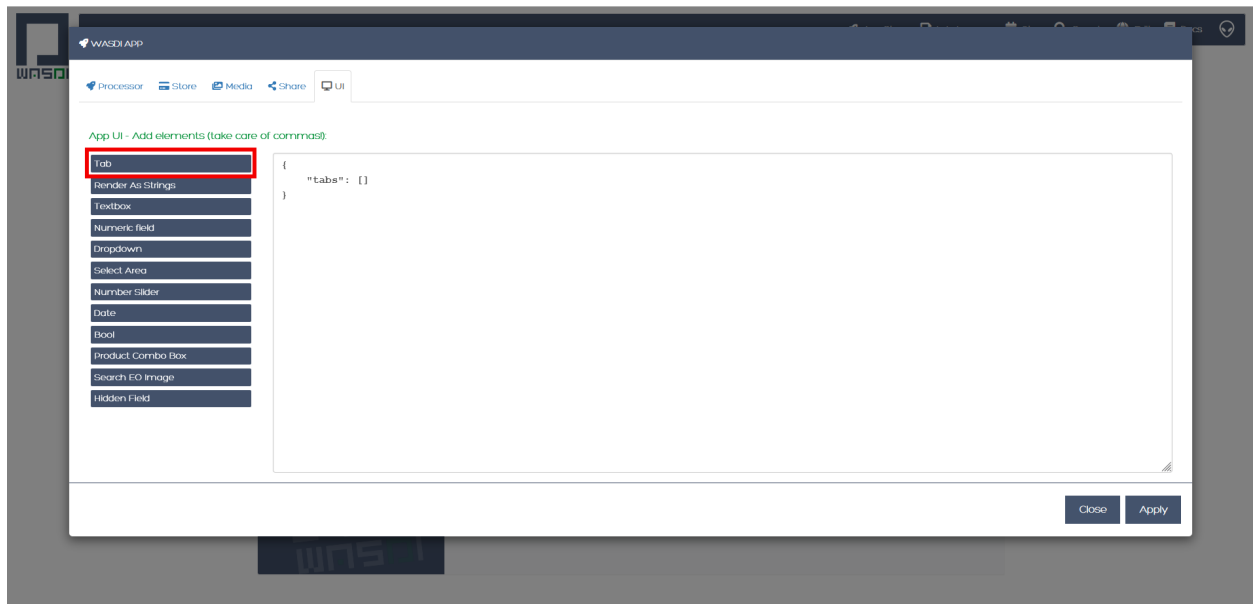


The UI editor is a text editor where the developer can edit the JSON describing the UI. The first time you open the UI text editor, it will look like this:

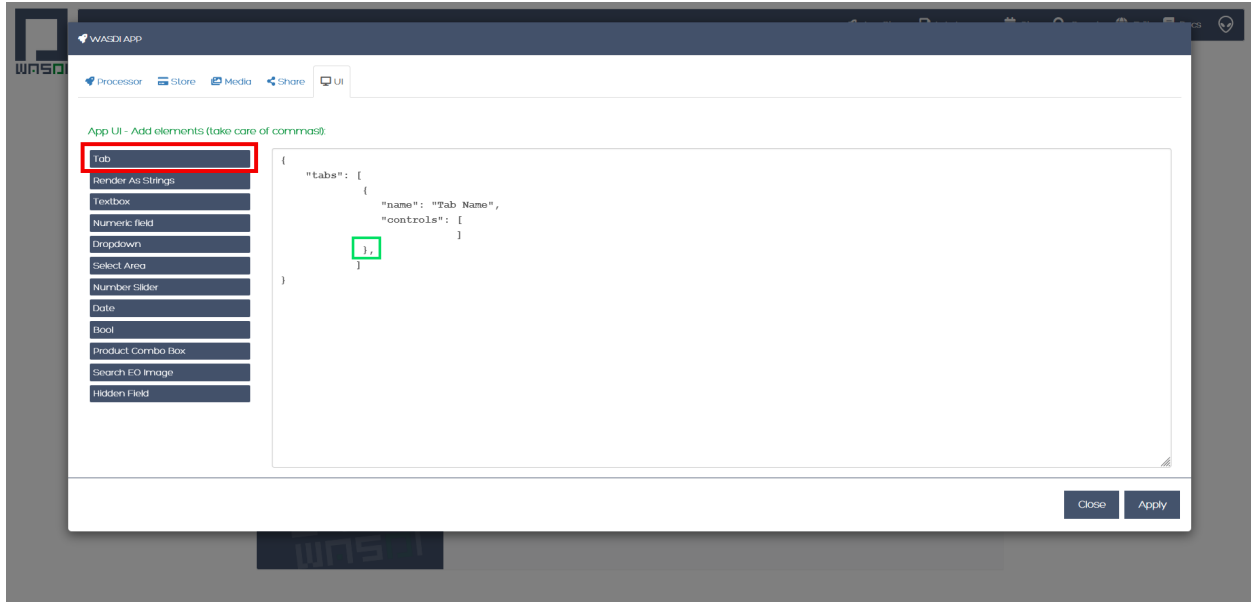


tabs indicates that what will be added between the squared brackets is going to become one or more tabs.

To add one tab, first set the cursor between the squared brackets and then click on Tab:



After clicking on Tab, this is what will appear on the right of the screen (the different brackets might be slightly differently indented):



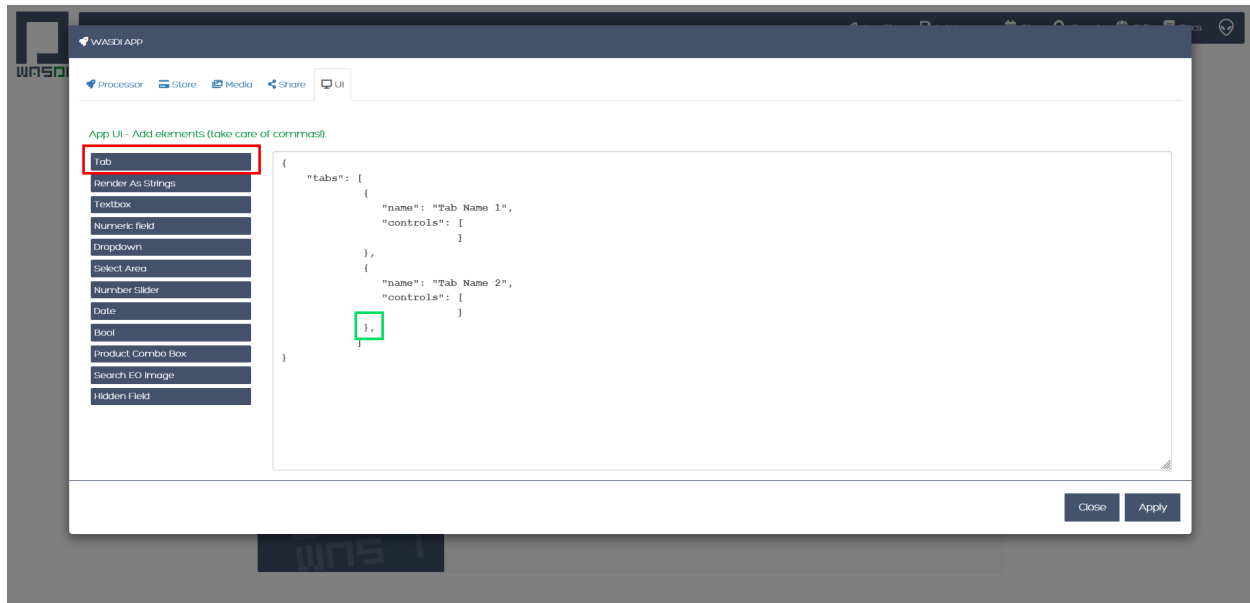
This means that we are planning to create, in the UI, a single tab named, for the moment, “Tab Name”.

This tab is going to have a certain number of controls that will need to be added between the squared brackets after “controls”:. In case you want to create a single tab, like in this example, make sure to remove the final comma.

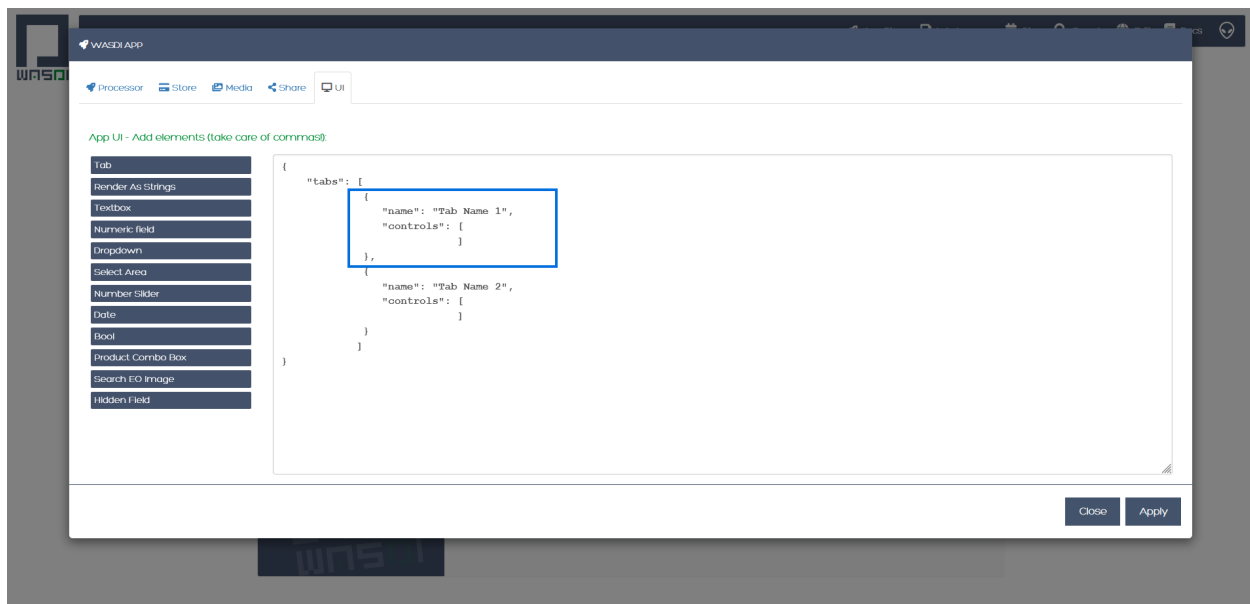
Otherwise, if you like to make more than one tab in the UI, set the cursor after the comma and click again on Tab. Let’s assume we want to make 2 tabs in the UI, each of them with their controls. Then we will have something like this:



In this case we have one tab named “Tab Name 1”, comprised between curly brackets, and a second tab named “Tab Name 2”, comprised between curly brackets. A comma separates the pairs of curly brackets defining each tab. For the second tab, I took care of removing the comma that the system automatically adds any time one click on Tab, like you can see here below:



Now we see how to build a certain tab, i.e. the portion highlighted in the blue square in the figure below.



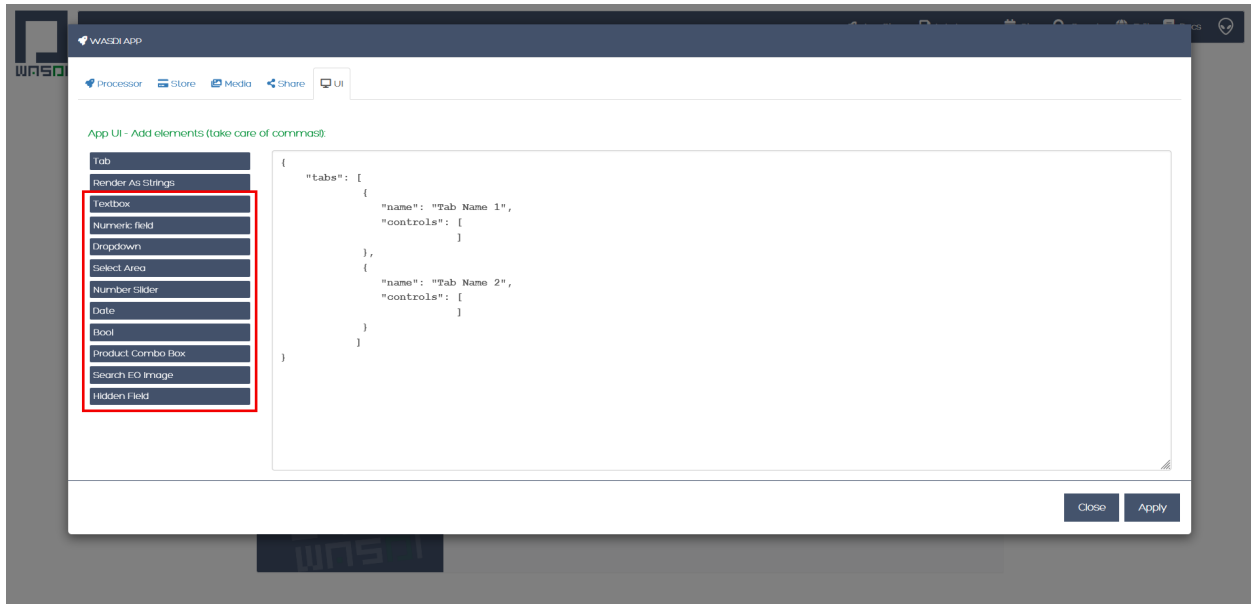
Here, “name” is the Tab Name. In the specific case “Tab Name 1” can be replaced by any strings, for example “Basic”, “Advanced”, “GIS”, ... This is the name that will be displayed in the UI. Careful: the order in which you put the tabs here is the same order in which they will be displayed in the UI.

Once the name of a tab has been chosen, we need to add its controls, between the squared brackets. Careful: the order in which you put the controls within the tab is the same order in which they will be displayed in the UI. The list of available controls is as follows:

- Textbox: to display a parameter in the form of text
- Numeric field: to display a parameter in the form of a float number
- Dropdown: to display a parameter as a pre-defined set of values in a dropdown menu
- Select Area: to display a parameter in the form of an area of interest to be selected over a map

- Number Slider: to display a parameter in the form of an integer number
- Date: to display a parameter in the form of a date to be selected from a calendar
- Bool: to display a parameter in the form of switch to insert a Boolean value
- Product Combo Box: to select from an existing workflow where the required images have been previously loaded, the image(s) to be used
- Search EO image: a mini-search image embedded in the store
- Hidden field: for parameters that the developer does not want to be exposed to the user

All these buttons are listed on the left of the screen.



Note: Every time you click one of these buttons, WASDI will add the relative JSON “snippet” in the position of the cursor. By default, WASDI adds also the comma at the end of the snippet. Remember to delete it if it is the last element of a list to have a correct JSON.

To add controls to a given tab, first set the cursor between the square brackets after

```
"controls": [ ]
```

Then, clicking on the elements on the left of the screen, you can add one or more controls. The following paragraph walks you through each different control and shows how to set its properties and how it will eventually look like in the UI.

4.10.2 Controls Shared Properties

Each control is defined by one or more properties. All controls have at least 2 properties:

- “param”: Name of the parameters. This links the parameter as displayed in the UI to the parameter defined in the params.json file
- “type”: type of the interface block that will be added. In other words, it is one of the types listed above as available controls.

Other than these 2, other common properties are:

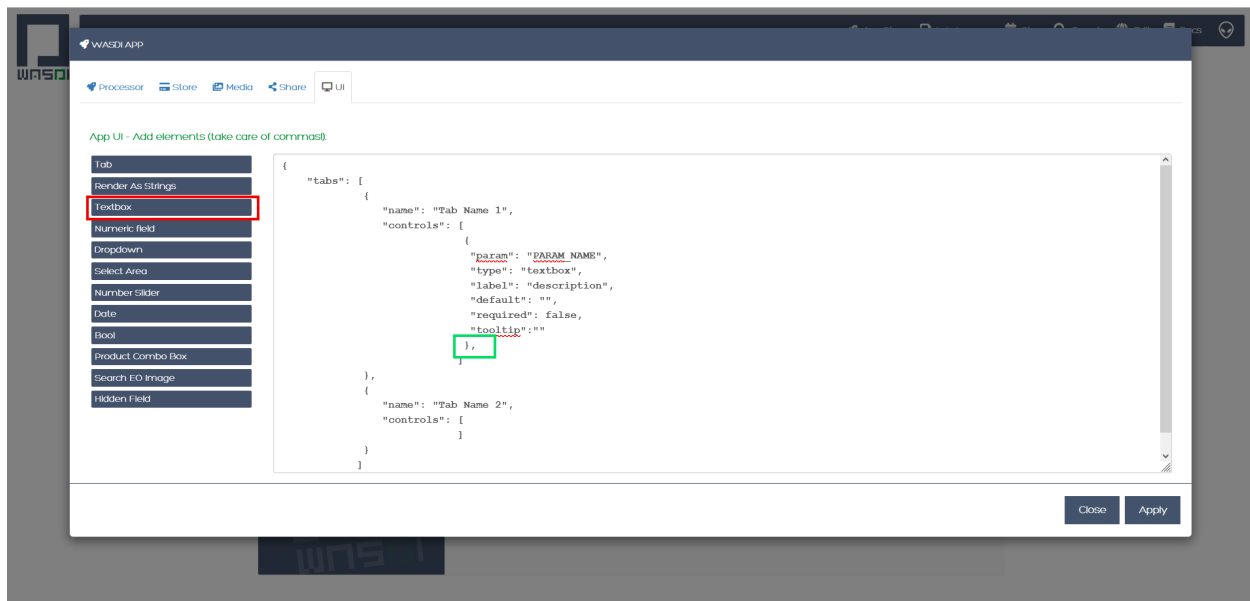
- “label”: used to define the name the parameter in the UI.
- “default”: used to set the default value of this parameter.
- “required”: used to define is the parameter is mandatory or not.
- “tooltip”: used as a little help about the parameter to show to the user when he overs the mouse on the control

Here it is an example with a textbox:

```
{
  "param": "PARAM_NAME",
  "type": "textbox",
  "label": "description",
  "default": "",
  "required": false,
  "tooltip": ""
},
```

4.10.3 Textbox

To add to “Tab Name 1” a control in the form of a text box, first set the cursor between the square brackets after “controls” and then click on the button Textbox (to the left of the screen).



Note: Careful: in case you have only one control in this specific tab, or this is the last control of the tab, make sure to remove the trailing comma!

```
{
  "param": "PARAM_NAME",
  "type": "textbox",
  "label": "description",
  "default": "",
  "required": false,
  "tooltip": ""
}
```

The property “param” is used to identify the parameter to be used here. “PARAM_NAME” has to be exactly the same as in the params.json file.

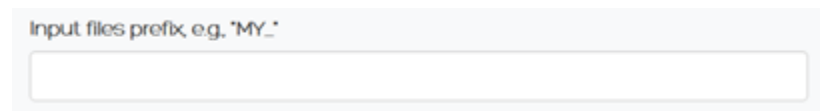
Do not modify the property “type”: “textbox”.

The property “label” is used to define the name the parameter in the UI. In this case, it is prefilled with the text “description”. Please change it to the name of your parameter as you would like to see it displayed in the UI.

The property “default” is used to set the default value of this parameter, in case the user does not know how to set it or does not want to change it.

The property “required” is used to define if the parameter is mandatory or not. It can be either true or false (careful: no “”).

This is an example of how a Textbox control appears in the UI:

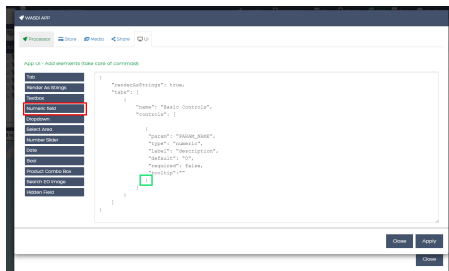


In this case, here is how the properties of this control were set:

```
{
  "label": "Input files prefix",
  "default": "",
  "required": true
}
```

4.10.4 Numeric field

To add to “Tab Name 1” a control in the form of a float number, first set the cursor between the square brackets after “controls” and then click on the button Numeric field (to the left of the screen).



Note: Careful: in case you have only one control in this specific tab, or this is the last control of the tab, make sure to remove the trailing comma!

```
{
  "param": "PARAM_NAME",
  "type": "numeric",
  "label": "description",
  "default": "0",
  "min": 0,
  "max": 100,
  "required": false,
  "tooltip": ""
}
```

The property “param” is used to identify the parameter to be used here. “PARAM_NAME” has to be exactly the same as in the params.json file.

Do not modify the property “type”: “numeric”.

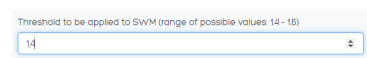
The property “label” is used to define the name the parameter in the UI. In this case, it is prefilled with the text “description”. Please change it to the name of your parameter as you would like to see it displayed in the UI.

The property “default” is used to set the default value of this parameter, in case the user does not know how to set it or does not want to change it. Set it to the numeric float value that you want as default.

The property “required” is used to define is the parameter is mandatory or not. It can be either true or false (careful: no “”).

Again, careful with the trailing comma! If you add one more control to this specific tab, click after the final comma, otherwise take case of removing the final comma.

This is an example of how a Numeric field control appears in the UI:



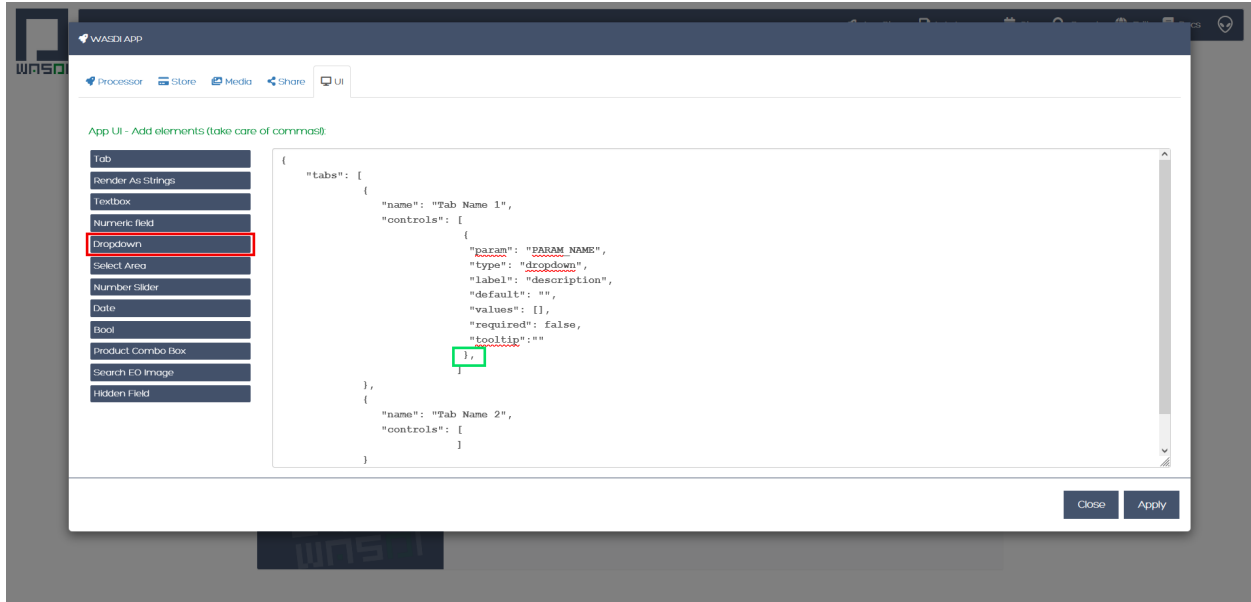
Threshold to be applied to SWM (range of possible values: 1.4 - 1.6)

In this case, here is how the properties of this control were set:

```
{
  "label": "Threshold to be applied to SWM (range of possible values: 1.4 - 1.6)",
  "default": 1.4,
  "required": true
}
```

4.10.5 Dropdown

To add to “Tab Name 1” a control in the form of a drop down menu with several options to choose from, first set the cursor between the square brackets after “controls” and then click on the button Dropdown (to the left of the screen).



Note: Careful: in case you have only one control in this specific tab, or this is the last control of the tab, make sure to remove the trailing comma!

```
{
  "param": "PARAM_NAME",
  "type": "dropdown",
  "label": "description",
  "default": "",
  "values": [],
  "required": false,
  "tooltip": ""
}
```

The property “param” is used to identify the parameter to be used here. “PARAM_NAME” has to be exactly the same as in the params.json file.

Do not modify the property “type”: “dropdown”.

The property “label” is used to define the name the parameter in the UI. In this case, it is prefilled with the text “description”. Please change it to the name of your parameter as you would like to see it displayed in the UI.

The property “default” is used to set the default value of this parameter, in case the user does not know how to set it or does not want to change it. Set it to the value that you want as default (one of those listed in “values” in the following line).

For the property “values”, within the squared brackets [], add a list of strings, that represent the values to appear in the dropdown menu. For example, it could be: “values”: [“ONDA”, “EODC”, “CREODIAS”]

This is an example of how a Dropdown menu control appears in the UI:

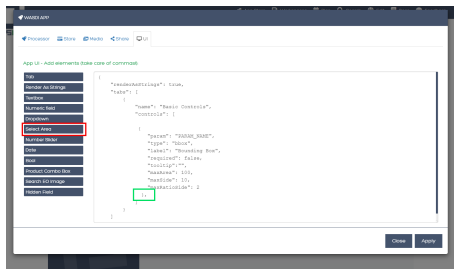


In this case, here is how the properties of this control were set:

```
{
  "label": "Data Provider:",
  "default": "ONDA",
  "values": [
    "ONDA",
    "EODC",
    "SOBLOO",
    "CREODIAS"
  ]
}
```

4.10.6 Select Area

To add to “Tab Name 1” a control in the form of a bounding box, that the user can either draw on the displayed map or that can be inputted as the 4 values of North, South, East, West within a pop up window in the UI, first set the cursor between the square brackets after “controls” and then click on the button Select Area (to the left of the screen).



Note: Careful: in case you have only one control in this specific tab, or this is the last control of the tab, make sure to remove the trailing comma!

```
{
  "param": "PARAM_NAME",
  "type": "bbox",
  "label": "Bounding Box",
  "required": false,
  "tooltip": "",
  "maxArea": 0,
  "maxSide": 0,
  "maxRatioSide": 0
}
```

The property “param” is used to identify the parameter to be used here. “PARAM_NAME” has to be exactly the same as in the params.json file.

Do not modify the property “type”: “bbox”.

The property “label” is used to define the name the parameter in the UI. In this case, it is prefilled with the text “Bounding Box”. If you want, please change it to the name of your parameter as you would like to see it displayed in the UI.

The property “required” is used to define if the parameter is mandatory or not. It can be either true or false (careful: no “”).

The control also allows to set some **limits** to the area selected. In case one or more of these constraints are violated, the user will receive a specific feedback and the application cannot be launched.

The limitations can be imposed upon:

- Max area in square kilometre
- Max side in kilometre
- Maximum ratio between the sides of the selected area computed as the greater side over the smaller one. (e.g. a bounding box of 2 kilometre by 1 kilometer will have the ratio equals to 2)

The last constraint can be used to avoid that application users, by mistake, set a bounding box very thin but also very large: imagine for instance 1 meter per 1000 kilometers. This setup will require the load of several tiles and will slow down the performances in general.

If maximum ratio is set to a reasonable value it can help users to avoid such errors.

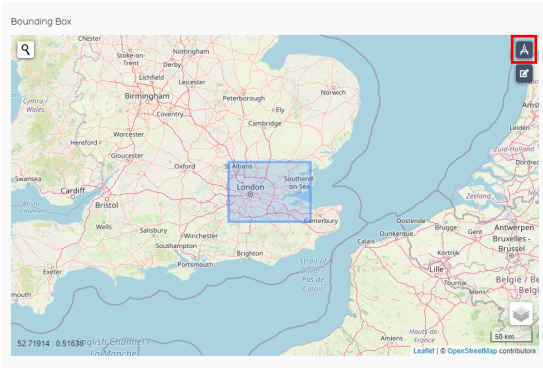
This is an example of how a Select Area control appears in the UI:



In this case, here is how the properties of this control were set:

```
{
  "param": "PARAM_NAME",
  "type": "bbox",
  "label": "Bounding Box",
  "required": false,
  "tooltip": "",
  "maxArea": 10000,
  "maxSide": 1500,
  "maxRatioSide": 10
}
```

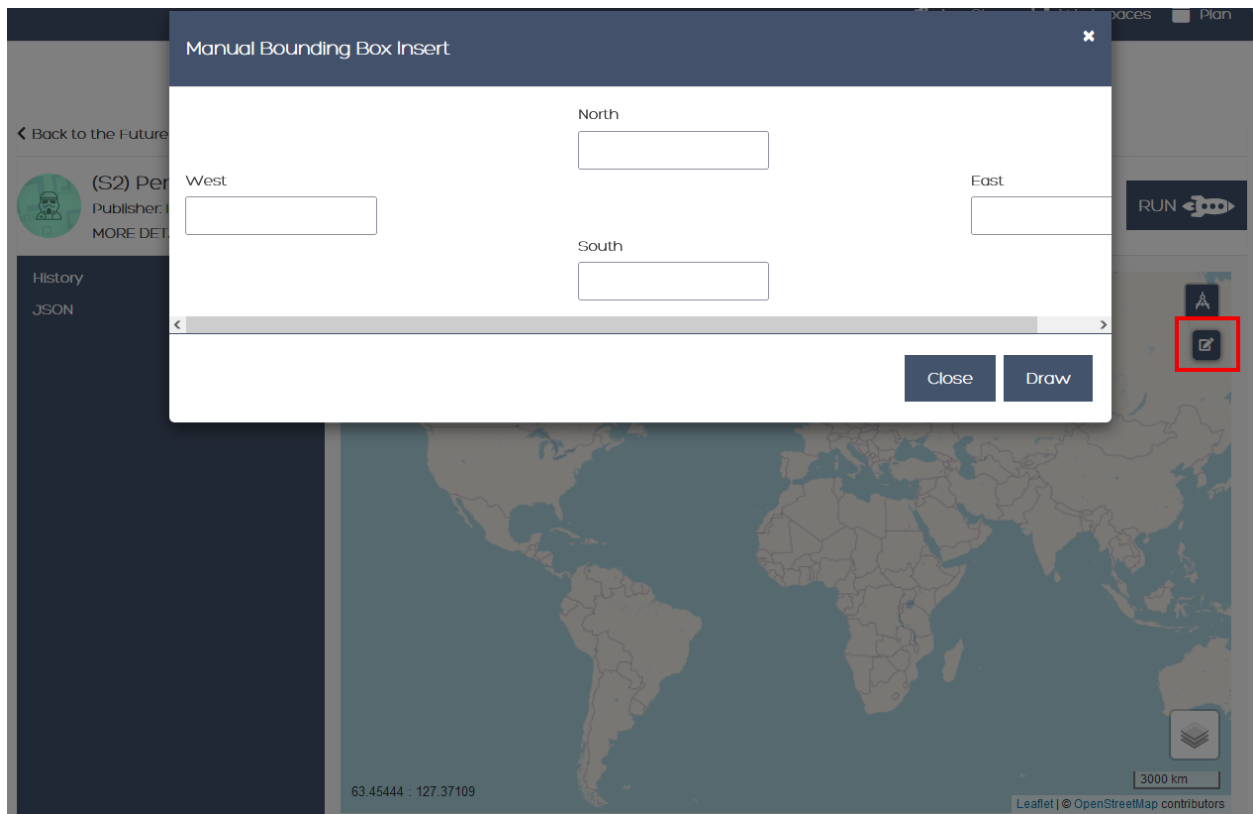
The option highlighted in the figure below is used to manually draw a rectangle:



If the area selected surpass the limits, a dedicated error message is shown and its not possible, for the user, to launch the application.

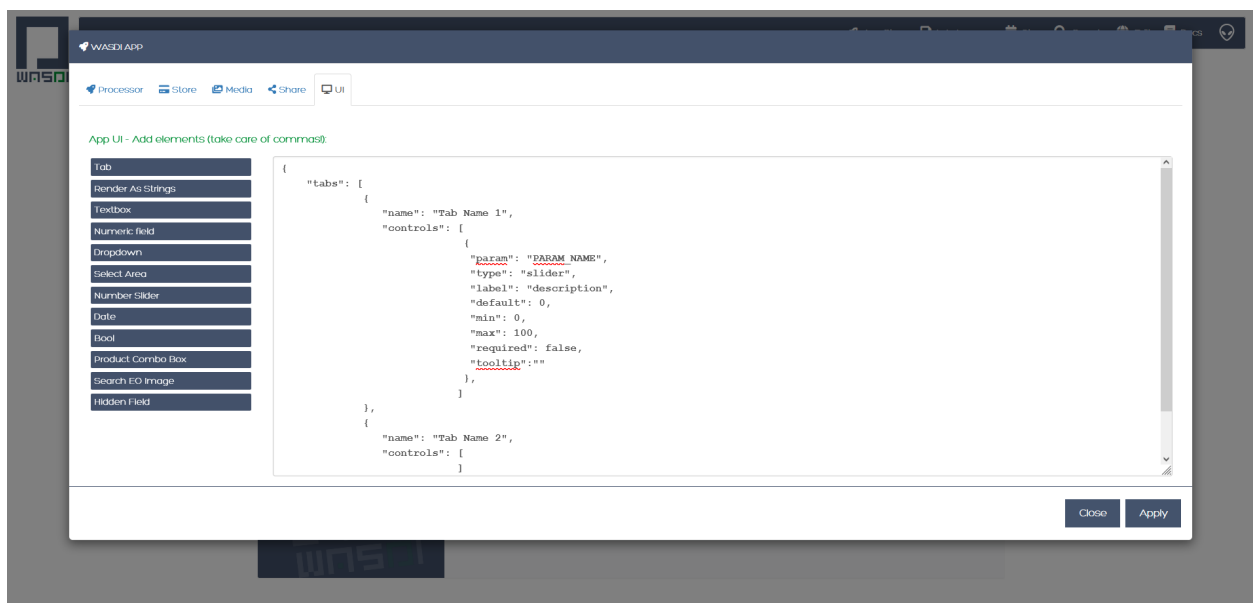


The other option, highlighted in the figure below, allow the user to manually enter the values of the bounding box:



4.10.7 Number Slider

To add to “Tab Name 1” a control in the form of an integer number within a range of values, first set the cursor between the square brackets after “controls” and then click on the button Number Slider (to the left of the screen).



Note: Careful: in case you have only one control in this specific tab, or this is the last control of the tab, make sure to

remove the trailing comma!

```
{
  "param": "PARAM_NAME",
  "type": "slider",
  "label": "description",
  "default": 0,
  "min": 0,
  "max": 100,
  "required": false,
  "tooltip": ""
},
```

The property “param” is used to identify the parameter to be used here. “PARAM_NAME” has to be exactly the same as in the params.json file.

Do not modify the property “type”: “slider”.

The property “label” is used to define the name the parameter in the UI. In this case, it is prefilled with the text “description”. Please change it to the name of your parameter as you would like to see it displayed in the UI.

The property “default” is used to set the default value of this parameter, in case the user does not know how to set it or does not want to change it. Set it to the numeric integer value that you want as default.

The property “min” is used to set the minimum (integer) acceptable value of this parameter. Set it to the numeric integer value that you want as minimum.

The property “max” is used to set the maximum (integer) acceptable value of this parameter. Set it to the numeric integer value that you want as maximum.

The property “required” is used to define is the parameter is mandatory or not. It can be either true or false (careful: no “”).

This is an example of how a Number Slider control appears in the UI:

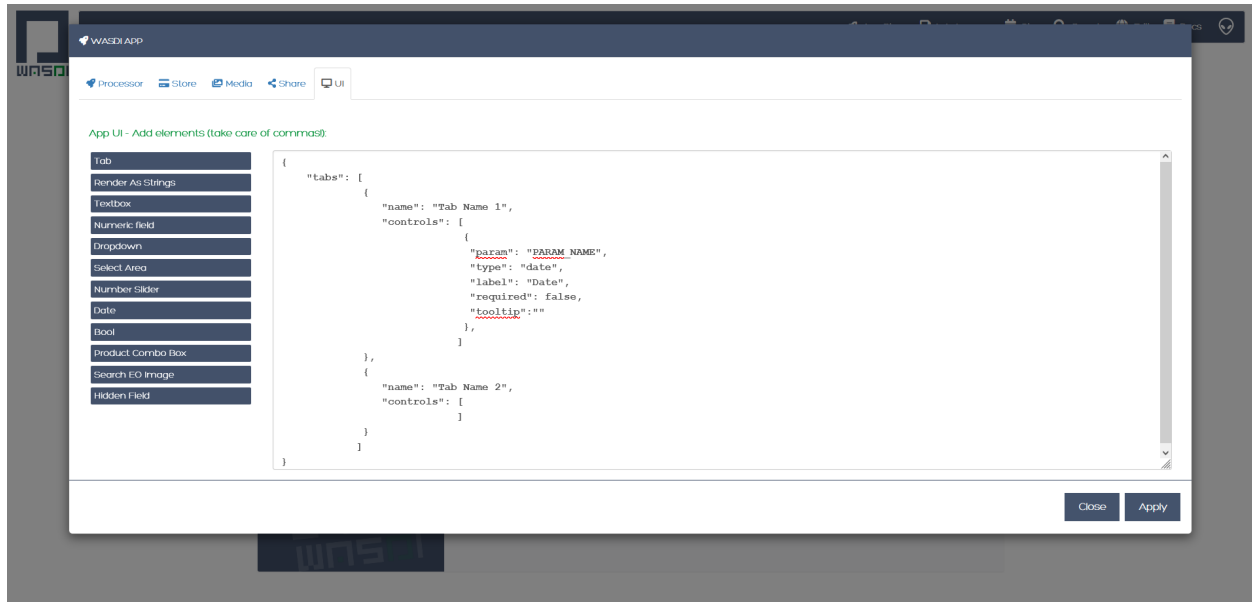


In this case, here is how the properties of this control were set:

```
{
  "label": "Days to search in the past",
  "default": 10,
  "min": 5,
  "max": 20,
  "required": true
}
```

4.10.8 Date

To add to “Tab Name 1” a control in the form of a date, first set the cursor between the square brackets after “controls” and then click on the button Date (to the left of the screen).



Note: Careful: in case you have only one control in this specific tab, or this is the last control of the tab, make sure to remove the trailing comma!

```
{
  "param": "PARAM_NAME",
  "type": "date",
  "label": "Date",
  "required": false,
  "tooltip": ""
},
```

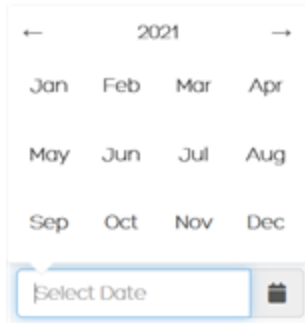
The property “param” is used to identify the parameter to be used here. “PARAM_NAME” has to be exactly the same as in the params.json file. Please change it to the name of your parameter if you want to.

Do not modify the property “type”: “date”.

The property “label” is used to define the name the parameter in the UI. In this case, it is prefilled with the text “Date”. If you want, please change it to the name of your parameter as you would like to see it displayed in the UI.

The property “required” is used to define if the parameter is mandatory or not. It can be either true or false (careful: no “”).

This is an example of how a Date control appears in the UI:

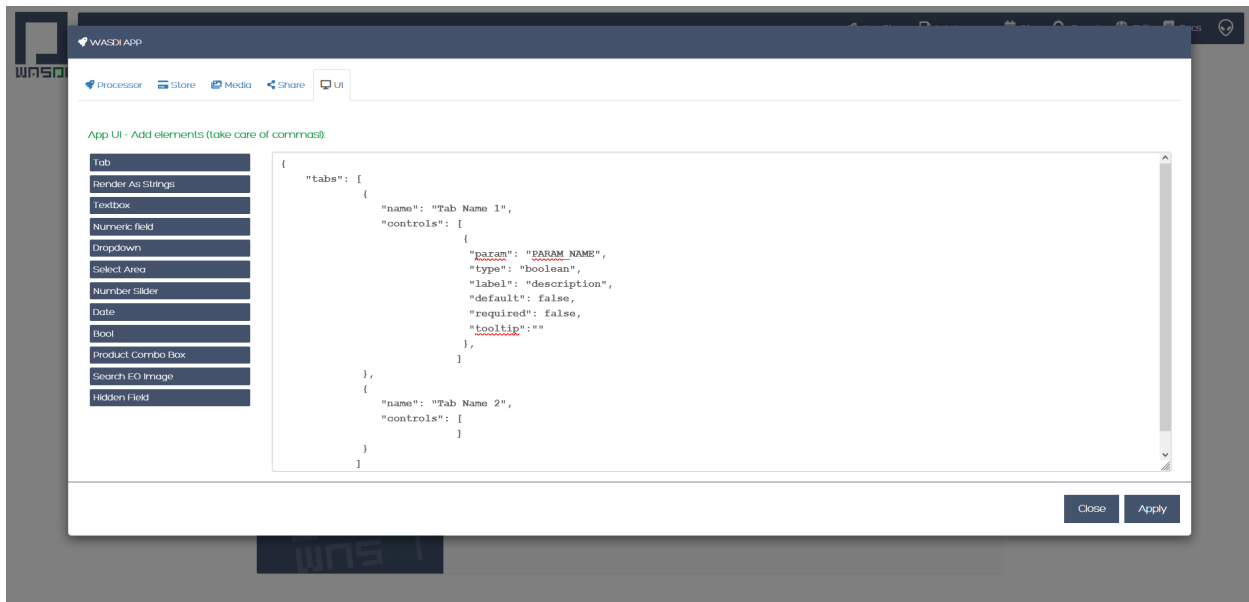


In this case, here is how the properties of this control were set:

```
{
  "label": "Date",
  "required": true
}
```

4.10.9 Bool

To add to “Tab Name 1” a control in the form of a Boolean variable, first set the cursor between the square brackets after “controls” and then click on the button Bool (to the left of the screen).



Note: Careful: in case you have only one control in this specific tab, or this is the last control of the tab, make sure to remove the trailing comma!

```
{
  "param": "PARAM_NAME",
  "type": "boolean",
  "label": "description",
  "default": false,
```

(continues on next page)

(continued from previous page)

```

"required": false,
"tooltip": ""
}

```

The property “param” is used to identify the parameter to be used here. “PARAM_NAME” has to be exactly the same as in the params.json file. Please change it to the name of your parameter if you want to.


Do not modify the property “type”: “boolean”.

The property “label” is used to define the name the parameter in the UI. In this case, it is prefilled with the text “description”. Please change it to the name of your parameter as you would like to see it displayed in the UI.

The property “default” is used to set the default value of this parameter, in case the user does not know how to set it or does not want to change it. Set it to the value that you want as default: false or true.

The property “required” is used to define is the parameter is mandatory or not. It can be either true or false (careful: no “”).

This is an example of how a Bool control appears in the UI:

 Option to delete intermediate files

In this case, here is how the properties of this control were set:

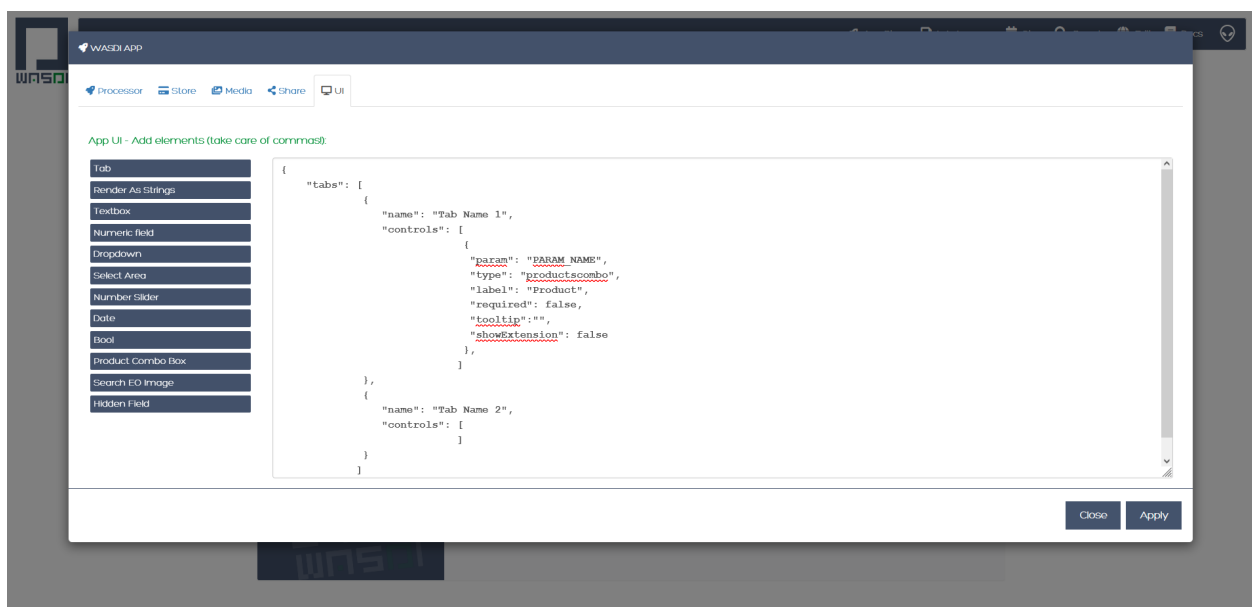
```

{
  "label": "Option to delete intermediate files",
  "default": true,
  "required": true
}

```

4.10.10 Products Combo Box

To add to “Tab Name 1” a control in the form of Product Combo Box to allow selecting a product from an existing workspace, first set the cursor between the square brackets after “controls” and then click on the button Product Combo Box (to the left of the screen).



Note: Careful: in case you have only one control in this specific tab, or this is the last control of the tab, make sure to remove the trailing comma!

```
{  
  "param": "PARAM_NAME",  
  "type": "productscombo",  
  "label": "Product",  
  "required": false,  
  "tooltip": "",  
  "showExtension": false  
},
```

The property “param” is used to identify the parameter to be used here. “PARAM_NAME” has to be exactly the same as in the params.json file. Please change it to the name of your parameter if you want to.

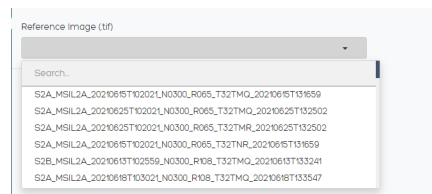
Do not modify the property “type”: “ productscombo “,.

The property “label” is used to define the name the parameter in the UI. In this case, it is prefilled with the text “Product”. Please change it to the name of your parameter as you would like to see it displayed in the UI.

The property “required” is used to define is the parameter is mandatory or not. It can be either true or false (careful: no “”).

The property “ showExtension “ is determine whether the extension of the output of the combo will be showed. It can be either false or true.

This is an example of how a Product Combo Box control appears in the UI:

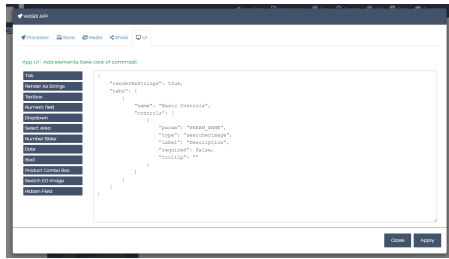


In this case, here is how the properties of this control were set:

```
{  
  "type": "productscombo",  
  "label": "Reference Image (.tif)",  
  "required": true,  
  "showExtension": false  
}
```

4.10.11 Search EO Image

To add to “Tab Name 1” a control in the form of ..., first set the cursor between the square brackets after “controls” and then click on the button Search EO Image (to the left of the screen).



Note: Careful: in case you have only one control in this specific tab, or this is the last control of the tab, make sure to remove the trailing comma!

```
{
  "param": "PARAM_NAME",
  "type": "searcheoimage",
  "label": "Description",
  "required": false,
  "tooltip": ""
}
```

The property “param” is used to identify the parameter to be used here. “PARAM_NAME” has to be exactly the same as in the params.json file. Please change it to the name of your parameter if you want to.

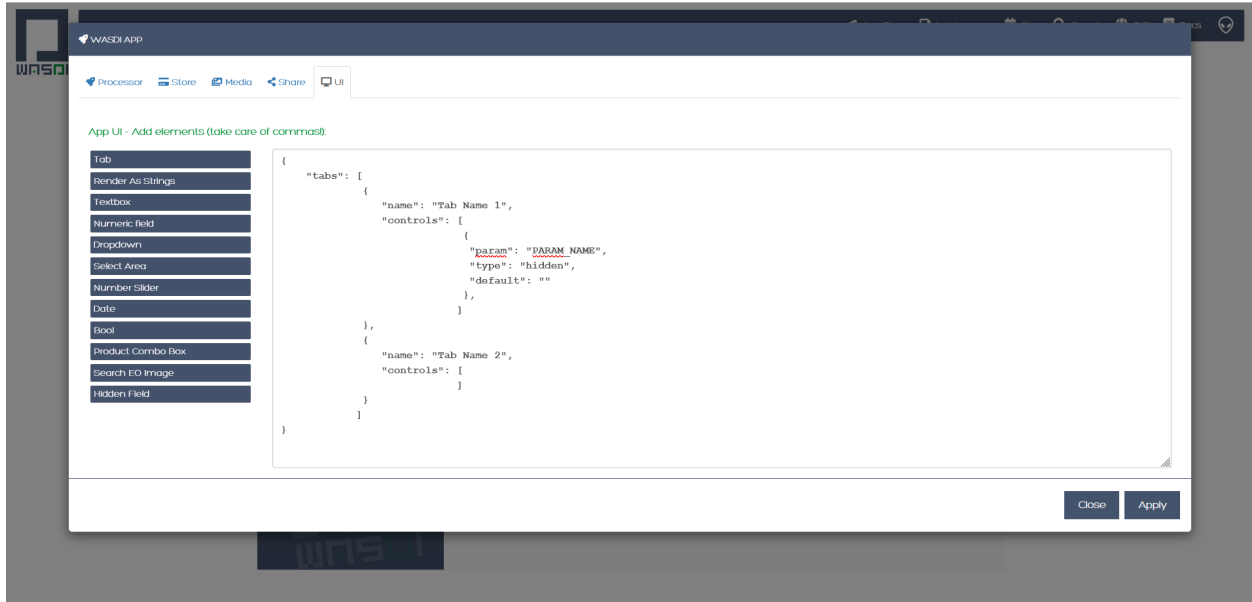
Do not modify the property “type”: “searcheoimage”.

The property “label” is used to define the name the parameter in the UI. In this case, it is prefilled with the text “Description”. Please change it to the name of your parameter as you would like to see it displayed in the UI.

The property “required” is used to define if the parameter is mandatory or not. It can be either true or false (careful: no “”).

4.10.12 Hidden Field

To add to “Tab Name 1” a control in the form of an **Hidden field**, first set the cursor between the square brackets after “controls” and then click on the button Hidden Field (to the left of the screen).



Note: Careful: in case you have only one control in this specific tab, or this is the last control of the tab, make sure to remove the trailing comma!

```
{
  "param": "PARAM_NAME",
  "type": "hidden",
  "default": ""
}
```

The property “param” is used to identify the parameter to be used here. “PARAM_NAME” has to be exactly the same as in the params.json file. Please change it to the name of your parameter if you want to.

Do not modify the property “type”: “ hidden “,.

The property “default” allows to set the actual value for this UI control.

4.10.13 List Box

To add to “Tab Name 1” a control in the form of an **List Box**, first set the cursor between the square brackets after “controls” and then click on the button List Box (to the left of the screen).

Note: Careful: in case you have only one control in this specific tab, or this is the last control of the tab, make sure to remove the trailing comma!

```
{
  "param": "PARAM_NAME",
  "type": "listbox",
  "label": "description",
  "values": [],
  "required": false,
```

(continues on next page)

(continued from previous page)

```
    "tooltip": ""
  },
```

The property “param” is used to identify the parameter to be used here.

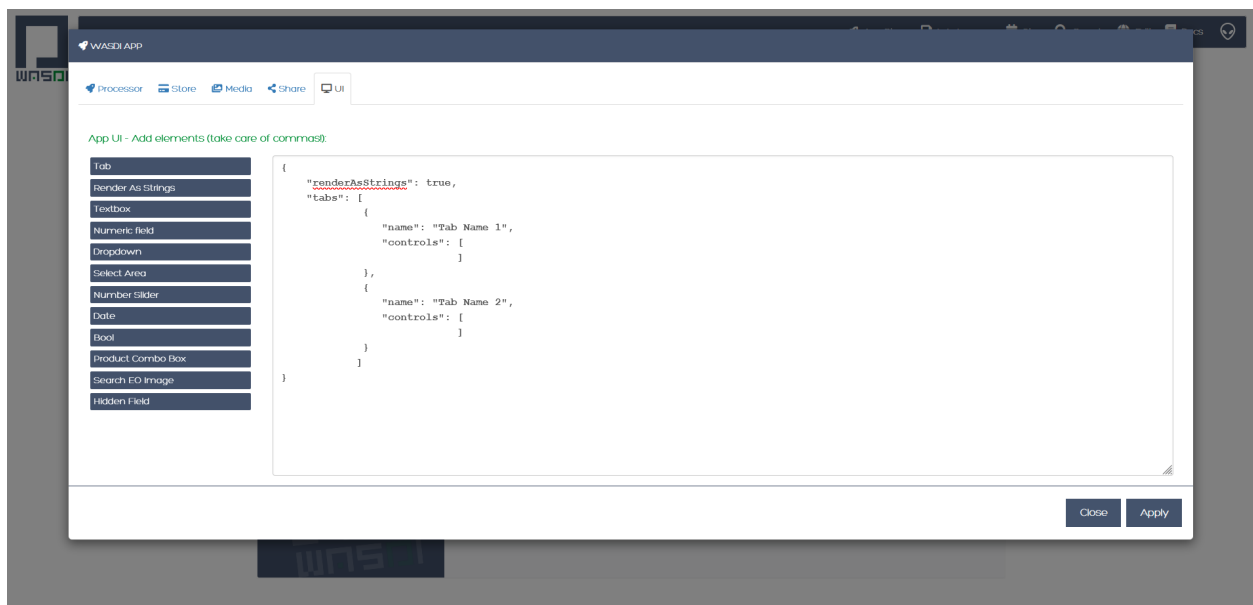
Do not modify the property “type”: “listbox”.

The property “values” is an array of strings that will be the elements of the ListBox.

4.10.14 Render As String

One additional option concerns the button “Render As Strings”. You can add this right after the very first curly brackets (i.e. before the section with the tabs).

Note: Render As String applies to ALL The UI and not to single controls.



The idea behind this button is that, without “Render As Strings” WASDI cannot generate primitive parameters. In other words, without “Render As Strings” a calendar will return a date, a map will return a bbox object, a slider will return a number. But with “Render As Strings”, WASDI will automatically convert all the parameters to strings.

“Render As Strings” is required with IDL and Matlab processors. In case of a Python processor, the developer has the choice between primitive types and strings.

Eventually, the user saves the UI that is then available in the marketplace.

```
{
  "renderAsStrings": true,
  "tabs": [
    {
      "name": "Input",
      "controls": [
        {
          "param": "FILEPRE",
```

(continues on next page)

(continued from previous page)

```
        "type": "productscombo",
        "label": "Pre Flood Image",
        "showExtension": false,
        "required": true
      },
      {
        "param": "FILEPOST",
        "type": "productscombo",
        "label": "Post Flood Image",
        "showExtension": false,
        "required": true
      }
    ]
  }
}
```

4.10.15 Example - Create an actual UI

The following is an example with 3 tabs: the first tab “Tab Name 1” has 3 controls, the second tab “Tab Name 2” has 1 control and the third tab “Tab Name 3” has 1 control. Please note the comma between “Tab Name 1” and “Tab Name 2” and between “Tab Name 2” and “Tab Name 3” (in orange) and the comma between the first and the second control and between the second and the third control in “Tab Name 1” (in purple). All trailing commas have been removed: please check the location of the red crosses.

```

{
  "renderAsStrings": true,
  "tabs": [
    {
      "name": "Tab Name 1",
      "controls": [
        {
          "param": "PARAM_NAME",
          "type": "dropdown",
          "label": "description",
          "default": "",
          "values": [],
          "required": false
        },
        {
          "param": "PARAM_NAME",
          "type": "date",
          "label": "Date",
          "required": false
        },
        {
          "param": "PARAM_NAME",
          "type": "bbox",
          "label": "Bounding Box",
          "required": false
        }
      ]
    },
    {
      "name": "Tab Name 2",
      "controls": [
        {
          "param": "PARAM_NAME",
          "type": "productscombo",
          "label": "Product",
          "required": false,
          "showExtension": false
        }
      ]
    },
    {
      "name": "Tab Name 3",
      "controls": [
        {
          "param": "PARAM_NAME",
          "type": "slider",
          "label": "description",
          "default": 0,
          "min": 0,
          "max": 100,
          "required": false
        }
      ]
    }
  ]
}

```

Now, let's try to reproduce together an example. We use the app developed here [Python Tutorial](#)

The file params.json contains 5 parameters

```
{  
  "BBOX": "45.9,8.5,45.7,8.7",  
  "MAXCLOUD": "50",  
  "DATE": "2020-10-25",  
  "SEARCHDAYS": "20",  
  "PROVIDER": "ONDA"  
}
```

Which means that the UI will contains 5 controls.

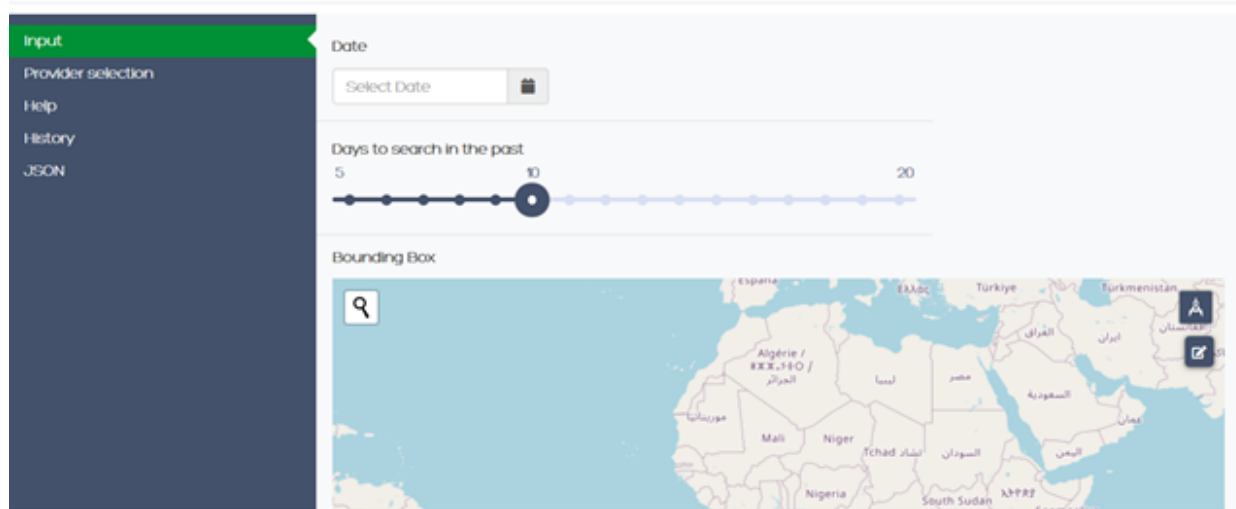
In this case we set 2 Tabs, one named “Input” and the other one named “Provider selection”.

```

{
  "renderAsStrings": true,
  "tabs": [
    {
      "name": "Input",
      "controls": [
        {
          "param": "DATE",
          "type": "date",
          "label": "Date",
          "required": true
        },
        {
          "param": "SEARCHDAYS",
          "type": "slider",
          "label": "Days to search in the past",
          "default": 10,
          "min": 5,
          "max": 20,
          "required": true
        },
        {
          "param": "BBOX",
          "type": "bbox",
          "label": "Bounding Box",
          "required": true
        },
        {
          "param": "MAXCLOUD",
          "type": "slider",
          "label": "Max cloud cover (percent)",
          "default": 30,
          "min": 0,
          "max": 100,
          "required": true
        }
      ]
    },
    {
      "name": "Provider selection",
      "controls": [
        {
          "param": "PROVIDER",
          "type": "dropdown",
          "label": "Data Provider:",
          "default": "ONDA",
          "values": [
            "ONDA",
            "EODC",
            "SOBLOO",
            "CREODIAS"
          ]
        }
      ]
    }
  ]
}

```

The first tab “Input” contains 4 controls (within the squared brackets []), the second tab “Provider selection” contains 1 control (within the squared brackets []). The order in which the tabs appear here is the same order in which they will appear in the UI, as you can see below.



The first tab “Input” is composed of a “date” control, a “slider” control, a “bbox” control and one more “slider” control.

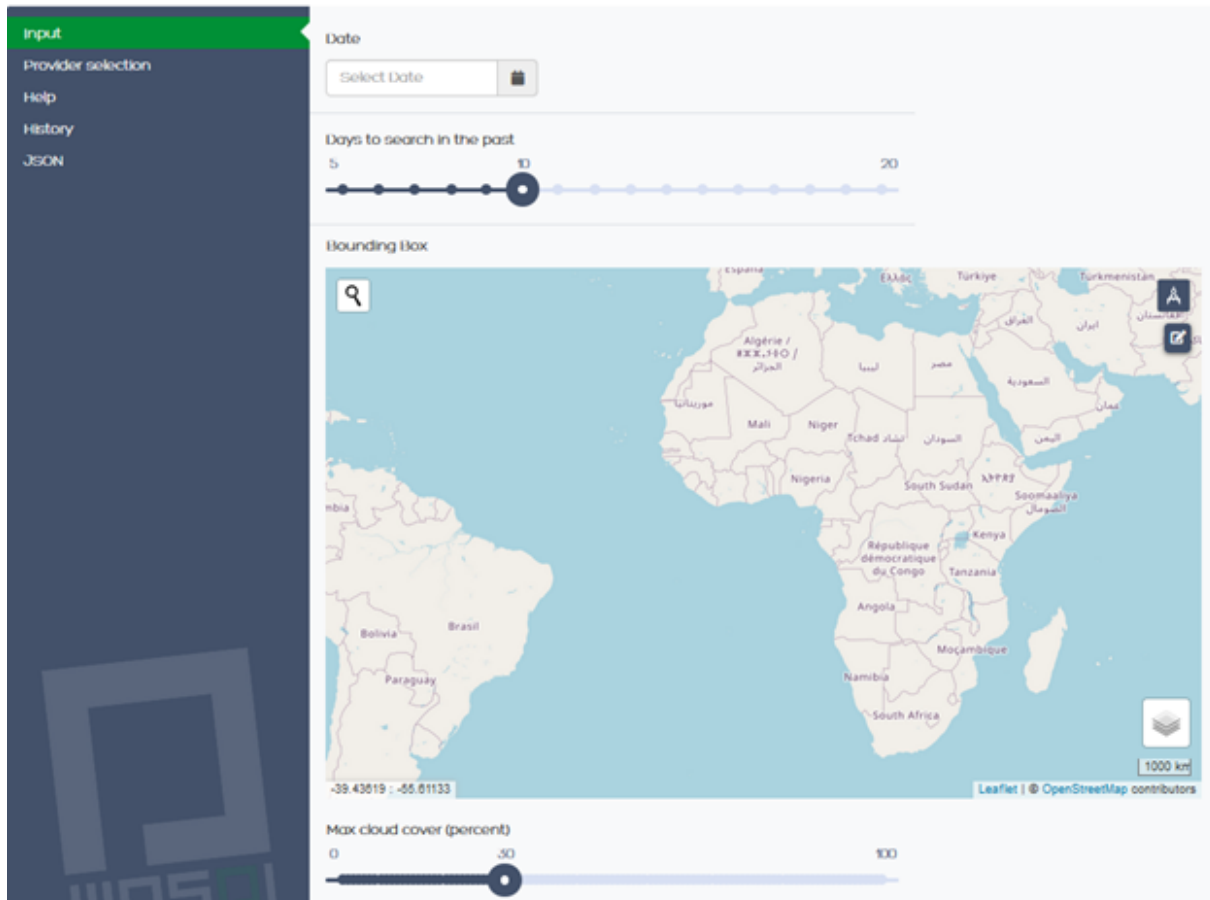
The “date” control refers to the parameter named DATE in the file params.json.

The first “slider” control refers to the parameter named SEARCHDAYS in the file params.json.

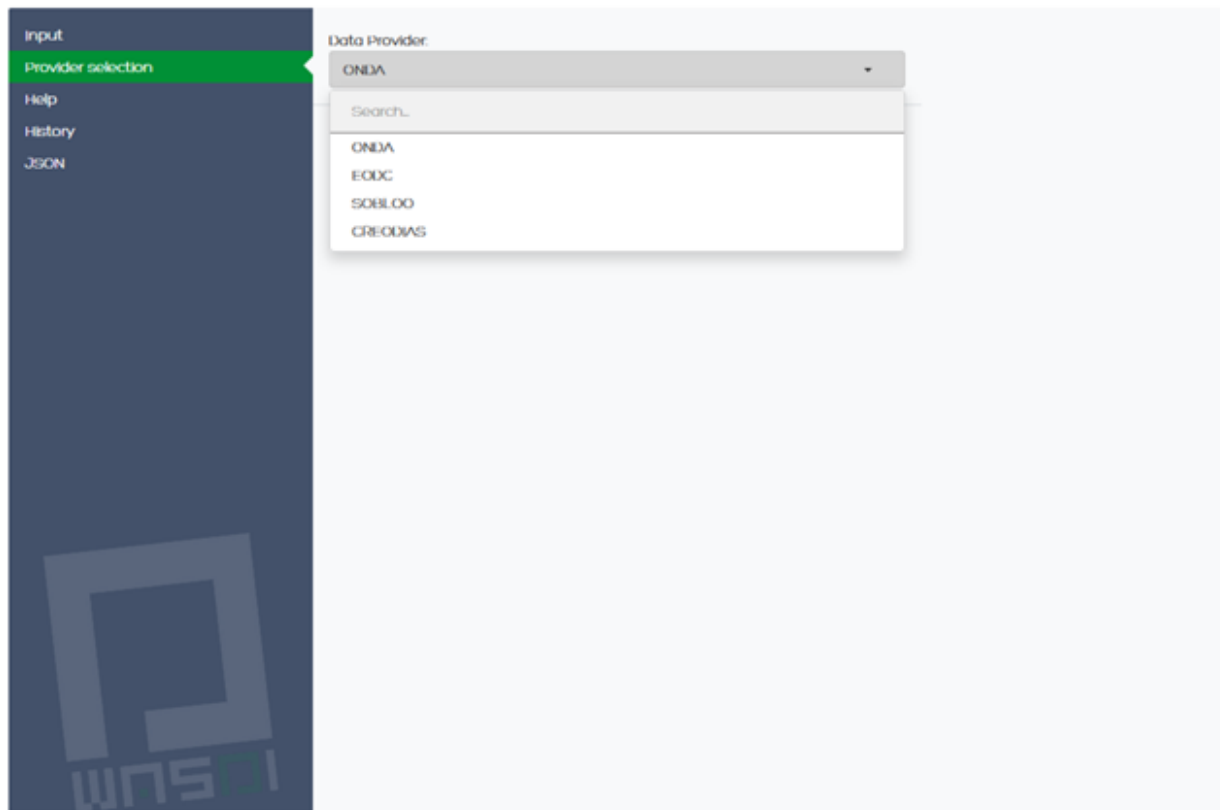
The “bbox” control refers to the parameter named BBOX in the file params.json.

The second “slider” control refers to the parameter named BBOX in the file params.json.

The order in which the controls appear here is the same order in which they will appear in the UI, within the “Input” tab, as you can see below.



The second tab is composed of 1 “dropdown” control. This “dropdown” control refers to the parameter named PROVIDER in the file params.json.



4.11 Javascript Web Tutorial

Note:

To make the most of this tutorial, prior experience with the WASDI platform is required.

For new users, it is highly recommended to follow the [Wasdi Web Platform access and basic usage tutorial](#) before continuing.

Also, to complete the tutorial, a validated account on WASDI is required.

In this tutorial we will show you how you can start using to use the Javascript library for WASDI. In this tutorial we will create a web page that show data gathered through the library just by using one **<script>** tag.

To keep the requirements of this tutorial as small and easy as possible all examples will be using browser-based DOM manipulation: no Javascript frameworks will be used and the produced code will (should) be compatible with any browser. (If it's not it's indeed time for an update ;-))

If you are an Angular developer, please refer to the [Angular dedicated tutorial](#).

4.11.1 Setup & tools

For this tutorial there are no specific tools required.

In general, to write an Html document you can use:

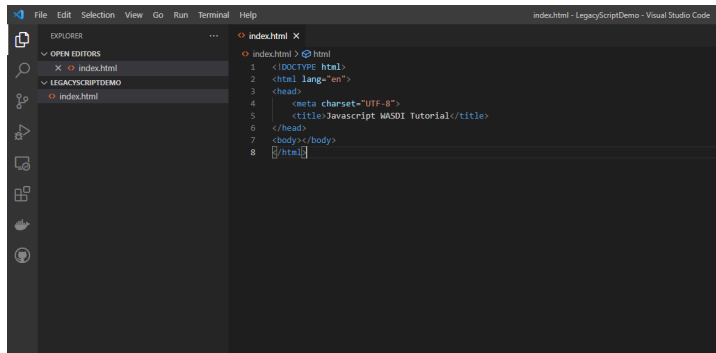
- Microsoft Visual Studio Code (or any HTML compatible IDE)
- -OR-
- A text editor (Notepad, for instance)

The images in the following will show the Visual Studio Code option.

Create index.html file with the following content:

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <title>Js Wasdi Tutorial</title>

</head>
<body></body>
</html>
```



4.11.2 Include the library

The library is served through npm so it is also automatically available through its related CDN at cdn.jsdelivr.net. The current version for Wasdi Javascript library is 0.0.18.

Please check package page on npmjs.org for the latest updates.

Link -> [Wasdi - JavaScript library](#)

Please edit index.html and add the import of the library at the end of the head section

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <title>Js Wasdi Tutorial</title>

<script src="https://cdn.jsdelivr.net/npm/wasdi@0.0.18/build/wasdi-javascript.js"></script>
```

(continues on next page)

(continued from previous page)

```
</head>
<body></body>
</html>
```

Now, to start using the functionalities exposed by the library, create a new file next to index.html and name it **main.js**.

Include the file in index.html as a new `<script>` tag :

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <title>Js Wasdi Tutorial</title>
  <!-- This script loads the library -->
  <script src="https://cdn.jsdelivr.net/npm/wasdi@0.0.18/build/wasdi-javascript.js"></
  <script>
  <!-- This script contains your custom code -->
  <script src="main.js"></script>

</head>
<body></body>
</html>
```

4.11.3 Login

WASDI is a web application that allow users to download, process and obtain results from satellite imagery.

To continue with this tutorial you will need a valid account on the platform: please, proceed to register to WASDI services and keep note of your credentials.

The first step to start interacting with **WASDI** services is to login by using the library facilities.

To achieve this you must add 2 files next the index.html file :

- config.json
- parameters.json

The second file will be introduced later on in the tutorial, when we will start using processors.

Add the following content to config.json, changing **[YOUR_USERNAME]** and **[YOUR_PASSWORD]** with your WASDI credentials

```
{
  "USER": "[YOUR_USERNAME]",
  "PASSWORD": "[YOUR_PASSWORD]"
}
```

Note that this file name, config.json, is **the default value**, if no filename is passed to the method. Please check library documentation for more details about the **loadconfig()** function.

WASDI libraries share the structure of the configuration files. The two fields used in the JSON above represent a sub-set of the available configuration fields.

Check [configuration chapter in Library Concepts](#) section for more details.

For **parameters.json** file, for the moment, please just add the following content:

```
{}
```

The “{}” parenthesis represents an empty JSON object, a quick starting point for the tutorial. In the following we will edit this file adding the actual parameters.

Please open main.js and start editing the file. Wasdi library is exposed as a global singleton, a common practice for Javascript libraries.

The variable to be used to access library methods is “**wasdi**” Add the following lines:

```
// load the configuration from config.json file
wasdi.loadConfig();
// login to Wasdi
wasdi.login();
```

After the successful login call, the wasdi global object will keep its state, allowing to make further requests to the system.

4.11.4 Create Workspaces

A **Workspace** is a basic concept of WASDI: think of it as a folder.

One of the main objective of the platform is to connect to various satellite imagery portals and download files from such services. The workspace is composed by a collection of images downloaded, called **products**.

Data retrieval doesn’t require local storage because it “happens” in the cloud. Also, a workspace holds the information about the elaborations on such data, done by the **processors**. Users can create their own workspaces, and they can also share them with other users.

In the following steps we will add some controls to HTML and some code to our main.js file to create a Workspace on WASDI.

In this step of the tutorial we will use this library call :

```
wasdi.createWorkspace(wsName);
```

The function call can be used to create a workspace in WASDI.

For more information, the library method documentation can be found [here](#)

Wasdi uses a conventional object, the **PrimitiveResult**, as response for, among other, creation calls. This object has the following structure :

```
{
  "IntValue" : 42,
  "StringValue" : "some_string",
  "DoubleValue" : 3.14159265359,
  "BoolValue" : true
}
```

In this case the response will contain a primitive result with only the StringValue setted. The value of the response represents the **workspaceID** an univoque identifier of the workspace.

For more information, the library method documentation can be found [here](#)

Going back to the webpage, please edit the index.html file by adding the following lines, inside the body tags :

```
<p>
Insert workspace name <input type="text" id="wsname">
<input type="button" onclick="createWorkspace()" value="Create Workspace">
</p>
```

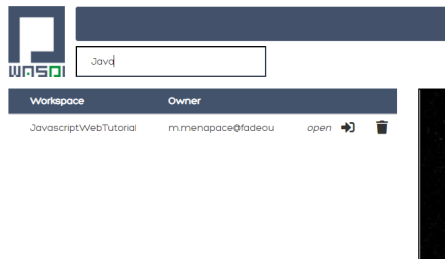
Then open our javascript file *main.js* and define the function `createWorkspace()` :

```
// Local function to create a workspace
createWorkspace = function() {
let wsName = document.getElementById("wsname").value;
// this is the actual call to WASDI services
wasdi.createWorkspace(wsName);
}
```

The function defined will be invoked when the user clicks on the “Create workspace” button. Open the *index.html* page on your browser and you will have a simple form like this:

Insert workspace name

When you click, the system will create a new workspace on WASDI. You can check it in the wasdi web application in the workspaces page:



There it is !

For the following part of the tutorial, we will use this workspace as the default one. This way, for the following features, it will not be necessary to create each time a new workspace.

To open it every time we reload the page, add this statement after the login call, at the beginning of the file *main.js*:

```
wasdi.loadConfig();
wasdi.login();
// From now on this tutorial uses JavascriptWebTutorial workspace as default
wasdi.openWorkspace("JavascriptWebTutorial");
```

For more information, the library method documentation can be found [here](#)

4.11.5 List the available Processors

Another key concept of the WASDI web application is the **Processor**: it represents a tool to gather and elaborate satellite imagery. Processors can be either public or private in WASDI, depending on your subscription. Any user can upload his own code in several languages to create a new Processor. Each processor has a defined set of parameters encoded in a specific JSON and, when we load a processor, a default template is served.

Wasdi has a dedicated section to allow users to parametrize and launch processor. In fact, the UI available in the system just allows to edit the JSON of the parameters before the execution.

In this step of the tutorial we will list the available processors, show them on a selection list and load the parameters of the selected one.

In the following we're gonna use this library call :

```
wasdi.getDeployed();
```

For more information, the library method documentation can be found [here](#)

The library ask for a list of available processors (or apps). The response is an array with each element structured as follow :

```
{
  "imgLink": null,
  "isPublic": 0,
  "minuteTimeout": 180,
  "paramsSample": "%7B%0A%20%20%22name%22:%20%22WASDI%22%0A%7D",
  "processorDescription": "Hello WASDI world for testing purposes",
  "processorId": "22c37982-34f1-4b92-9983-93afb921a8f6",
  "processorName": "hellowasdiworld",
  "processorVersion": "1",
  "publisher": "c.nattero@fadeout.it",
  "sharedWithMe": true,
  "type": "ubuntu_python37_snap"
}
```

The fields above represents a reference to application for WASDI.

One note about **paramsSample**: the value, as you probably noted, is URL-encoded. In this context, in which we are using Javascript, to view and modify the parameters we can use the 2 functions :

- decodeURI() -> To convert sample in a plain string
- encodeURI() -> To re-convert it as URL compatible string

These functions are available natively on any modern Browser/Javascript engine and will be used in the following steps.

Add the following line to the index.html file, containing

- the button to load the deployed processor.
- a selection list that will be populated with the available ones.
- a button to load the parameters of the selected ones.
- a textarea to show the JSON of the parameters.

```
<p>
  <input type="button" onclick="getDeployed()" value="Get processor list">
  <div id="processorList"></div>
```

(continues on next page)

(continued from previous page)

```

</p>

<p>
  <select id="ProcessorSelect" size="8"></select>
  <input type="button" onclick="loadProcessorParameters()" value="Load processor_
↪parameters">
</p>

<p>
  Edit parameters <br>
  <textarea rows="10" cols="100" id="parameters"> </textarea>
</p>

```

Then, open the main.js file and add the definition to actual load the data for the controller defined:

```

getDeployed = function() {
  //Obtain a list of available processors from WASDI
  var deployed = wasdi.getDeployed();
  let selectionList = document.getElementById("ProcessorSelect");

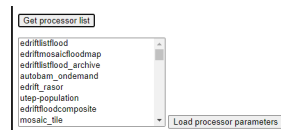
  deployed.forEach(element => {
    let option = document.createElement("option");
    option.text=element.processorName;
    selectionList.add(option);
  });
}

loadProcessorParameters = function(){
  let list = document.getElementById("ProcessorSelect");
  let selectedProcessor = list.options[list.selectedIndex].text;

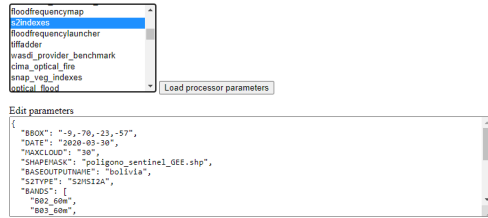
  wasdi.getDeployed().forEach(element => {
    if (element.processorName == selectedProcessor){
      // Here is required the devode URI call
      document.getElementById("parameters").value =decodeURI(element.paramsSample);
    }
  });
}

```

Opening again the index.html and clicking on the first button the list will be populated:



And, after selecting a processor, clicking on the second button the parameters are then showed:



4.11.6 Execute a processor

In this step we will use the data gathered on the previous task of the tutorial to launch an actual application on WASDI. The first approach will be by using a simple test application, which implements a pretty common feature for programming newbie. After that we will introduce the request to obtain the status of the launched processors. This data will be showed by adding a string to the html DOM.

In this step of the tutorial this library call will be used :

```
wasdi.executeProcessor(processorName, parametersJSON);
```

For more information, the library method documentation can be found [here](#)

The methods has two parameters:

- **processorName** the name of the processor that we want to be launched
- **parametersJSON** a JSON string containing the parameters for the processor. As stating point use the template available through getDeployed() library call.

The response to this method has the following structure:

```
{
  "jsonEncodedResult": "",
  "name": "hellowasdiworld",
  "processingIdentifier": "8f09edca-2f7b-4745-aada-bff50cdc6383",
  "processorId": "22c37982-34f1-4b92-9983-93afb921a8f6",
  "status": "CREATED"
}
```

The most important parameter is the **processingIdentifier**: using this will allows us to follow the status of the processing task. In this example, for the sake of clarity, the update will be triggered by the pressing of a button. In any case the call can be integrated in more sophisticated front-end frameworks.

To retrieve the status of the process launched we will use the following library method:

```
wasdi.getProcessStatus(processId);
```

For more information, the library method documentation can be found [here](#)

The response of this method has the following parameters:

```
{
  "fileSize": "",
  "lastChangeDate": "2022-03-16 17:56:44 Z",
  "operationDate": "2022-03-16 17:56:42 Z",
  "operationEndDate": "2022-03-16 17:56:48 Z",
  "operationStartDate": "2022-03-16 17:56:44 Z",
}
```

(continues on next page)

(continued from previous page)

```

    "operationSubType": "",
    "pid": 3860834,
    "payload": "{\"name\": \"WASDI\", \"done\": true, \"the answer is\": 42}",
    "processObjId": "8f09edca-2f7b-4745-aada-bff50cdc6383",
    "productName": "hellowasdiworld",
    "progressPerc": 100,
    "status": "DONE",
    "userId": "m.menapace@fadeout.it"
  }

```

Across the several fields of the response, the ones used in this tutorial are :

- **productName** which identifies the processor name, “hellowasdiworld” in this example.
- **status** represents the possible state of the processor among: { WAITING | RUNNING | DONE | ERROR }.
- **progressPerc** is a number indicating the percentage of the progress for the current processing work.
- **payload** is a JSON which contains information about the outcome of the elaboration.

You can check their usage in the **getProcessorString** function definition in the following javascript snippets.

Open index.html and add the following components inside the `<body>` tags:

```

<p>
  <input type="button" onclick="executeProcessor()" value="Execute processor">
</p>

<p>
  <input type="button" onclick="getStatus()" value="Get status of processor launched">
  <div id="processorStatus"> </div>
</p>

```

First, in order to have a support variable keeping the launched process from this webpage, add this line at the top of the *main.js* file

```
var launchedProcessorID=[];
```

Then add the following methods to *main.js*:

```

executeProcessor = function() {
let list = document.getElementById("ProcessorSelect");
let selectedProcessor = list.options[list.selectedIndex].text;
let parameters = document.getElementById("parameters").value;
let response = wasdi.executeProcessor(selectedProcessor, encodeURIComponent(parameters));
console.log(response.processingIdentifier);
launchedProcessorID.push(response.processingIdentifier);
}

// Util function to render a formatted string from the process status response
getProcessorString = function(status) {
let response = "";
response = response.concat("Processor name " + status.productName + " | " + "status " +
↳ status.status + " | % " + status.progressPerc + " | Payload " + status.payload );

```

(continues on next page)

(continued from previous page)

```

return response;
}

getStatus = function() {
document.getElementById("processorStatus").innerHTML = "";
launchedProcessorID.forEach(element => {
    let status = wasdi.getProcessStatus(element);
    document.getElementById("processorStatus").innerHTML = document.getElementById(
        "processorStatus").innerHTML.concat(
        getProcessorString(status) + "<br>"
    );
});
}
}

```

The first function *executeProcessor* invoke the wasdi library method to run a processor (remember, on the workspace “JavascriptWebTutorial”).

The second function *getProcessorString* it’s an util method to shown the process status of the processes started from the current page.

The last function use the wasdi library to gather the data of the launched processors and push the formatted result on a dedicated div.

We can then test the page by launching the application **hellowasdiworld**: after clicking on both buttons, *execute processor* and *Get status of processor launched* a string with the status will showed :

Get processor list

grd_dualband_avg
hobalfood
lff_images_average
slc_dual_band_coherence_avg
uif_slc_preproc
alutbamaps
TestApplicationMM
hellowasdiworld

Load processor parameters

Edit parameters

```
{
  "name": "WASDI"
}
```

Execute processor

Get status of processor launched

Processor name hellowasdiworld | status DONE | % 100 | Payload {"name": "WASDI", "done": true, "the answer is": 42}

If you open WASDI on wasdi.net, login with your user credentials and open the workspace, you will see that the processor were executed:

Workspace Processed List								
STATUS ▾	TYPE ▾	NAME	DATE	T Apply Filters	Reset Filters	Download		
Operation	Name	User	Size	Created	Started	Progress	Duration	
OK	APP	hellowasdiworld	hellowasdiworld@wasdi.net		2022-03-16 10:03:12	2022-03-16 10:03:12	100%	00:00:04
Load More								

4.12 Javascript Angular Tutorial

Note:

To make the most of this tutorial, prior experience with the WASDI platform is required.

For new users, it is highly recommended to follow the [Wasdi Web Platform access and basic usage tutorial](#) before continuing.

Also, to complete the tutorial, a validated account on WASDI is required.

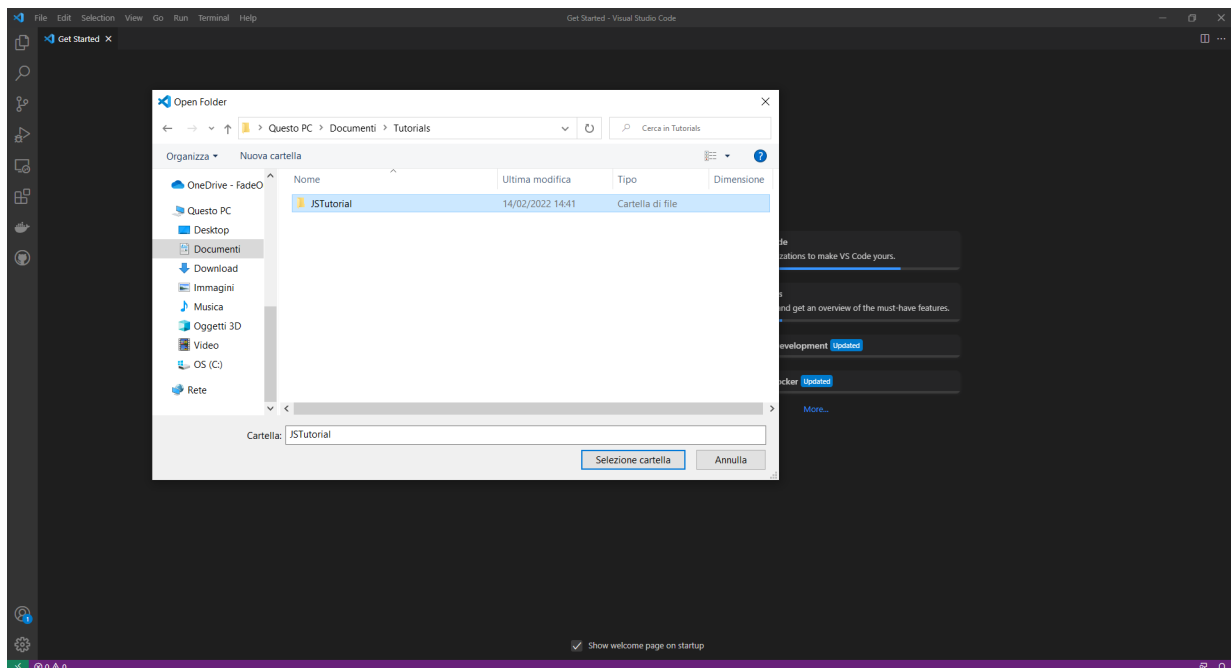
In this tutorial we will introduce WASDI javascript library. To showcase the usage of the library and how it connects to WASDI services, an easy application will be created using Angular, a popular framework from Google.

4.12.1 Setup

The requirements for this tutorial are :

- Microsoft Visual Studio Code (or any TS/JS compatible IDE)
- npm, node package manager installed on the dev machine
- Angular Cli, which will help in creating the project and adding components

Open VSCode, select open folder and select the parent directory for the project. The project folder will be created in the following steps.



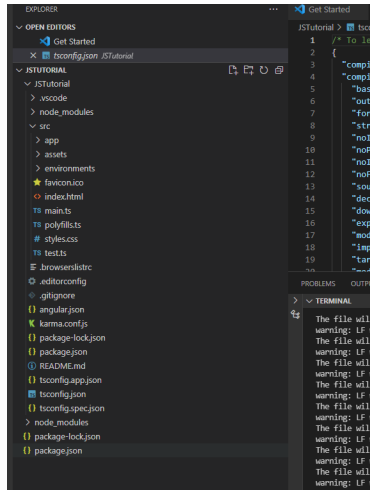
Assuming you have already installed npm globally in the system, open a terminal and install Angular Cli with the following command :

```
npm i @angular/cli
```

The Angular Cli will add the ng command that we use to generate a new project on the current folder

```
ng new JSTutorial
```

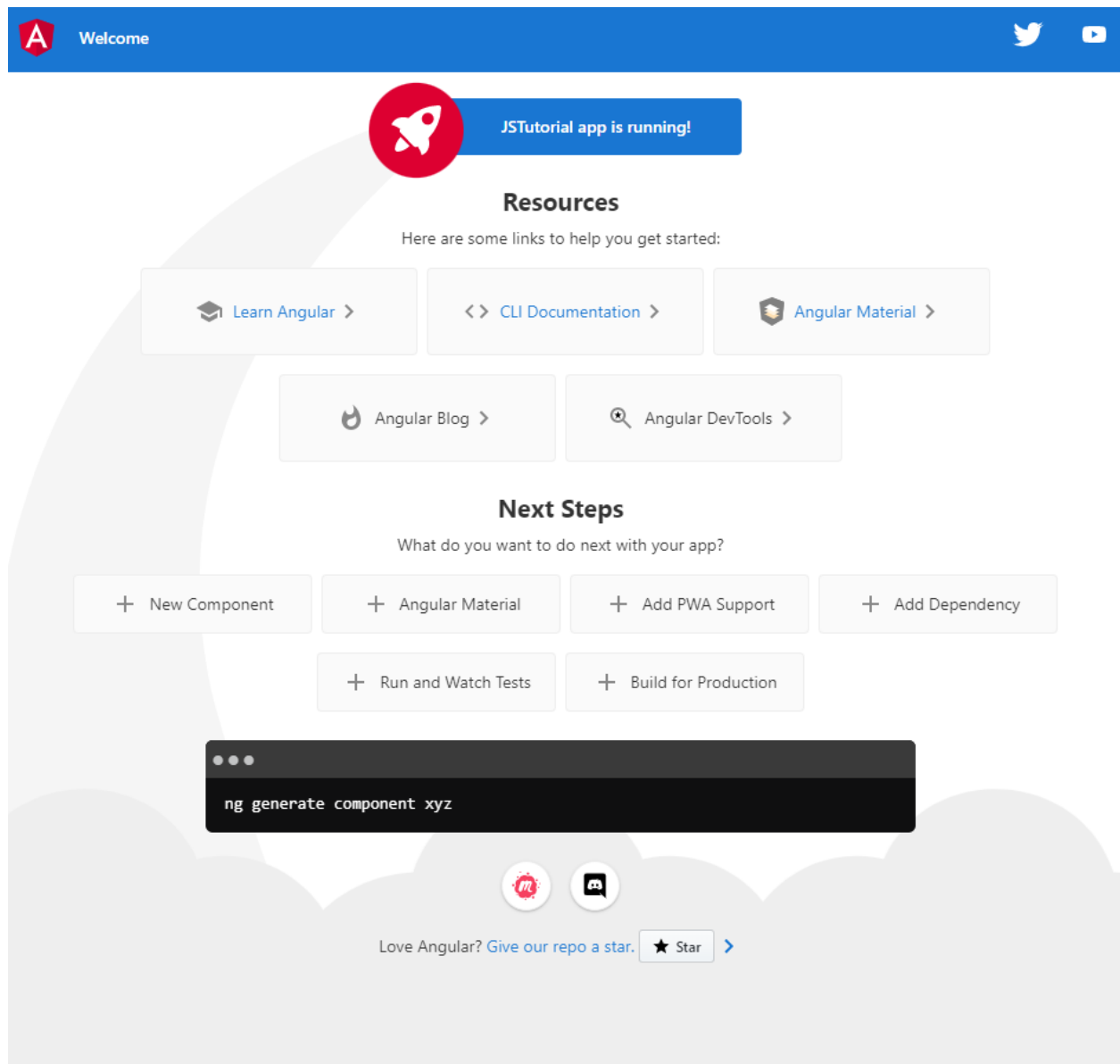
This command will take some time but, at the end, you'll end up with a new and complete Angular application, ready to go!



For the sake of this tutorial we will not showcase all the framework functionalities. The objective of the tutorial is to make a brief introduction to WASDI library. For further details about Angular, please check project documentation. Just to check that everything is up and running raise the following command :

```
ng serve
```

Open then a browser and navigate to <http://localhost:4200>, starter application will show up.



This is only a filler provided by Angular team, we can remove it. Open `src/app/app.component.html` and delete its content.

The application will be now empty, don't worry that's exactly what we want.

4.12.2 Importing the library

Now we need to add the dependency to WASDI lib: the library is hosted on npm repository so, to install it, we can use this command on terminal:

```
npm i wasdi
```

Now that Wasdi lib is installed we need to add to angular its capabilities. To have an injectable service, we can use the following command:

```
ng generate service WasdiService
```

This will add to our Angular application a new service called WasdiService. We will use this service to access library capabilities. We need a little bit of setup, now: please open wasdi-service.service.ts file. Add * Import of the wasdi library * A method to access the wasdi instance * Set username and password to the library * Login use wasdi.login() to obtain a session

```
ts wasdi-service.service.ts U X
src > app > ts wasdi-service.service.ts > WasdiServiceService > constructor
1 import { Injectable } from '@angular/core';
2 import wasdi from 'wasdi';
3
4 @Injectable({
5   providedIn: 'root'
6 })
7 export class WasdiServiceService {
8
9   lib() {
10     return wasdi;
11   }
12
13   constructor() {
14     wasdi._u_user = "MY USER NAME";
15     wasdi._u_password = "MY PASSWORD";
16     wasdi.login();
17   }
18 }
```

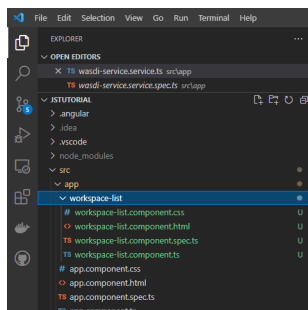
Great ! now we have a service that can connect to WASDI and use functionalities exposed.

4.12.3 Using the library

Next step is to add an angular component that will show the list of workspaces of the current user. First create the component with :

```
ng generate component WorkspaceList
```

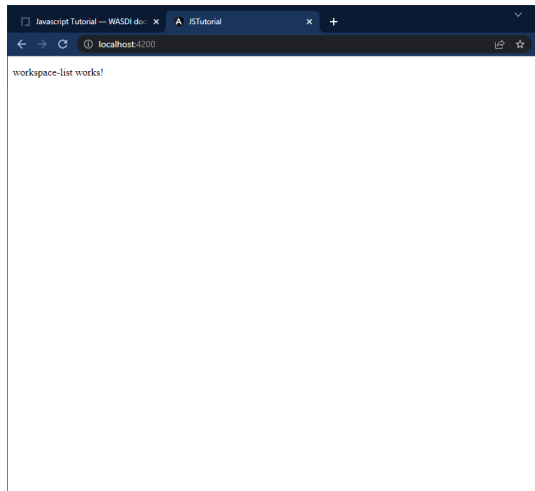
this will create a workspace-list folder, with all the files and a starter implementation of the component:



Try to open again app.component.html and add the following line :

```
<app-workspace-list></app-workspace-list>
```

If you serve again the app and open localhost:4200 you will see the following:



Now the objective is to briefly show the workspace coming from WASDI server: to do this open the ts file of our workspace-list component and add the following

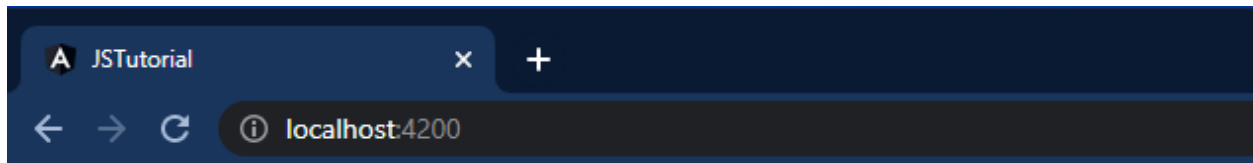
- Inject our WasdiService in the constructor of the component
- Create the variable wsList which will hold the workspace list
- Initialize the variable by using the lib, as follows

```
ts wasdi-service.service.ts workspace-list.component.ts workspace-list.component.html
src > app > workspace-list > workspace-list.component.ts > WorkspaceListComponent > wsList
1 import { Component, OnInit } from '@angular/core';
2 import { WasdiServiceService } from '../wasdi-service.service';
3
4 @Component({
5   selector: 'app-workspace-list',
6   templateUrl: './workspace-list.component.html',
7   styleUrls: ['./workspace-list.component.css']
8 })
9 export class WorkspaceListComponent implements OnInit {
10   wsList: any;
11   constructor(private wasdiService: WasdiServiceService) {}
12
13   ngOnInit(): void {
14     // Get our list with the workspace list from WASDI
15     this.wsList = this.wasdiService.lib().workspaceList();
16   }
17
18 }
19
```

Open now the html file and add the following code, which loop over element in wsList to render them as paragraph :

```
ts wasdi-service.service.ts workspace-list.component.ts workspace-list.component.html
src > app > workspace-list > workspace-list.component.html > ...
Go to component
1
2 <!-- Creates a paragraph with each workspace from the user -->
3
4 <p *ngFor="let ws of wsList">
5   {{ws.workspaceName}}
6 </p>
7
```

Open now the webpage and you will see a list of the workspaces coming from your WASDI account!



BE - Flood 15/07/2021

DE - Flood 15/07/2021

TestSnapIntermediateFiles

LUX-NDVI-Archive

Kenya

S1 Level1 for LUX NDVI Archive

SEN2CORE

WaspyTest

The tutorial ends here. We have briefly showed how can a JS/TS application can interact with WASDI. Please refer to the official documentation of the library for a complete cover of alla the functionalities.

REFERENCE CENTER

WASDI allows users and developer to interact though **libraries** and **APIs**. Find the reference of your language library on the left menu.

5.1 C# WasdiLib

```
public class Wasdi
```

5.1.1 Fields

```
m_sUser
```

```
private string m_sUser
```

5.1.2 Constructors

Wasdi

```
public Wasdi()
```

Self constructor. If there is a config file initializes the class members

5.1.3 Methods

Init

```
public bool Init(string sConfigFilePath)
```

Init the WASDI Library starting from a configuration file. If the path is not provided, the application will attempt to use the conventional appsettings.json file.

Parameters

- **sConfigFilePath** – the name of the file to be added to the open workspace

Returns

True if the system is initialized, False if there is any error

InternalInit

public bool **InternalInit**(string *sConfigFilePath*)

Call this after base parameters settings to init the system. Needed at least: Base Path, User, Password or SessionId.

Returns

True if the system is initialized, False if there is any error

AddFileToWASDI

public string **AddFileToWASDI**(string *sFileName*)

Adds a generated file to current open workspace in a synchronous way.

Parameters

- **sFileName** – the name of the file to be added to the open workspace

Returns

the process Id or empty string in case of any issues

GetDefaultProvider

public string **GetDefaultProvider**()

Explicit accessor for the defaultProvider property.

Returns

the defaultProvider

SetDefaultProvider

public void **SetDefaultProvider**(string *sProvider*)

Explicit mutator for the defaultProvider property.

Parameters

- **sProvider** – the provider to be used by default

GetUser

public string **GetUser**()

Explicit accessor for the user property.

Returns

the user

SetUser

public void **SetUser**(string *sUser*)

Explicit mutator for the user property.

Parameters

- **sUser** – the user

GetPassword

public string **GetPassword**()

Explicit accessor for the password property.

Returns

the password

SetPassword

public void **SetPassword**(string *sPassword*)

Explicit mutator for the password property.

Parameters

- **sPassword** – the password

GetActiveWorkspace

public string **GetActiveWorkspace**()

Explicit accessor for the activeWorkspace property.

Returns

the activeWorkspace

SetActiveWorkspace

public void **SetActiveWorkspace**(string *sNewActiveWorkspaceId*)

Explicit mutator for the activeWorkspace property. If the new active workspace is not null, sets also the workspace owner.

Parameters

- **sNewActiveWorkspaceId** – the new Id of the activeWorkspace

GetSessionId

public string **GetSessionId**()

Explicit accessor for the sessionId property.

Returns

the sessionId

SetSessionId

public void **SetSessionId**(string *sSessionId*)

Explicit mutator for the sessionId property. Sets the sessionId only if the input is not null.

Parameters

- **sSessionId** – the sessionId

GetBaseUrl

public string **GetBaseUrl**()

Explicit accessor for the baseUrl property.

Returns

the baseUrl

SetBaseUrl

public void **SetBaseUrl**(string *sBaseUrl*)

Explicit mutator for the baseUrl property. Sets the baseUrl only if the input is not null and if it represents a valid URI.

Parameters

- **sBaseUrl** – the new baseUrl

GetIsOnServer

public bool **GetIsOnServer**()

Explicit accessor for the isOnServer property.

Returns

True if the application is deployed on server, False if it is running on local development machine

SetIsOnServer

public void **SetIsOnServer**(bool *bIsOnServer*)

Explicit mutator for the isOnServer property.

Parameters

- **bIsOnServer** – Indicates whether the application is deployed on server or running on local development machine

GetDownloadActive

public bool **GetDownloadActive**()

Explicit accessor for the downloadActive property.

Returns

the value of the downloadActive flag

SetDownloadActive

public void **SetDownloadActive**(bool *bDownloadActive*)

Explicit mutator for the downloadActive property.

Parameters

- **bDownloadActive** – the desired value of the downloadActive flag

GetBasePath

public string **GetBasePath**()

Explicit accessor for the basePath property.

Returns

the basePath

SetBasePath

public void **SetBasePath**(string *sBasePath*)

Explicit mutator for the basePath property. Sets the basePath only if the input is not null and if it represents a valid path and the user has the permissions to read and write.

Parameters

- **sBasePath** – the new basePath

GetMyProcId

public string **GetMyProcId**()

Explicit accessor for the myProcId property.

Returns

the myProcId

SetMyProcId

public void **SetMyProcId**(string *sMyProcId*)

Explicit mutator for the myProcId property. Set the myProcessId only if the input is not null or empty.

Parameters

- **sMyProcId** – the value of myProcId

GetVerbose

public bool **GetVerbose**()

Explicit accessor for the verbose property.

Returns

the value of the verbose flag

SetVerbose

public void **SetVerbose**(bool *bVerbose*)

Explicit mutator for the verbose property. If the verbose flag is set to True, the level of logging is INFORMATION. If the verbose flag is set to False, the level of logging is ERROR.

Parameters

- **bVerbose** – the desired value of the verbose flag

GetParams

public Dictionary<string, string> **GetParams**()

Get the parameters (except for the user, sessionId and workspaceid).

Returns

the Params dictionary

GetParamsAsJsonString

public string **GetParamsAsJsonString**()

Get the parameters in Json format.

Returns

the parameters as a Json string

AddParam

public void **AddParam**(string *sKey*, string *sParam*)

Add a parameter to the parameters dictionary.

Parameters

- **sKey** – the new key
- **sParam** – the new value

GetParam

```
public string GetParam(string sKey)
```

Get a specific parameter from the parameters dictionary. If the key is not contained by the dictionary, an empty string is returned.

Parameters

- **sKey** – the key

Returns

the value corresponding to the key or an empty string

GetParametersFilePath

```
public string GetParametersFilePath()
```

Explicit accessor for the parametersFilePath property.

Returns

the parameters file path

SetParametersFilePath

```
public void SetParametersFilePath(string sParametersFilePath)
```

Sets the parametersFilePath only if the input is not null and if it represents a valid path.

Parameters

- **sParametersFilePath** – the parameters file path

CreateSession

```
public string CreateSession(string sUser, string sPassword)
```

Create a new session and return its Id.

Parameters

- **sUser** – the username
- **sPassword** – the password

Returns

the newly created sessionId

CheckSession

```
public string CheckSession(string sSessionId, string sUser)
```

Check the session.

Parameters

- **sSessionId** – the actual session Id
- **sUser** – the username of the expected user

Returns

True if the actual user is the same as the expected user, false otherwise

GetWorkspaceBaseUrl

public string **GetWorkspaceBaseUrl**()

Explicit accessor for the workspaceBaseUrl property.

Returns

the workspace's baseUrl

SetWorkspaceBaseUrl

public void **SetWorkspaceBaseUrl**(string *sWorkspaceBaseUrl*)

Sets the workspace's baseUrl only if the input is not null and if it represents a valid URI.

Parameters

- **sWorkspaceBaseUrl** – the new baseUrl of the workspace

Hello

public string **Hello**()

Call the hello endpoint and return thre response.

Returns

the response of the server or null in case of any error

GetWorkspaces

public List<Workspace> **GetWorkspaces**()

Get the list of workspaces of the logged user.

Returns

the list of workspaces or null in case of any error

GetWorkspacesNames

public List<string> **GetWorkspaces**()

Get the list of workspaces' names of the logged user.

Returns

the list of workspaces' names or an empty list in case of any error

GetWorkspaceIdByName

public string **GetWorkspaceIdByName**(string *sWorkspaceName*)

Get the Id of a workspace identified by name.

Parameters

- **sWorkspaceName** – the name of the workspace

Returns

the Id of the workspace or an empty string in case of an error or if there is no workspace with the name indicated

GetWorkspaceNameById

public string **GetWorkspaceNameById**(string *sWorkspacesId*)

Get the name of a workspace identified by Id.

Parameters

- **sWorkspacesId** – the Id of the workspace

Returns

the name of the workspace or an empty string in case of an error or if there is no workspace with the Id indicated

GetWorkspaceOwnerByName

public string **GetWorkspaceOwnerByName**(string *sWorkspacesId*)

Get the userId of the owner of a workspace identified by name.

Parameters

- **sWorkspacesId** – the name of the workspace

Returns

the user Id of the workspace's owner or an empty string in case of an error or if there is no workspace with the name indicated

GetWorkspaceOwnerByWSId

public string **GetWorkspaceOwnerByWSId**(string *sWorkspaceId*)

Get the userId of the owner of a workspace identified by Id.

Parameters

- **sWorkspaceId** – the Id of the workspace

Returns

the user Id of the workspace's owner or an empty string in case of an error or if there is no workspace with the Id indicated

GetWorkspaceUrlByWsId

public string **GetWorkspaceUrlByWsId**(string *sWorkspaceId*)

Get the workspace's URL of a workspace identified by Id.

Parameters

- **sWorkspaceId** – the Id of the workspace

Returns

the workspace's URL or an empty string in case of an error or if there is no workspace with the Id indicated

OpenWorkspaceById

public string **OpenWorkspaceById**(string *sWorkspaceId*)

Open a workspace given its Id.

Parameters

- **sWorkspaceId** – the Id of the workspace

Returns

the workspace Id if opened successfully, empty string otherwise

OpenWorkspace

public string **OpenWorkspace**(string *sWorkspaceName*)

Open a workspace.

Parameters

- **sWorkspaceName** – Workspace name to open

Returns

the workspace Id if opened successfully, empty string otherwise

GetProductsByWorkspace

public List<string> **GetProductsByWorkspace**(string *sWorkspaceName*)

Get a List of the products in a workspace.

Parameters

- **sWorkspaceName** – the name of the workspace

Returns

List of Strings representing the product names

GetProductsByWorkspaceId

public List<string> **GetProductsByWorkspaceId**(string *sWorkspaceId*)

Get a List of the products in a Workspace

Parameters

- **sWorkspaceId** – the Id of the workspace

Returns

List of Strings representing the product names

GetProductsByActiveWorkspace

public List<string> **GetProductsByWorkspaceId()**

Get a List of the products in the active workspace

Returns

List of Strings representing the product names

GetProductName

public string **GetProductName**(Dictionary<string, object> *oProduct*)

Get the name of the product provided. For the names starting with S1 or S2, add the .zip extension in case it is missing.

Parameters

- **oProduct** – the product

Returns

the name of the product or null in case of any error

ImportAndPreprocessWithLinks

public void **ImportAndPreprocessWithLinks**(List<string> *asProductsLink*, List<string> *asProductsNames*,
string *sWorkflow*, string *sPreProcSuffix*)

Import and pre-process with links.

Parameters

- **asProductsLink** – the list of product links
- **asProductsNames** – the list of product names
- **sWorkflow** – the workflow
- **sPreProcSuffix** – the pre-process suffix

ImportAndPreprocessWithLinks

public void **ImportAndPreprocessWithLinks**(List<string> *asProductsLink*, List<string> *asProductsNames*,
string *sWorkflow*, string *sPreProcSuffix*, string *sProvider*)

Import and pre-process with links.

Parameters

- **asProductsLink** – the list of product links
- **asProductsNames** – the list of product names
- **sWorkflow** – the workflow
- **sPreProcSuffix** – the pre-process suffix
- **sProvider** – the provider

ImportAndPreprocess

```
public void ImportAndPreprocess(List<Dictionary<string, object>> aoProductsToImport, string sWorkflow,  
                                string sPreProcSuffix)
```

Import and pre-process.

Parameters

- **aoProductsToImport** – the list of products
- **sWorkflow** – the workflow
- **sPreProcSuffix** – the pre-process suffix

ImportAndPreprocess

```
public void ImportAndPreprocess(List<Dictionary<string, object>> aoProductsToImport, string sWorkflow,  
                                string sPreProcSuffix, string sProvider)
```

Import and pre-process.

Parameters

- **aoProductsToImport** – the list of products
- **sWorkflow** – the workflow
- **sPreProcSuffix** – the pre-process suffix
- **sProvider** – the provider

AsynchPreprocessProductsOnceDownloaded

```
public List<string> AsynchPreprocessProductsOnceDownloaded(List<Dictionary<string, object>>  
                                                            aoProductsToImport, string sWorkflow, string  
                                                            sPreProcSuffix, List<string> asDownloadIds)
```

Asynchronously pre-process products once they are downloaded.

Parameters

- **aoProductsToImport** – the list of products
- **sWorkflow** – the workflow
- **sPreProcSuffix** – the pre-process suffix
- **asDownloadIds** – the list of download Ids

Returns

the list of workflow ids

AsynchPreprocessProductsOnceDownloadedWithNames

```
public List<string> AsynchPreprocessProductsOnceDownloadedWithNames(List<string> asProductsNames,
                                                                    string sWorkflow, string
                                                                    sPreProcSuffix, List<string>
                                                                    asDownloadIds)
```

Asynchronously pre-process products once they are downloaded and names are provided.

Parameters

- **asProductsNames** – the list of product names
- **sWorkflow** – the workflow
- **sPreProcSuffix** – the pre-process suffix
- **asDownloadIds** – the list of download Ids

Returns

the list of workflow ids

GetPath

```
public string GetPath(string sProductName)
```

Get the local path of a file.

Parameters

- **sProductName** – the name of the file

Returns

the full local path

InternalGetFullProductPath

```
public string InternalGetFullProductPath(string sProductName)
```

Get the full local path of a product given the product name. Use the output of this API to open the file.

Parameters

- **sProductName** – the product name

Returns

the product's full path as a String ready to open file

GetSavePath

```
public string GetSavePath()
```

Get the local Save Path to use to save custom generated files.

Returns

the local path to use to save a custom generated file

GetWorkflows

```
public List<Workflow> GetWorkflows()
```

Get the list of workflows for the user.

Returns

the list of the workflows or null in case of any error

AsynchExecuteWorkflow

```
public string AsynchExecuteWorkflow(List<string> asInputFileNames, List<string> asOutputFileNames, string  
                                     sWorkflowName)
```

Executes a WASDI SNAP Workflow in a asynch mode.

Parameters

- **asInputFileNames** – the list of input file names
- **asOutputFileNames** – the list of output file names
- **sWorkflowName** – the workflow's name

Returns

the Id of the workflow process or empty string in case of any issue

ExecuteWorkflow

```
public string AsynchExecuteWorkflow(List<string> asInputFileNames, List<string> asOutputFileNames, string  
                                     sWorkflowName)
```

Executes a WASDI SNAP Workflow waiting for the process to finish.

Parameters

- **asInputFileNames** – the list of input file names
- **asOutputFileNames** – the list of output file names
- **sWorkflowName** – the workflow's name

Returns

output status of the Workflow Process

GetProcessStatus

```
public string GetProcessStatus(string sProcessId)
```

Get WASDI Process Status.

Parameters

- **sProcessId** – the process's Id

Returns

process Status as a String: CREATED, RUNNING, STOPPED, DONE, ERROR, WAITING, READY

GetProcessesStatus

public string **GetProcessesStatus**(List<string> *asIds*)

Get the status of a List of WASDI processes.

Parameters

- **asIds** – the list of processes Ids

Returns

Process Status as a String: CREATED, RUNNING, STOPPED, DONE, ERROR, WAITING, READY

GetProcessesStatusAsList

public string **GetProcessesStatusAsList**(List<string> *asIds*)

Get the status of a List of WASDI processes.

Parameters

- **asIds** – the list of processes Ids

Returns

Process Status as a String: CREATED, RUNNING, STOPPED, DONE, ERROR, WAITING, READY

UpdateStatus

public string **UpdateStatus**(string *sStatus*)

Update the status of the current process.

Parameters

- **sStatus** – the status to set

Returns

updated status as a String or empty string in case of any issue

UpdateStatus

public string **UpdateStatus**(string *sStatus*, int *iPerc*)

Update the status of the current process.

Parameters

- **sStatus** – the status to set
- **iPerc** – the progress in %

Returns

updated status as a String or empty string in case of any issue

UpdateProcessStatus

public string **UpdateProcessStatus**(string *sProcessId*, string *sStatus*, int *iPerc*)

Update the status of a process.

Parameters

- **sProcessId** – the process Id
- **sStatus** – the status to set
- **iPerc** – the progress in %

Returns

updated status as a String or empty string in case of any issue

UpdateProgressPerc

public string **UpdateProgressPerc**(int *iPerc*)

Update the status of a process.

Parameters

- **iPerc** – the progress in %

Returns

updated status as a String or empty string in case of any issue

WaitProcess

public string **WaitProcess**(string *sProcessId*)

Wait for a process to finish.

Parameters

- **sProcessId** – the process Id

Returns

the process status

WaitProcesses

public List<string> **WaitProcesses**(List<string> *asIds*)

Wait for a collection of processes to finish.

Parameters

- **asIds** – the list of processes Ids

Returns

the list of process statuses

SetPayload

public void **SetPayload**(string *sData*)

Set the payload of the current process (only if the input is not null or empty).

Parameters

- **sData** – the payload as a String. JSON format recommended

SetProcessPayload

public string **SetProcessPayload**(string *sProcessId*, string *sData*)

Set the payload of the current process (only if the input is not null or empty).

Parameters

- **sProcessId** – the process Id
- **sData** – the payload as a String. JSON format recommended

Returns

the status of the process or empty string in case of any issues

RefreshParameters

public void **RefreshParameters**()

Refresh Parameters reading again the file.

AddFileToWASDI

public string **AddFileToWASDI**(string *sFileName*, string *sStyle*)

Adds a generated file to current open workspace in synchronous way.

Parameters

- **sFileName** – the name of the file to be added to the open workspace
- **sStyle** – name of a valid WMS style

Returns

the process Id or empty string in case of any issues

AsynchAddFileToWASDI

public string **AsynchAddFileToWASDI**(string *sFileName*, string *sStyle*)

Adds a generated file to current open workspace in asynchronous way.

Parameters

- **sFileName** – the name of the file to be added to the open workspace
- **sStyle** – name of a valid WMS style

Returns

the process Id or empty string in case of any issues

AddFileToWASDI

public string **AddFileToWASDI**(string *sFileName*)

Adds a generated file to current open workspace in synchronous way.

Parameters

- **sFileName** – the name of the file to be added to the open workspace

Returns

the process Id or empty string in case of any issues

AsynchAddFileToWASDI

public string **AsynchAddFileToWASDI**(string *sFileName*)

Adds a generated file to current open workspace in asynchronous way.

Parameters

- **sFileName** – the name of the file to be added to the open workspace

Returns

the process Id or empty string in case of any issues

Mosaic

public string **Mosaic**(List<string> *asInputFiles*, string *sOutputFile*)

Mosaic with minimum parameters: input and output files.

Parameters

- **asInputFiles** – the list of input files to mosaic
- **sOutputFile** – the name of the mosaic output file

Returns

the end status of the process

Mosaic

public string **Mosaic**(List<string> *asInputFiles*, string *sOutputFile*, string *sNoDataValue*, string *sInputIgnoreValue*)

Mosaic with minimum parameters: input and output files.

Parameters

- **asInputFiles** – the list of input files to mosaic
- **sOutputFile** – the name of the mosaic output file
- **sNoDataValue** – the value to use in output as no data
- **sInputIgnoreValue** – the value to use as input no data

Returns

the end status of the process

Mosaic

```
public string Mosaic(List<string> asInputFiles, string sOutputFile, string sNoDataValue, string sInputIgnoreValue,
    double dPixelSizeX, double dPixelSizeY)
```

Mosaic with minimum parameters: input and output files.

Parameters

- **asInputFiles** – the list of input files to mosaic
- **sOutputFile** – the name of the mosaic output file
- **sNoDataValue** – the value to use in output as no data
- **sInputIgnoreValue** – the value to use as input no data
- **dPixelSizeX** – the X Pixel Size
- **dPixelSizeY** – the Y Pixel Size

Returns

the end status of the process

AsynchMosaic

```
public string Mosaic(List<string> asInputFiles, string sOutputFile)
```

Asynch Mosaic with minimum parameters: input and output files.

Parameters

- **asInputFiles** – the list of input files to mosaic
- **sOutputFile** – the name of the mosaic output file

Returns

the end status of the process

AsynchMosaic

```
public string Mosaic(List<string> asInputFiles, string sOutputFile, string sNoDataValue, string sInputIgnoreValue)
```

Asynch Mosaic with minimum parameters: input and output files.

Parameters

- **asInputFiles** – the list of input files to mosaic
- **sOutputFile** – the name of the mosaic output file
- **sNoDataValue** – the value to use in output as no data
- **sInputIgnoreValue** – the value to use as input no data

Returns

the end status of the process

AsynchMosaic

```
public string Mosaic(List<string> asInputFiles, string sOutputFile, string sNoDataValue, string sInputIgnoreValue,  
                    double dPixelSizeX, double dPixelSizeY)
```

Asynch Mosaic with minimum parameters: input and output files.

Parameters

- **asInputFiles** – the list of input files to mosaic
- **sOutputFile** – the name of the mosaic output file
- **sNoDataValue** – the value to use in output as no data
- **sInputIgnoreValue** – the value to use as input no data
- **dPixelSizeX** – the X Pixel Size
- **dPixelSizeY** – the Y Pixel Size

Returns

the end status of the process

SearchEOImages

```
public List<QueryResult> SearchEOImages(string sPlatform, string sDateFrom, string sDateTo, Double dULLat,  
                                         Double dULLon, Double dLRLat, Double dLRLon, string  
                                         sProductType, int iOrbitNumber, string sSensorOperationalMode,  
                                         string sCloudCoverage)
```

Search EO-Images

Parameters

- **sPlatform** – the Satellite Platform. Accepts “S1”, “S2”, “S3”, “S5P”, “ENVI”, “L8”, “VIIRS”, “ERA5”
- **sDateFrom** – the Starting date in format “YYYY-MM-DD”
- **sDateTo** – the End date in format “YYYY-MM-DD”
- **dULLat** – Upper Left Lat Coordinate. Can be null.
- **dULLon** – Upper Left Lon Coordinate. Can be null.
- **dLRLat** – Lower Right Lat Coordinate. Can be null.
- **dLRLon** – Lower Right Lon Coordinate. Can be null.
- **sProductType** – the Product Type. If Platform = “S1” -> Accepts “SLC”, “GRD”, “OCN”. If Platform = “S2” -> Accepts “S2MSI1C”, “S2MSI2Ap”, “S2MSI2A”. Can be null.
- **iOrbitNumber** – the Sentinel Orbit Number. Can be null.
- **sSensorOperationalMode** – the Sensor Operational Mode. ONLY for S1. Accepts -> “SM”, “IW”, “EW”, “WV”. Can be null. Ignored for Platform “S2”
- **sCloudCoverage** – the Cloud Coverage. Sample syntax: [0 TO 9.4]

Returns

the list of the available products

GetFoundProductName

public string **GetFoundProductName**(QueryResult *oProduct*)

Get the name of a Product found by searchEOImage.

Parameters

- **oProduct** – the Product as returned by searchEOImage

Returns

the name of the product

GetFoundProductName

public string **GetFoundProductName**(Dictionary<string, object> *oProduct*)

Get the name of a Product found by searchEOImage.

Parameters

- **oProduct** – the JSON Dictionary Product as returned by searchEOImage

Returns

the name of the product

GetFoundProductLink

public string **GetFoundProductLink**(QueryResult *oProduct*)

Get the direct download link of a Product found by searchEOImage.

Parameters

- **oProduct** – the Product as returned by searchEOImage

Returns

the link of the product

GetFoundProductLink

public string **GetFoundProductLink**(Dictionary<string, object> *oProduct*)

Get the direct download link of a Product found by searchEOImage.

Parameters

- **oProduct** – the JSON Dictionary Product as returned by searchEOImage

Returns

the link of the product

GetFoundProductFootprint

public string **GetFoundProductFootprint**(QueryResult *oProduct*)

Get the footprint of a Product found by searchEOImage.

Parameters

- **oProduct** – the Product as returned by searchEOImage

Returns

the footprint of the product

GetFoundProductFootprint

public string **GetFoundProductFootprint**(Dictionary<string, object> *oProduct*)

Get the footprint of a Product found by searchEOImage.

Parameters

- **oProduct** – the JSON Dictionary Product as returned by searchEOImage

Returns

the footprint of the product

AsynchImportProduct

public string **AsynchImportProduct**(Dictionary<string, object> *oProduct*)

Asynchronously import a product.

Parameters

- **oProduct** – the product to be imported

Returns

the status of the Import process

AsynchImportProduct

public string **AsynchImportProduct**(Dictionary<string, object> *oProduct*, string *sProvider*)

Asynchronously import a product.

Parameters

- **oProduct** – the product to be imported
- **sProvider** – the provider of choice. If null, the default provider will be used

Returns

the status of the Import process

ImportProduct

public string **ImportProduct**(Dictionary<string, object> *oProduct*)

Import a Product from a Provider in WASDI.

Parameters

- **oProduct** – the product to be imported

Returns

the status of the Import process

ImportProduct

public string **ImportProduct**(QueryResult *oProduct*)

Import a Product from a Provider in WASDI.

Parameters

- **oProduct** – the product to be imported

Returns

the status of the Import process

ImportProduct

public string **ImportProduct**(string *sFileUrl*, string *sFileName*)

Import a Product from a Provider in WASDI.

Parameters

- **sFileUrl** – the Direct link of the product
- **sFileName** – the name of the file

Returns

the status of the Import process

ImportProduct

public string **ImportProduct**(string *sFileUrl*, string *sFileName*, string *sBoundingBox*)

Import a Product from a Provider in WASDI.

Parameters

- **sFileUrl** – the Direct link of the product
- **sFileName** – the name of the file
- **sBoundingBox** – the bounding box

Returns

the status of the Import process

ImportProduct

public string **ImportProduct**(string *sFileUrl*, string *sFileName*, string *sBoundingBox*, string *sProvider*)

Import a Product from a Provider in WASDI.

Parameters

- **sFileUrl** – the Direct link of the product
- **sFileName** – the name of the file
- **sBoundingBox** – the bounding box
- **sProvider** – the provider

Returns

the status of the Import process

AsynchImportProduct

public string **AsynchImportProduct**(string *sFileUrl*, string *sFileName*)

Import a Product from a Provider in WASDI asynchronously.

Parameters

- **sFileUrl** – the Direct link of the product
- **sFileName** – the name of the file

Returns

the status of the Import process

AsynchImportProduct

public string **AsynchImportProduct**(string *sFileUrl*, string *sFileName*, string *sBoundingBox*)

Import a Product from a Provider in WASDI asynchronously.

Parameters

- **sFileUrl** – the Direct link of the product
- **sFileName** – the name of the file
- **sBoundingBox** – the bounding box

Returns

the status of the Import process

AsynchImportProduct

public string **AsynchImportProduct**(string *sFileUrl*, string *sFileName*, string *sBoundingBox*, string *sProvider*)

Import a Product from a Provider in WASDI asynchronously.

Parameters

- **sFileUrl** – the Direct link of the product
- **sFileName** – the name of the file
- **sBoundingBox** – the bounding box

- **sProvider** – the provider

Returns

the status of the Import process

AsynchImportProductListWithMaps

```
public List<string> AsynchImportProductListWithMaps(List<Dictionary<string, object>>
                                                    aoProductsToImport)
```

Imports a list of product asynchronously.

Parameters

- **aoProductsToImport** – the list of products to import

Returns

a list of String containing the WASDI process ids of all the imports

AsynchImportProductList

```
public List<string> AsynchImportProductList(List<string> asProductsToImport, List<string> asNames)
```

Imports a list of product asynchronously.

Parameters

- **asProductsToImport** – the list of products to import
- **asNames** – the list of names

Returns

a list of String containing the WASDI process ids of all the imports

ImportProductListWithMaps

```
public List<string> ImportProductListWithMaps(List<Dictionary<string, object>> aoProductsToImport)
```

Imports a list of product.

Parameters

- **aoProductsToImport** – the list of products to import

Returns

a list of String containing the WASDI process ids of all the imports

ImportProductList

```
public List<string> ImportProductList(List<string> asProductsToImport, List<string> asNames)
```

Imports a list of product.

Parameters

- **asProductsToImport** – the list of products to import
- **asNames** – the list of names

Returns

a list of String containing the WASDI process ids of all the imports

Subset

```
public string Subset(string sInputFile, string sOutputFile, double dLatN, double dLonW, double dLatS, double dLonE)
```

Make a Subset (tile) of an input image in a specified Lat Lon Rectangle.

Parameters

- **sInputFile** – the name of the input file
- **sOutputFile** – the name of the output file
- **dLatN** – the Lat North Coordinate
- **dLonW** – the Lon West Coordinate
- **dLatS** – the Lat South Coordinate
- **dLonE** – the Lon East Coordinate

Returns

the status of the operation

MultiSubset

```
public string MultiSubset(string sInputFile, List<string> asOutputFiles, List<Double> adLatN, List<Double> adLonW, List<Double> adLatS, List<Double> adLonE)
```

Creates a Many Subsets from an image. MAX 10 TILES PER CALL. Assumes big tiff format by default.

Parameters

- **sInputFile** – the name of the input file
- **asOutputFiles** – the name of the output file
- **adLatN** – the list of Lat North Coordinates
- **adLonW** – the list of Lon West Coordinates
- **adLatS** – the list of Lat South Coordinates
- **adLonE** – the list of Lon East Coordinates

Returns

the status of the operation

MultiSubset

```
public string MultiSubset(string sInputFile, List<string> asOutputFiles, List<Double> adLatN, List<Double> adLonW, List<Double> adLatS, List<Double> adLonE, bool bBigTiff)
```

Creates a Many Subsets from an image. MAX 10 TILES PER CALL.

Parameters

- **sInputFile** – the name of the input file
- **asOutputFiles** – the name of the output file
- **adLatN** – the list of Lat North Coordinates
- **adLonW** – the list of Lon West Coordinates
- **adLatS** – the list of Lat South Coordinates

- **adLonE** – the list of Lon East Coordinates
- **bBigTiff** – flag indicating whether to use the bigtiff format, for files bigger than 4 GB

Returns

the status of the operation

AsynchMultiSubset

```
public string MultiSubset(string sInputFile, List<string> asOutputFiles, List<Double> adLatN, List<Double>
                        adLonW, List<Double> adLatS, List<Double> adLonE)
```

Asynchronous multisubset: creates a Many Subsets from an image. MAX 10 TILES PER CALL. Assumes big tiff format by default.

Parameters

- **sInputFile** – the name of the input file
- **asOutputFiles** – the name of the output file
- **adLatN** – the list of Lat North Coordinates
- **adLonW** – the list of Lon West Coordinates
- **adLatS** – the list of Lat South Coordinates
- **adLonE** – the list of Lon East Coordinates

Returns

the status of the operation

AsynchMultiSubset

```
public string MultiSubset(string sInputFile, List<string> asOutputFiles, List<Double> adLatN, List<Double>
                        adLonW, List<Double> adLatS, List<Double> adLonE, bool bBigTiff)
```

Asynchronous multisubset: creates a Many Subsets from an image. MAX 10 TILES PER CALL.

Parameters

- **sInputFile** – the name of the input file
- **asOutputFiles** – the name of the output file
- **adLatN** – the list of Lat North Coordinates
- **adLonW** – the list of Lon West Coordinates
- **adLatS** – the list of Lat South Coordinates
- **adLonE** – the list of Lon East Coordinates
- **bBigTiff** – flag indicating whether to use the bigtiff format, for files bigger than 4 GB

Returns

the status of the operation

ExecuteProcessor

public string **ExecuteProcessor**(string *sProcessorName*, Dictionary<string, object> *aoParams*)

Executes a synchronous process, i.e., runs the process and waits for it to complete.

Parameters

- **sProcessorName** – the name of the processor
- **aoParams** – the dictionary of params

Returns

the WASDI processor Id

AsynchExecuteProcessor

public string **AsynchExecuteProcessor**(string *sProcessorName*, Dictionary<string, object> *aoParams*)

Execute a WASDI processor in Asynch way.

Parameters

- **sProcessorName** – the name of the processor
- **aoParams** – the dictionary of params

Returns

the WASDI processor Id

ExecuteProcessor

public string **ExecuteProcessor**(string *sProcessorName*, string *sEncodedParams*)

Executes a synchronous process, i.e., runs the process and waits for it to complete.

Parameters

- **sProcessorName** – the name of the processor
- **sEncodedParams** – a JSON formatted string of parameters for the processor

Returns

the WASDI processor Id

AsynchExecuteProcessor

public string **AsynchExecuteProcessor**(string *sProcessorName*, string *sEncodedParams*)

Execute a WASDI processor in Asynch way.

Parameters

- **sProcessorName** – the name of the processor
- **sEncodedParams** – a JSON formatted string of parameters for the processor

Returns

the WASDI processor Id

DeleteProduct

public string **DeleteProduct**(string *sProduct*)

Delete a Product in the active Workspace.

Parameters

- **sProduct** – the product's name

Returns

the status of the operation

WasdiLog

public void **WasdiLog**(string *sLogRow*)

Write one row of Log.

Parameters

- **sLogRow** – the text to log

GetProcessorPath

public string **GetProcessorPath**()

Get the processor Path. The value should resemble the following path: C:/dev/WasdiLib.Client/bin/Debug/net6.0

Returns

the processor path

CreateWorkspace

public string **CreateWorkspace**(string *sWorkspaceName*)

Create a workspace using the provided name. Once the workspace is created, it is also opened.

Parameters

- **sWorkspaceName** – the name of the workspace

Returns

the Id of the newly created workspace or empty string in case of any issues

CreateWorkspace

public string **CreateWorkspace**(string *sWorkspaceName*, string *nodeCode*)

Create a workspace using the provided name on the indicated node. Once the workspace is created, it is also opened.

Parameters

- **sWorkspaceName** – the name of the workspace
- **nodeCode** – the node on which to create the workspace

Returns

the Id of the newly created workspace or empty string in case of any issues

DeleteWorkspace

public string **DeleteWorkspace**(string *workspaceId*)

Deletes the workspace given its Id.

Parameters

- **workspaceId** – the Id of the workspace

Returns

the Id of the workspace as a String if succesful, empty string otherwise

GetProcessWorkspacesByWorkspaceId

public List<ProcessWorkspace> **GetProcessWorkspacesByWorkspaceId**(string *workspaceId*)

Get the process workspaces by workspace id

Parameters

- **workspaceId** – the Id of the workspace

Returns

the list of process workspaces or an empty list in case of any issues

GetProcessesByWorkspaceAsListOfJson

public List<string> **GetProcessesByWorkspaceAsListOfJson**(int *iStartIndex*, Int32 *iEndIndex*, string *sStatus*, string *sOperationType*, string *sNamePattern*)

Get a paginated list of processes in the active workspace, each element of which is a JSON string.

Parameters

- **iStartIndex** – the start index of the process (0 by default is the last one)
- **iEndIndex** – the end index of the process (optional)
- **sStatus** – the status filter, null by default. Can be CREATED, RUNNING, STOPPED, DONE, ERROR, WAITING, READY
- **sOperationType** – the Operation Type Filter, null by default. Can be RUNPROCESSOR, RUNIDL, RUNMATLAB, INGEST, DOWNLOAD, GRAPH, DEPLOYPROCESSOR
- **sNamePattern** – the Name filter. The name meaning depends by the operation type, null by default. For RUNPROCESSOR, RUNIDL and RUNMATLAB is the name of the application

Returns

a list of process IDs

GetProcessesByWorkspace

```
public List<string> GetProcessesByWorkspaceAsListOfJson(int iStartIndex, Int32 iEndIndex, string sStatus,
string sOperationType, string sNamePattern)
```

Get a paginated list of processes in the active workspace.

Parameters

- **iStartIndex** – the start index of the process (0 by default is the last one)
- **iEndIndex** – the end index of the process (optional)
- **sStatus** – the status filter, null by default. Can be CREATED, RUNNING, STOPPED, DONE, ERROR, WAITING, READY
- **sOperationType** – the Operation Type Filter, null by default. Can be RUNPROCESSOR, RUNIDL, RUNMATLAB, INGEST, DOWNLOAD, GRAPH, DEPLOYPROCESSOR
- **sNamePattern** – the Name filter. The name meaning depends by the operation type, null by default. For RUNPROCESSOR, RUNIDL and RUNMATLAB is the name of the application

Returns

a list of process IDs

GetProcessorPayload

```
public Dictionary<string, object> GetProcessorPayload(string sProcessObjId)
```

Gets the processor payload as a dictionary.

Parameters

- **sProcessObjId** – the Id of the processor

Returns

the processor payload as a dictionary

GetProcessorPayloadAsJSON

```
public Dictionary<string, object> GetProcessorPayloadAsJSON(string sProcessObjId)
```

Retrieve the payload of a processor formatted as a JSON string.

Parameters

- **sProcessObjId** – the Id of the processor

Returns

the payload as a JSON string, or null if error occurred

GetProductBbox

public string **GetProductBbox**(string *sFileName*)

Get the product bounding box.

Parameters

- **sFileName** – the file name

Returns

the product bounding box

DownloadFile

public string **DownloadFile**(string *sFileName*)

Download a file on the local PC.

Parameters

- **sFileName** – the name of the file

Returns

the full path of the file

UploadFile

public bool **UploadFile**(string *sFileName*)

Uploads and ingest a file in WASDI.

Parameters

- **sFileName** – the name of the file to upload

Returns

True if the file was uploaded, False otherwise

Throws

- **Exception** – in case of any issues

CopyFileToSftp

public string **CopyFileToSftp**(string *sFileName*)

Copy a file from a workspace to the WASDI user's SFTP Folder in a synchronous way.

Parameters

- **sFileName** – the filename to move to the SFTP folder

Returns

the Process Id is synchronous execution, end status otherwise. An empty string is returned in case of failure

CopyFileToSftp

public string **CopyFileToSftp**(string *sFileName*, string *sRelativePath*)

Copy a file from a workspace to the WASDI user's SFTP Folder in a synchronous way.

Parameters

- **sFileName** – the filename to move to the SFTP folder
- **sRelativePath** – the relative path in the SFTP root

Returns

the Process Id is synchronous execution, end status otherwise. An empty string is returned in case of failure

AsynchCopyFileToSftp

public string **CopyFileToSftp**(string *sFileName*)

Copy a file from a workspace to the WASDI user's SFTP Folder in a asynchronous way.

Parameters

- **sFileName** – the filename to move to the SFTP folder

Returns

the Process Id is asynchronous execution, end status otherwise. An empty string is returned in case of failure

AsynchCopyFileToSftp

public string **CopyFileToSftp**(string *sFileName*, string *sRelativePath*)

Copy a file from a workspace to the WASDI user's SFTP Folder in a asynchronous way.

Parameters

- **sFileName** – the filename to move to the SFTP folder
- **sRelativePath** – the relative path in the SFTP root

Returns

the Process Id is asynchronous execution, end status otherwise. An empty string is returned in case of failure

SetSubPid

public string **SetSubPid**(string *sProcessId*, int *iSubPid*)

Sets the sub pid.

Parameters

- **sProcessId** – the process Id
- **iSubPid** – the subPid of the process

Returns

the updated status of the processs

PrintStatus

```
public void PrintStatus()
```

Print the status information of the Wasdi application.

5.2 Java WasdiLib

```
public class WasdiLib
```

5.2.1 Fields

```
s_oMapper
```

```
protected static ObjectMapper s_oMapper
```

5.2.2 Constructors

```
WasdiLib
```

```
public WasdiLib()
```

Self constructor. If there is a config file initializes the class members

5.2.3 Methods

```
addFileToWASDI
```

```
public String addFileToWASDI(String sFileName)
```

Ingest a new file in the Active WASDI Workspace waiting for the result The method takes a file saved in the workspace root (see `getSaveFilePath`) not already added to the WS To work be sure that the file is on the server

Parameters

- **sFileName** – Name of the file to add

Returns

Output state of the ingestion process

```
addParam
```

```
public void addParam(String sKey, String sParam)
```

Add Param

Parameters

- **sKey** –
- **sParam** –

asynchAddFileToWASDI

public **String** **asynchAddFileToWASDI**(**String** *sFileName*)

Ingest a new file in the Active WASDI Workspace in an asynch way The method takes a file saved in the workspace root (see `getSaveFilePath`) not already added to the WS To work be sure that the file is on the server

Parameters

- **sFileName** – Name of the file to add

Returns

Process Id of the ingestion process

asynchExecuteProcessor

public **String** **asynchExecuteProcessor**(**String** *sProcessorName*, **HashMap**<**String**, **Object**> *aoParams*)

Execute a WASDI processor in Asynch way

Parameters

- **sProcessorName** – Processor Name
- **aoParams** – Dictionary of Params

Returns

ProcessWorkspace Id

asynchExecuteProcessor

public **String** **asynchExecuteProcessor**(**String** *sProcessorName*, **String** *sEncodedParams*)

Execute a WASDI processor in Asynch way

Parameters

- **sProcessorName** – Processor Name
- **sEncodedParams** – Already JSON Encoded Params

Returns

ProcessWorkspace Id

asynchExecuteWorkflow

public **String** **asynchExecuteWorkflow**(**String**[] *asInputFileName*, **String**[] *asOutputFileName*, **String** *sWorkflowName*)

Executes a WASDI SNAP Workflow in a asynch mode

Parameters

- **sInputFileName** –
- **sOutputFileName** –
- **sWorkflowName** –

Returns

Workflow Process Id if every thing is ok, “ if there was any problem

asynchMosaic

```
public String asynchMosaic(List<String> asInputFiles, String sOutputFile)
```

Asynch Mosaic with minimum parameters

Parameters

- **asInputFiles** – List of input files to mosaic
- **sOutputFile** – Name of the mosaic output file

Returns

Process id

asynchMosaic

```
public String asynchMosaic(List<String> asInputFiles, String sOutputFile, String sNoDataValue, String  
sInputIgnoreValue)
```

Asynch Mosaic with also Bands Parameters

Parameters

- **asInputFiles** – List of input files to mosaic
- **sOutputFile** – Name of the mosaic output file

Returns

Process id

asynchMosaic

```
public String asynchMosaic(List<String> asInputFiles, String sOutputFile, String sNoDataValue, String  
sInputIgnoreValue, List<String> asBands)
```

Asynch Mosaic with also Bands Parameters

Parameters

- **asInputFiles** – List of input files to mosaic
- **sOutputFile** – Name of the mosaic output file
- **asBands** – List of the bands to use for the mosaic

Returns

Process id

asynchMosaic

```
public String asynchMosaic(List<String> asInputFiles, String sOutputFile, String sNoDataValue, String  
sInputIgnoreValue, List<String> asBands, double dPixelSizeX, double dPixelSizeY)
```

Asynch Mosaic with also Pixel Size Parameters

Parameters

- **asInputFiles** – List of input files to mosaic
- **sOutputFile** – Name of the mosaic output file
- **asBands** – List of the bands to use for the mosaic

- **dPixelSizeX** – X Pixel Size
- **dPixelSizeY** – Y Pixel Size

Returns

Process id

asynchMosaic

```
public String asynchMosaic(List<String> asInputFiles, String sOutputFile, String sNoDataValue, String
                           sInputIgnoreValue, List<String> asBands, double dPixelSizeX, double dPixelSizeY,
                           String sCrs)
```

Asynch Mosaic with also CRS Input

Parameters

- **asInputFiles** – List of input files to mosaic
- **sOutputFile** – Name of the mosaic output file
- **asBands** – List of the bands to use for the mosaic
- **dPixelSizeX** – X Pixel Size
- **dPixelSizeY** – Y Pixel Size
- **sCrs** – WKT of the CRS to use

Returns

Process id

asynchMosaic

```
public String asynchMosaic(List<String> asInputFiles, String sOutputFile, String sNoDataValue, String
                           sInputIgnoreValue, List<String> asBands, double dPixelSizeX, double dPixelSizeY,
                           String sCrs, double dSouthBound, double dNorthBound, double dEastBound, double
                           dWestBound, String sOverlappingMethod, boolean bShowSourceProducts, String
                           sElevationModelName, String sResamplingName, boolean bUpdateMode, boolean
                           bNativeResolution, String sCombine)
```

Asynch Mosaic with all the input parameters

Parameters

- **asInputFiles** – List of input files to mosaic
- **sOutputFile** – Name of the mosaic output file
- **asBands** – List of the bands to use for the mosaic
- **dPixelSizeX** – X Pixel Size
- **dPixelSizeY** – Y Pixel Size
- **sCrs** – WKT of the CRS to use
- **dSouthBound** – South Bound
- **dNorthBound** – North Bound
- **dEastBound** – East Bound
- **dWestBound** – West Bound

- **sOverlappingMethod** – Overlapping Method
- **bShowSourceProducts** – Show Source Products Flag
- **sElevationModelName** – DEM Model Name
- **sResamplingName** – Resampling Method Name
- **bUpdateMode** – Update Mode Flag
- **bNativeResolution** – Native Resolution Flag
- **sCombine** – Combine verb

Returns

Process id

checkSession

public **String** **checkSession**(**String** *sSessionID*)

Call CheckSession API

Parameters

- **sSessionID** – actual session Id

Returns

Session Id or “” if there are problems

copyStream

protected void **copyStream**(**InputStream** *oInputStream*, **OutputStream** *oOutputStream*)

copyStreamAndClose

protected void **copyStreamAndClose**(**InputStream** *oInputStream*, **OutputStream** *oOutputStream*)

Copy Input Stream in Output Stream

Parameters

- **oInputStream** –
- **oOutputStream** –

Throws

- **IOException** –

deleteProduct

public **String** **deleteProduct**(**String** *sProduct*)

Delete a Product in the active Workspace

Parameters

- **sProduct** –

downloadFile

protected **String** **downloadFile**(*String sFileName*)

Download a file on the local PC

Parameters

- **sFileName** – File Name

Returns

Full Path

executeWorkflow

public **String** **executeWorkflow**(*String[] asInputFileName, String[] asOutputFileName, String sWorkflowName*)

Executes a WASDI SNAP Workflow waiting for the process to finish

Parameters

- **sInputFileName** –
- **sOutputFileName** –
- **sWorkflowName** –

Returns

output status of the Workflow Process

getActiveWorkspace

public **String** **getActiveWorkspace**()

Get Active Workspace

getBasePath

public **String** **getBasePath**()

Set Base Path

getBaseUrl

public **String** **getBaseUrl**()

Get Base Url

getDownloadActive

public **Boolean** **getDownloadActive**()

Get Download Active flag

getFoundProductLink

public **String** **getFoundProductLink**(Map<String, Object> *oProduct*)

Get the direct download link of a Product found by searchEOImage

Parameters

- **oProduct** – JSON Dictionary Product as returned by searchEOImage

Returns

Name of the product

getFoundProductName

public **String** **getFoundProductName**(Map<String, Object> *oProduct*)

Get the name of a Product found by searchEOImage

Parameters

- **oProduct** – JSON Dictionary Product as returned by searchEOImage

Returns

Name of the product

getFullProductPath

public **String** **getFullProductPath**(String *sProductName*)

Get the full local path of a product given the product name. Use the output of this API to open the file

Parameters

- **sProductName** – Product Name

Returns

Product Full Path as a String ready to open file

getIsOnServer

public **Boolean** **getIsOnServer**()

Get is on server flag

getMyProcId

public **String** **getMyProcId**()

Get my own Process Id

getParam

```
public String getParam(String sKey)
```

Get Param

Parameters

- **sKey** –

getParametersFilePath

```
public String getParametersFilePath()
```

Get Parameters file path

Returns

parameters file path

getParams

```
public HashMap<String, String> getParams()
```

Get Params HashMap

Returns

Params Dictionary

getPassword

```
public String getPassword()
```

Get Password

getPath

```
public String getPath(String sProductName)
```

Get the local path of a file

Parameters

- **sProductName** – Name of the file

Returns

Full local path

getProcessStatus

```
public String getProcessStatus(String sProcessId)
```

Get WASDI Process Status

Parameters

- **sProcessId** – Process Id

Returns

Process Status as a String: CREATED, RUNNING, STOPPED, DONE, ERROR, WAITING, READY

getProcessorPath

public **String** **getProcessorPath()**

Get the processor Path

getProductsByActiveWorkspace

public **List<String>** **getProductsByActiveWorkspace()**

Get a List of the products in the active Workspace

Returns

List of Strings representing the product names

getProductsByWorkspace

public **List<String>** **getProductsByWorkspace(String sWorkspaceName)**

Get a List of the products in a Workspace

Parameters

- **sWorkspaceName** – Workspace Name

Returns

List of Strings representing the product names

getSavePath

public **String** **getSavePath()**

Get the local Save Path to use to save custom generated files

Returns

Local Path to use to save a custom generated file

getSessionId

public **String** **getSessionId()**

Get Session Id

getStandardHeaders

protected **HashMap<String, String>** **getStandardHeaders()**

Get the standard headers for a WASDI call

getStreamingHeaders

```
protected HashMap<String, String> getStreamingHeaders()
```

Get the headers for a Streaming POST call

getUser

```
public String getUser()
```

Get User

Returns

User

getVerbose

```
public Boolean getVerbose()
```

Get Verbose Flag

getWorkflows

```
public List<Map<String, Object>> getWorkflows()
```

Get the list of Workflows for the user Return None if there is any error Return an array of WASI Workspace JSON Objects if everything is ok: { "description":STRING, "name": STRING, "workflowId": STRING }

getWorkspaceBaseUrl

```
public String getWorkspaceBaseUrl()
```

getWorkspaceIdByName

```
public String getWorkspaceIdByName(String sWorkspaceName)
```

Get Id of a Workspace from the name Return the WorkspaceId as a String, "" if there is any error

Parameters

- **sWorkspaceName** – Workspace Name

Returns

Workspace Id if found, "" if there is any error

getWorkspaceOwnerByName

```
public String getWorkspaceOwnerByName(String sWorkspaceName)
```

Get User Id of the owner of a Workspace from the name Return the userId as a String, "" if there is any error

Parameters

- **sWorkspaceName** – Workspace Name

Returns

User Id if found, "" if there is any error

getWorkspaceOwnerByWSId

public **String** **getWorkspaceOwnerByWSId**(**String** *sWorkspaceId*)

Get userId of the owner of a Workspace from the workspace Id Return the userId as a String, "" if there is any error

Parameters

- **WorkspaceId** – Workspace Id

Returns

userId if found, "" if there is any error

getWorkspaces

public **List**<**Map**<**String**, **Object**>> **getWorkspaces**()

get the list of workspaces of the logged user

Returns

List of Workspace as JSON representation

httpGet

public **String** **httpGet**(**String** *sUrl*, **Map**<**String**, **String**> *asHeaders*)

Http get Method Helper

Parameters

- **sUrl** – Url to call
- **asHeaders** – Headers Dictionary

Returns

Server response

httpPost

public **String** **httpPost**(**String** *sUrl*, **String** *sPayload*, **Map**<**String**, **String**> *asHeaders*)

Standard http post utility function

Parameters

- **sUrl** – url to call
- **sPayload** – payload of the post
- **asHeaders** – headers dictionary

Returns

server response

importProduct

```
public String importProduct(Map<String, Object> oProduct)
```

Import a Product from a Provider in WASDI.

Parameters

- **oProduct** – Product Map JSON representation as returned by searchEOImage

Returns

status of the Import process

importProduct

```
public String importProduct(String sFileUrl)
```

Import a Product from a Provider in WASDI.

Parameters

- **sFileUrl** – Direct link of the product

Returns

status of the Import process

importProduct

```
public String importProduct(String sFileUrl, String sBoundingBox)
```

Import a Product from a Provider in WASDI.

Parameters

- **sFileUrl** – Direct link of the product
- **sBoundingBox** – Bounding Box of the product

Returns

status of the Import process

init

```
public Boolean init(String sConfigFilePath)
```

Init the WASDI Library starting from a configuration file

Parameters

- **sConfigFilePath** – full path of the configuration file

Returns

True if the system is initialized, False if there is any error

init

```
public Boolean init()
```

internalAddFileToWASDI

```
protected String internalAddFileToWASDI(String sFileName, Boolean bAsynch)
```

Private version of the add file to wasdi function. Adds a generated file to current open workspace

Parameters

- **sFileName** – File Name to add to the open workspace
- **bAsynch** – true if the process has to be asynch, false to wait for the result

internalExecuteWorkflow

```
protected String internalExecuteWorkflow(String[] asInputFileNames, String[] asOutputFileNames, String  
                                           sWorkflowName, Boolean bAsynch)
```

Internal execute workflow

Parameters

- **asInputFileNames** –
- **asOutputFileNames** –
- **sWorkflowName** –
- **bAsynch** – true if asynch, false for synch

Returns

if Asynch, the process Id else the output status of the workflow process

internalInit

```
public Boolean internalInit()
```

Call this after base parameters settings to init the system Needed at least: Base Path User Password or SessionId

internalMosaic

```
protected String internalMosaic(boolean bAsynch, List<String> asInputFiles, String sOutputFile)
```

Protected Mosaic with minimum parameters

Parameters

- **bAsynch** – True to return after the triggering, False to wait the process to finish
- **asInputFiles** – List of input files to mosaic
- **sOutputFile** – Name of the mosaic output file

Returns

Process id or end status of the process

internalMosaic

protected `String internalMosaic`(boolean *bAsynch*, `List<String>` *asInputFiles*, `String` *sOutputFile*, `String` *sNoDataValue*, `String` *sInputIgnoreValue*)

Protected Mosaic with also nodata value parameters

Parameters

- **bAsynch** – True to return after the triggering, False to wait the process to finish
- **asInputFiles** – List of input files to mosaic
- **sOutputFile** – Name of the mosaic output file
- **sNoDataValue** – Value to use in output as no data
- **sInputIgnoreValue** – Value to use as input no data

Returns

Process id or end status of the process

internalMosaic

protected `String internalMosaic`(boolean *bAsynch*, `List<String>` *asInputFiles*, `String` *sOutputFile*, `String` *sNoDataValue*, `String` *sInputIgnoreValue*, `List<String>` *asBands*)

Protected Mosaic with also Bands Parameters

Parameters

- **bAsynch** – True to return after the triggering, False to wait the process to finish
- **asInputFiles** – List of input files to mosaic
- **sOutputFile** – Name of the mosaic output file
- **asBands** – List of the bands to use for the mosaic

Returns

Process id or end status of the process

internalMosaic

protected `String internalMosaic`(boolean *bAsynch*, `List<String>` *asInputFiles*, `String` *sOutputFile*, `String` *sNoDataValue*, `String` *sInputIgnoreValue*, `List<String>` *asBands*, double *dPixelSizeX*, double *dPixelSizeY*)

Protected Mosaic with also Pixel Size Parameters

Parameters

- **bAsynch** – True to return after the triggering, False to wait the process to finish
- **asInputFiles** – List of input files to mosaic
- **sOutputFile** – Name of the mosaic output file
- **asBands** – List of the bands to use for the mosaic
- **dPixelSizeX** – X Pixel Size
- **dPixelSizeY** – Y Pixel Size

Returns

Process id or end status of the process

internalMosaic

```
protected String internalMosaic(boolean bAsynch, List<String> asInputFiles, String sOutputFile, String  
sNoDataValue, String sInputIgnoreValue, List<String> asBands, double  
dPixelSizeX, double dPixelSizeY, String sCrS)
```

Protected Mosaic with also CRS Input

Parameters

- **bAsynch** – True to return after the triggering, False to wait the process to finish
- **asInputFiles** – List of input files to mosaic
- **sOutputFile** – Name of the mosaic output file
- **asBands** – List of the bands to use for the mosaic
- **dPixelSizeX** – X Pixel Size
- **dPixelSizeY** – Y Pixel Size
- **sCrS** – WKT of the CRS to use

Returns

Process id or end status of the process

internalMosaic

```
protected String internalMosaic(boolean bAsynch, List<String> asInputFiles, String sOutputFile, String  
sNoDataValue, String sInputIgnoreValue, List<String> asBands, double  
dPixelSizeX, double dPixelSizeY, String sCrS, double dSouthBound, double  
dNorthBound, double dEastBound, double dWestBound, String  
sOverlappingMethod, boolean bShowSourceProducts, String  
sElevationModelName, String sResamplingName, boolean bUpdateMode,  
boolean bNativeResolution, String sCombine)
```

Protected Mosaic with all the input parameters

Parameters

- **bAsynch** – True to return after the triggering, False to wait the process to finish
- **asInputFiles** – List of input files to mosaic
- **sOutputFile** – Name of the mosaic output file
- **asBands** – List of the bands to use for the mosaic
- **dPixelSizeX** – X Pixel Size
- **dPixelSizeY** – Y Pixel Size
- **sCrS** – WKT of the CRS to use
- **dSouthBound** – South Bound
- **dNorthBound** – North Bound
- **dEastBound** – East Bound

- **dWestBound** – West Bound
- **sOverlappingMethod** – Overlapping Method
- **bShowSourceProducts** – Show Source Products Flag
- **sElevationModelName** – DEM Model Name
- **sResamplingName** – Resampling Method Name
- **bUpdateMode** – Update Mode Flag
- **bNativeResolution** – Native Resolution Flag
- **sCombine** – Combine verb

Returns

Process id or end status of the process

log

protected void **log**(String sLog)

Log

Parameters

- **sLog** – Log row

login

public String **login**(String sUser, String sPassword)

Call Login API

Parameters

- **sUser** –
- **sPassword** –

mosaic

public String **mosaic**(List<String> asInputFiles, String sOutputFile)

Mosaic with minimum parameters: input and output files

Parameters

- **asInputFiles** – List of input files to mosaic
- **sOutputFile** – Name of the mosaic output file

Returns

End status of the process

mosaic

```
public String mosaic(List<String> asInputFiles, String sOutputFile, String sNoDataValue, String  
                    sInputIgnoreValue)
```

Mosaic with also NoData Parameters

Parameters

- **asInputFiles** – List of input files to mosaic
- **sOutputFile** – Name of the mosaic output file
- **sNoDataValue** – Value to use in output as no data
- **sInputIgnoreValue** – Value to use as input no data

Returns

End status of the process

mosaic

```
public String mosaic(List<String> asInputFiles, String sOutputFile, String sNoDataValue, String  
                    sInputIgnoreValue, List<String> asBands)
```

Mosaic with also Bands Parameters

Parameters

- **asInputFiles** – List of input files to mosaic
- **sOutputFile** – Name of the mosaic output file
- **asBands** – List of the bands to use for the mosaic

Returns

End status of the process

mosaic

```
public String mosaic(List<String> asInputFiles, String sOutputFile, String sNoDataValue, String  
                    sInputIgnoreValue, List<String> asBands, double dPixelSizeX, double dPixelSizeY)
```

Mosaic with also Pixel Size Parameters

Parameters

- **asInputFiles** – List of input files to mosaic
- **sOutputFile** – Name of the mosaic output file
- **asBands** – List of the bands to use for the mosaic
- **dPixelSizeX** – X Pixel Size
- **dPixelSizeY** – Y Pixel Size

Returns

End status of the process

mosaic

```
public String mosaic(List<String> asInputFiles, String sOutputFile, String sNoDataValue, String sInputIgnoreValue,
                    List<String> asBands, double dPixelSizeX, double dPixelSizeY, String sCrS)
```

Mosaic with also CRS Input

Parameters

- **asInputFiles** – List of input files to mosaic
- **sOutputFile** – Name of the mosaic output file
- **asBands** – List of the bands to use for the mosaic
- **dPixelSizeX** – X Pixel Size
- **dPixelSizeY** – Y Pixel Size
- **sCrS** – WKT of the CRS to use

Returns

End status of the process

mosaic

```
public String mosaic(List<String> asInputFiles, String sOutputFile, String sNoDataValue, String sInputIgnoreValue,
                    List<String> asBands, double dPixelSizeX, double dPixelSizeY, String sCrS, double
                    dSouthBound, double dNorthBound, double dEastBound, double dWestBound, String
                    sOverlappingMethod, boolean bShowSourceProducts, String sElevationModelName, String
                    sResamplingName, boolean bUpdateMode, boolean bNativeResolution, String sCombine)
```

Mosaic with all the input parameters

Parameters

- **asInputFiles** – List of input files to mosaic
- **sOutputFile** – Name of the mosaic output file
- **asBands** – List of the bands to use for the mosaic
- **dPixelSizeX** – X Pixel Size
- **dPixelSizeY** – Y Pixel Size
- **sCrS** – WKT of the CRS to use
- **dSouthBound** – South Bound
- **dNorthBound** – North Bound
- **dEastBound** – East Bound
- **dWestBound** – West Bound
- **sOverlappingMethod** – Overlapping Method
- **bShowSourceProducts** – Show Source Products Flag
- **sElevationModelName** – DEM Model Name
- **sResamplingName** – Resampling Method Name
- **bUpdateMode** – Update Mode Flag
- **bNativeResolution** – Native Resolution Flag

- **sCombine** – Combine verb

Returns

End status of the process

openWorkspace

```
public String openWorkspace(String sWorkspaceName)
```

Open a workspace

Parameters

- **sWorkspaceName** – Workspace name to open

Returns

WorkspaceId as a String, ‘’ if there is any error

refreshParameters

```
public void refreshParameters()
```

Refresh Parameters reading again the file

searchEOImages

```
public List<Map<String, Object>> searchEOImages(String sPlatform, String sDateFrom, String sDateTo, Double dULLat, Double dULLon, Double dLRLat, Double dLRLon, String sProductType, Integer iOrbitNumber, String sSensorOperationalMode, String sCloudCoverage)
```

Search EO-Images

Parameters

- **sPlatform** – Satellite Platform. Accepts “S1”, “S2”
- **sDateFrom** – Starting date in format “YYYY-MM-DD”
- **sDateTo** – End date in format “YYYY-MM-DD”
- **dULLat** – Upper Left Lat Coordinate. Can be null.
- **dULLon** – Upper Left Lon Coordinate. Can be null.
- **dLRLat** – Lower Right Lat Coordinate. Can be null.
- **dLRLon** – Lower Right Lon Coordinate. Can be null.
- **sProductType** – Product Type. If Platform = “S1” -> Accepts “SLC”, “GRD”, “OCN”. If Platform = “S2” -> Accepts “S2MSI1C”, “S2MSI2Ap”, “S2MSI2A”. Can be null.
- **iOrbitNumber** – Sentinel Orbit Number. Can be null.
- **sSensorOperationalMode** – Sensor Operational Mode. ONLY for S1. Accepts -> “SM”, “IW”, “EW”, “WV”. Can be null. Ignored for Platform “S2”
- **sCloudCoverage** – Cloud Coverage. Sample syntax: [0 TO 9.4]

Returns

List of the available products as a LIST of Dictionary representing JSON Object: { footprint = id = link = provider = Size = title = properties = < Another JSON Object containing other product-specific info > }

setActiveWorkspace

public void **setActiveWorkspace**(String *sActiveWorkspace*)

Set Active Workspace

Parameters

- **sActiveWorkspace** –

setBasePath

public void **setBasePath**(String *sBasePath*)

Get Base Path

Parameters

- **sBasePath** –

setBaseUrl

public void **setBaseUrl**(String *sBaseUrl*)

Set Base URL

Parameters

- **sBaseUrl** –

setDownloadActive

public void **setDownloadActive**(Boolean *bDownloadActive*)

Set Download Active Flag

Parameters

- **bDownloadActive** –

setIsOnServer

public void **setIsOnServer**(Boolean *bIsOnServer*)

Set is on server flag

Parameters

- **bIsOnServer** –

setMyProcId

public void **setMyProcId**(String *sMyProcId*)

Set My own process ID

Parameters

- **m_sMyProcId** –

setParametersFilePath

public void **setParametersFilePath**(String *sParametersFilePath*)

Set Parameters file path

Parameters

- **sParametersFilePath** – parameters file path

setPassword

public void **setPassword**(String *sPassword*)

Set Password

Parameters

- **sPassword** –

setProcessPayload

public String **setProcessPayload**(String *sProcessId*, String *sData*)

Adds output payload to a process

Parameters

- **sProcessId** –
- **sData** –

setSessionId

public void **setSessionId**(String *sSessionId*)

Set Session Id

Parameters

- **sSessionId** –

setUser

public void **setUser**(String *sUser*)

Set User

Parameters

- **sUser** – User

setVerbose

```
public void setVerbose(Boolean bVerbose)
```

Set Verbose flag

Parameters

- **bVerbose** –

setWorkspaceBaseUrl

```
public void setWorkspaceBaseUrl(String m_sWorkspaceBaseUrl)
```

subset

```
public String subset(String sInputFile, String sOutputFile, double dLatN, double dLonW, double dLatS, double dLonE)
```

Make a Subset (tile) of an input image in a specified Lat Lon Rectangle

Parameters

- **sInputFile** – Name of the input file
- **sOutputFile** – Name of the output file
- **dLatN** – Lat North Coordinate
- **dLonW** – Lon West Coordinate
- **dLatS** – Lat South Coordinate
- **dLonE** – Lon East Coordinate

Returns

Status of the operation

updateProcessStatus

```
public String updateProcessStatus(String sProcessId, String sStatus, int iPerc)
```

Update the status of a process

Parameters

- **sProcessId** – Process Id
- **sStatus** – Status to set
- **iPerc** – Progress in %

Returns

updated status as a String or “” if there was any problem

updateProgressPerc

public **String** **updateProgressPerc**(int *iPerc*)

Update the status of a process

Parameters

- **sProcessId** – Process Id
- **sStatus** – Status to set
- **iPerc** – Progress in %

Returns

updated status as a String or “” if there was any problem

updateStatus

public **String** **updateStatus**(**String** *sStatus*)

Update the status of the current process

Parameters

- **sStatus** – Status to set
- **iPerc** – Progress in %

Returns

updated status as a String or “” if there was any problem

updateStatus

public **String** **updateStatus**(**String** *sStatus*, int *iPerc*)

Update the status of the current process

Parameters

- **sStatus** – Status to set
- **iPerc** – Progress in %

Returns

updated status as a String or “” if there was any problem

uploadFile

public void **uploadFile**(**String** *sFileName*)

Parameters

- **sFileName** –

waitForResume

protected void **waitForResume**()

Wait for a process to finish

Parameters

- **sProcessId** –

waitProcess

public **String** **waitProcess**(**String** *sProcessId*)

Wait for a process to finish

Parameters

- **sProcessId** –

5.3 Matlab WasdiLib

5.3.1 Methods

startWasdi

matlabwasdilb.**startWasdi**(*config_path*)

Initialize the Wasdi object

Parameters

config_path – The path to be used to import the configuration.

Returns

Wasdi

The object to be used to invoke all the following call to Wasdi services

wAddFileToWASDI

matlabwasdilb.**wAddFileToWASDI**(*Wasdi*, *sFileName*)

Ingest a new file in the Active WASDI Workspace waiting for the result The method takes a file saved in the workspace root (see `getSaveFilePath`) not already added to the WS o work be sure that the file is on the server Syntax `sStatus =wAddFileToWASDI(Wasdi, sFileName);`

Parameters

- **Wasdi** – Wasdi object created after the wasdilb call
- **sFileName** – Name of the file to add

Returns

sStatus

Status of the Ingest Process as a String: CREATED, RUNNING, STOPPED, DONE, ERROR

wAddParam

`matlabwasdilib.wAddParam(Wasdi, sKey, sValue)`

Adds a parameter to current processor Syntax `wAddParam(Wasdi, sKey, sValue)`

Parameters

- **Wasdi** – Wasdi object created after the wasdilib call
- **sKey** – a string to be used as key for the param
- **sValue** – a string to be used as a value

wAsynchAddFileToWASDI

`matlabwasdilib.wAsynchAddFileToWASDI(Wasdi, sFileName)`

Ingest a new file in the Active WASDI Workspace WITHOUT waiting for the result The method takes a file saved in the workspace root (see `getSaveFilePath`) not already added to the WS If the file is not present in the WASDI cloud workspace, it will be automatically uploaded if the config AUTOUPLOAD flag is true (default) Syntax `sStatus =wAsynchAddFileToWASDI(Wasdi, sFileName);`

Parameters

- **Wasdi** – Wasdi object created after the wasdilib call
- **sFileName** – Name of the file to add

Returns

sProcessId

Process Id of the WASDI Ingest operation on the server. Can be used as input to the `wWaitProcess` method or `wGetProcessStatus` methods to check the execution.

wAsynchCopyFileToSftp

`matlabwasdilib.wAsynchCopyFileToSftp(Wasdi, sFileName, sRelativePath)`

Copy file to SFTP folder, asynchronous version Syntax `wAsynchCopyFileToSftp(Wasdi, sFileName, sRelativePath)`

Parameters

- **Wasdi** – Wasdi object created after the wasdilib call
- **sFileName** – a string containing the file name
- **sRelativePath** – a string containinng the relative path

wAsynchExecuteProcessor

`matlabwasdilib.wAsynchExecuteProcessor(Wasdi, sProcessorName, asParams)`

Execute a WASDI processor asynchronously Syntax `sStatus=wAsynchExecuteProcessor(Wasdi, sProcessorName, asParams)`

Parameters

- **Wasdi** – Wasdi object created after the wasdilib call
- **sProcessorName** – Processor Name
- **asParams** – Processor parameters, as a key/value dictionary

Returns

sProcessId

process workspace id. It can be used as input to the `wWaitProcess` method or `wGetProcessStatus` methods to check the execution.

wAsynchExecuteWorkflow

`matlabwasdilib.wAsynchExecuteWorkflow(Wasdi, sWorkflow, asInputFiles, asOutputFiles)`

Executes a SNAP workflow in Asynch mode. The workflow has to be uploaded in WASDI: it can be public or private of a user. If it is private, it must be triggered from the owner. Syntax `sProcessId = wExecuteWorkflow(Wasdi, sWorkflow, asInputFiles, asOutputFiles);`

Parameters

- **Wasdi** – Wasdi object created after the wasdilib call
- **sWorkflow** – Name of the workflow
- **asInputFiles** – array of strings with the name of the input files. Must be one file for each Read Node of the workflow, in the exact order
- **asOutputFiles** – array of strings with the name of the output files. Must be one file for each Write Node of the workflow, in the exact order

Returns

sProcessId

Id of the process representing the Workflow execution. Can be used as input to the `wWaitProcess` method or `wGetProcessStatus` methods to check the execution.

wAsynchImportProduct

`matlabwasdilib.wAsynchImportProduct(Wasdi, sProductLink, sName, sBoundingBox='', sProvider='AUTO')`

Import an EO Image in WASDI. This is the asynchronous version Syntax `sStatus=wImportProduct(Wasdi, sProductLink, sName)`

Parameters

- **Wasdi** – Wasdi object created after the wasdilib call
- **sProductLink** – Product Direct Link as returned by `wSearchEOImage`
- **sName** – Product Name as returned by `wSearchEOImage`

- **sBoundingBox** – product bounding box, optional
- **sProvider** – data provider, optional

Returns

param sProcessObjId
Identifier of the import process

wAsynchImportProductList

matlabwasdilib.**wAsynchImportProductList**(*Wasdi, asProductLinks, asProductNames*)

Import an EO Image in WASDI. This is the asynchronous version Syntax `sProcessObjId=wAsynchImportProductList(Wasdi, asProductLinks, asProductNames)`

Parameters

- **Wasdi** – Wasdi object created after the wasdilib call
- **asProductLinks** – collection of Product Direct Link as returned by `wSearchEOImage`
- **asProductNames** – collection of Product Names as returned by `wSearchEOImage`

Returns

param asStatuses
list of statuses of the import processes

wAsynchMosaic

matlabwasdilib.**wAsynchMosaic**(*Wasdi, asInputFileNames, sOutputFile, sNoDataValue, sInputIgnoreValue*)

Mosaic input images in the output file Syntax `sProcessId=wAsynchMosaic(Wasdi, asInputFileNames, sOutputFile, sNoDataValue, sInputIgnoreValue)`

Parameters

- **Wasdi** – Wasdi object created after the wasdilib call
- **asInputFileNames** – Array of input file names
- **sOutputFile** – Name of the output file
- **sNoDataValue** – value to use as no data in the output file
- **sInputIgnoreValue** – value used as no data in the input file

Returns

sProcessId
Id of the mosaic process on WASDI. Can be used as input to the `wWaitProcess` method or `wGetProcessStatus` methods to check the execution.

wAsynchMultiSubset

matlabwasdilib.**wAsynchMultiSubset**(Wasdi, sInputFile, asOutputFiles, adLatN, adLonW, adLatS, adLonE)

Extracts subsets of an image given its name and the desired bounding boxes. Asynchronous version
Syntax sReturn=wAsynchMultiSubset(Wasdi, sInputFile, asOutputFiles, adLatN, adLonW, adLatS, adLonE)

Parameters

- **Wasdi** – Wasdi object created after the wasdilib call
- **sInputFile** – the input file from where subsets must be extracted
- **asOutputFiles** – names to be given to output files
- **adLatN** – a collection of Northernmost latitudes
- **adLonW** – a collection of Westernmost longitudes
- **adLatS** – a collection of Southernmost latitudes
- **adLonE** – a collection of Easternmost longitudes

wCopyFileToSftp

matlabwasdilib.**wCopyFileToSftp**(Wasdi, sFileName, sRelativePath)

Copy file to SFTP folder, synchronous version Syntax wCopyFileToSftp(Wasdi, sFileName, sRelativePath)

Parameters

- **Wasdi** – Wasdi object created after the wasdilib call
- **sFileName** – a string containing the file name
- **sRelativePath** – a string containing the relative path

Returns

sProcessId

process workspace id. It can be used as input to the wWaitProcess method or wGetProcessStatus methods to check the execution.

wCreateWorkspace

matlabwasdilib.**wCreateWorkspace**(Wasdi, sWorkspaceName, sNodeCode='')

Copy file to SFTP folder, asynchronous version Syntax wCreateWorkspace(Wasdi, sWorkspaceName, sNodeCode)

Parameters

- **Wasdi** – Wasdi object created after the wasdilib call
- **sWorkspaceName** – the name of the workspace
- **sNodeCode** – the code of the node, optional

Returns

sWorkspaceId

the ID of the workspace (empty in case of error)

wDeleteProduct

`matlabwasdilib.wDeleteProduct(Wasdi, sProductName)`

Deletes a product in active workspace Syntax `sStatus = wDeleteProduct(Wasdi, sProductName)`

Parameters

- **Wasdi** – Wasdi object created after the wasdilib call
- **sFileName** – Name of the file to add

Returns

sStatus

empty string if deletion was successful, null in case it did not work

wDeleteWorkspace

`matlabwasdilib.wDeleteWorkspace(Wasdi, sWorkspaceName)`

Deletes a workspace. If the user is not the workspace owner, then just the sharing is deleted Syntax `sStatus = wDeleteProduct(Wasdi, sWorkspaceName)`

Parameters

- **Wasdi** – Wasdi object created after the wasdilib call
- **sWorkspaceName** – the name of the workspace to be deleted

Returns

sStatus

empty string if deletion was successful, null in case it did not work

wExecuteProcessor

`matlabwasdilib.wExecuteProcessor(Wasdi, sProcessorName, asParams)`

Execute a WASDI processor asynchronously Syntax `sStatus=wExecuteProcessor(Wasdi, sProcessorName, asParams)`

Parameters

- **Wasdi** – Wasdi object created after the wasdilib call
- **sProcessorName** – Processor Name
- **asParams** – Processor parameters, as a key/value dictionary

Returns

sProcessId

process workspace id. It can be used as input to the `wWaitProcess` method or `wGetProcessStatus` methods to check the execution.

wExecuteWorkflow

`matlabwasdilib.wExecuteWorkflow(Wasdi, sWorkflow, asInputFiles, asOutputFiles)`

Executes a SNAP workflow. The workflow has to be uploaded in WASDI: it can be public or private of a user. If it is private, it must be triggered from the owner. Syntax `sStatus = wExecuteWorkflow(Wasdi, sWorkflow, asInputFiles, asOutputFiles);`

Parameters

- **Wasdi** – Wasdi object created after the wasdilib call
- **sWorkflow** – Name of the workflow
- **asInputFiles** – array of strings with the name of the input files. Must be one file for each Read Node of the workflow, in the exact order
- **asOutputFiles** – array of strings with the name of the output files. Must be one file for each Write Node of the workflow, in the exact order

Returns

sStatus

Exit Workflow Process Status as a String: CREATED, RUNNING, STOPPED, DONE, ERROR

wGetActiveWorkspace

`matlabwasdilib.wGetActiveWorkspace(Wasdi)`

Gets the active workspace ID Syntax `sWorkspaceId = wGetActiveWorkspace(Wasdi)`

Parameters

Wasdi – Wasdi object created after the wasdilib call

Returns

sWorkspaceId

the ID of active WASDI workspace

wGetBasePath

`matlabwasdilib.wGetBasePath(Wasdi)`

Gets the base path Syntax `sBasePath = wGetBasePath(Wasdi)`

Parameters

Wasdi – Wasdi object created after the wasdilib call

Returns

sBasePath

the base path in use

wGetBaseUrl

matlabwasdilib.**wGetBaseUrl**(*Wasdi*)

Gets the base URL Syntax *sBasePath* = wGetBaseUrl(*Wasdi*)

Parameters

Wasdi – Wasdi object created after the wasdilib call

Returns

sBaseUrl

the base URL for WASDI

wGetDownloadActive

matlabwasdilib.**wGetDownloadActive**(*Wasdi*)

Gets whether download is active or not Syntax *bDownloadActive* = wGetDownloadActive(*Wasdi*)

Parameters

Wasdi – Wasdi object created after the wasdilib call

Returns

bDownloadActive

true if download is active, false otherwise

wGetFullProductPath

matlabwasdilib.**wGetFullProductPath**(*Wasdi*, *sFileName*)

Get the full local path of a product. If it is not present on the local PC and DownloadActive flag is true the product will be downloaded Syntax *sFullPath* =wGetFullProductPath(*Wasdi*, *sFileName*);

Parameters

- **Wasdi** – Wasdi object created after the wasdilib call
- **sFileName** – Name of the file

Returns

sFullPath

full local path

wGetMyProcId

matlabwasdilib.**wGetMyProcId**(*Wasdi*)

Gets own processor ID Syntax *sProcId* = wGetProcessorPayloadAsJSON(*Wasdi*, *sProcessObjId*)

Parameters

Wasdi – Wasdi object created after the wasdilib call

Returns

sProcId

own processor ID

wGetParameter

`matlabwasdilib.wGetParameter(Wasdi, sKey)`

Get the value of a parameter identified by `sKey` in the parameters file Syntax `sParameter = wGetParameter(Wasdi, sKey)`

Parameters

- **Wasdi** – Wasdi object created after the wasdilib call
- **sKey** – The KEY of the parameter in the paramteres file

Returns

sParameter

The value of the parameter. If it does not exists the function returns ""

wGetParametersFilePath

`matlabwasdilib.wGetParametersFilePath(Wasdi)`

Gets the parameters file path Syntax `sParametersFilePath = wGetParametersFilPath(Wasdi)`

Parameters

Wasdi – Wasdi object created after the wasdilib call

Returns

sParametersFilePath

the path to the parameters file

wGetParams

`matlabwasdilib.wGetParams(Wasdi)`

Gets processor parameters Syntax `asParams = wGetParams(Wasdi)`

Parameters

Wasdi – Wasdi object created after the wasdilib call

Returns

asParams

a map containing the parameters

wGetPassword

`matlabwasdilib.wGetPassword(Wasdi)`

Gets the password Syntax `sPassword = wGetPassword(Wasdi)`

Parameters

Wasdi – Wasdi object created after the wasdilib call

Returns

sPassword

WASDI user's password

wGetPath

matlabwasdilib.**wGetPath**(Wasdi, sFileName)

Gets the path of given product Syntax sPath = wGetPath(Wasdi, sFileName)

Parameters

Wasdi – Wasdi object created after the wasdilib call

Returns

sPath

wasdi local path for given product

wGetProcessStatus

matlabwasdilib.**wGetProcessStatus**(Wasdi, sProcessId)

Get the status of a Process Syntax sStatus =wGetProcessStatus(Wasdi, sProcessId);

Parameters

- **Wasdi** – Wasdi object created after the wasdilib call
- **sProcessId** – Id of the process to query

Returns

sStatus

Process Status as a String: CREATED, RUNNING, STOPPED, DONE, ERROR

wGetProcessesByWorkspace

matlabwasdilib.**wGetProcessesByWorkspace**(Wasdi, iStartIndex=0, iEndIndex=, []sStatus=, []sOperationType=, []sNamePattern=[])

Get a paginated list of processes in the active workspace Syntax asProcesses=wgetProcessesByWorkspace(Wasdi, iStartIndex, iEndIndex=[], sStatus=[], sOperationType=[], sNamePattern=[]?)

Parameters

- **Wasdi** – Wasdi object created after the wasdilib call
- **iStartIndex** – first index
- **iEndIndex** – last index
- **sStatus** – filter by statuses
- **sOperationType** – filter by operation name
- **sNamePattern** – filter by name

Returns

asProcesses

list of processes

wGetProcessorPath

`matlabwasdilib.wGetProcessorPath(Wasdi)`

Gets the parameters file path Syntax `sProcessorPath = wGetProcessorPath(Wasdi)`

Parameters

Wasdi – Wasdi object created after the wasdilib call

Returns

sProcessorPath

the path to current processor

wGetProcessorPayload

`matlabwasdilib.wGetProcessorPayload(Wasdi, sProcessObjId)`

Gets the payload of given processor Syntax `oProcessPayload = wGetProcessorPayload(Wasdi, sProcessObjId)`

Parameters

- **Wasdi** – Wasdi object created after the wasdilib call
- **sProcessObjId** – process ID for which the payload must be retrieve

Returns

oProcessPayload

an object containing the payload

wGetProcessorPayloadAsJSON

`matlabwasdilib.wGetProcessorPayloadAsJSON(Wasdi, sProcessObjId)`

Gets the payload of given processor as a JSON string Syntax `sProcessPayload = wGetProcessorPayloadAsJSON(Wasdi, sProcessObjId)`

Parameters

- **Wasdi** – Wasdi object created after the wasdilib call
- **sProcessObjId** – process ID for which the payload must be retrieve

Returns

sProcessPayload

a JSON formatted string containing the payload

wGetProductBbox

`matlabwasdilib.wGetProductBbox(Wasdi, sProductName)`

Gets own processor ID Syntax `sProcId = wGetProductBbox(Wasdi, sProductName)`

Parameters

- **Wasdi** – Wasdi object created after the wasdilib call
- **sProductName** – the product name for which the bounding box must be retrieved

Returns

sBbox

the requested bounding box

wGetProductsByActiveWorkspace

matlabwasdilib.**wGetProductsByActiveWorkspace**(*Wasdi*)

Get the List of Products in the active Workspace Syntax asProducts=wGetProductsByActiveWorkspace(*Wasdi*);

Parameters

Wasdi – Wasdi object created after the wasdilib call

Returns

asProducts

array of strings that are the names of the products

wGetProductsByWorkspace

matlabwasdilib.**wGetProductsByWorkspace**(*Wasdi*, *sWorkspaceName*)

Get the List of Products in a Workspace Syntax asProducts=wGetProductsByWorkspace(*Wasdi*, *sWorkspaceName*);

Parameters

- **Wasdi** – Wasdi object created after the wasdilib call
- **sWorkspaceName** – name of the workspace

Returns

asProducts

array of strings that are the names of the products

wGetSavePath

matlabwasdilib.**wGetSavePath**(*Wasdi*)

Get the full local path where to save a product in the active workspace Syntax sSavePath =wGetSavePath(*Wasdi*);

Parameters

Wasdi – Wasdi object created after the wasdilib call

Returns

sSavePath

the local path to use to save the file, including last /

wGetSessionId

matlabwasdilib.**wGetSessionId**(Wasdi)

Get the session ID Syntax sSessionId = wGetSessionId(Wasdi)

Parameters

Wasdi – Wasdi object created after the wasdilib call

Returns

sSessionId

the current session

wGetUploadActive

matlabwasdilib.**wGetUploadActive**(Wasdi)

Gets whether Upload is active or not Syntax bUploadActive = wGetUploadActive(Wasdi)

Parameters

Wasdi – Wasdi object created after the wasdilib call

Returns

bUploadActive

true if Upload is active, false otherwise

wGetUser

matlabwasdilib.**wGetUser**(Wasdi)

Gets the user Syntax sUser = wGetUser(Wasdi)

Parameters

Wasdi – Wasdi object created after the wasdilib call

Returns

sUser

the username on wasdi

wGetVerbose

matlabwasdilib.**wGetVerbose**(Wasdi)

Gets verbosity flag Syntax bVerbose = wGetVerbose(Wasdi)

Parameters

Wasdi – Wasdi object created after the wasdilib call

Returns

bVerbose

verbosity flag

wGetWorkflows

matlabwasdilib.**wGetWorkflows**(*Wasdi*)

Get the List of Workflows of the actual User Syntax [asWorkflowNames, asWorkflowIds]=wGetWorkflows(Wasdi);

Parameters

Wasdi – Wasdi object created after the wasdilib call

Returns

asWorkflowNames

array of strings that are the names of the workflows

asWorkflowIds

array of strings that are the id of the workflows

wGetWorkspaceBaseUrl

matlabwasdilib.**wGetWorkspaceBaseUrl**(*Wasdi*)

Gets the workspace base URL Syntax sWorkspaceBaseUrl = wGetWorkspaceBaseUrl(Wasdi)

Parameters

Wasdi – Wasdi object created after the wasdilib call

Returns

sWorkspaceBaseUrl

the base URL for active workspace

wGetWorkspaceIdByName

matlabwasdilib.**wGetWorkspaceIdByName**(*Wasdi*, *sWorkspaceName*)

Get the Id of a Workspace from the name Syntax sWorkspaceId=wGetWorkspaceIdByName(Wasdi, sWorkspaceName);

Parameters

- **Wasdi** – Wasdi object created after the wasdilib call
- **sWorkspaceName** – Name of the workspace

Returns

sWorkspaceId

id of the workspace

wGetWorkspaceOwnerByName

matlabwasdilib.**wGetWorkspaceOwnerByName**(*Wasdi*, *sWorkspaceName*)

Gets the owner of the workspace given its name Syntax sWorkspaceOwner = wGetWorkspaceOwnerByName(Wasdi, sWorkspaceName)

Parameters

- **Wasdi** – Wasdi object created after the wasdilib call
- **sWorkspaceName** – the name of the workspace

Returns

sWorkspaceOwner
the owner of the workspace

wGetWorkspaceUrlByWsId

```
matlabwasdilib.wGetWorkspaceUrlByWsId(Wasdi, sWorkspaceId)
```

Gets the workspace URL given its ID Syntax: sWorkspaceUrl = wGetWorkspaceUrlByWsId(Wasdi, sWorkspaceId)

Parameters

Wasdi – Wasdi object created after the wasdilib call

Returns

sBaseUrl
the base URL for WASDI

wGetWorkspaces

```
matlabwasdilib.wGetWorkspaces(Wasdi)
```

wImportAndPreprocess

```
matlabwasdilib.wImportAndPreprocess(Wasdi, asProductLinks, asProductNames,  
sWorkflowName, sSuffix, sProvider=[])
```

Import and preprocess a collection of EO products Syntax wImportAndPreprocess(Wasdi, asProductLinks, asProductNames, sWorkflowName, sSuffix)

Parameters

- **Wasdi** – Wasdi object created after the wasdilib call
- **asProductLinks** – collection of Product Direct Link as returned by wSearchEOImages
- **asProductNames** – collection of Product names, as returned by wSearchEOImages
- **sWorkflowName** – the name of the SNAP workflow to be applied to downloaded imagesc
- **sSuffix** – the suffix to append to the preprocessed files
- **sProvider** – optional, the provider from where data must be collected

wImportProduct

```
matlabwasdilib.wImportProduct(Wasdi, sProductLink, sProductName, sBoundingBox='',  
sProvider='AUTO')
```

Import an EO Image in WASDI Syntax sStatus=wImportProduct(Wasdi, sProductLink, sProductName)

Parameters

- **Wasdi** – Wasdi object created after the wasdilib call

- **sProductLink** – Product Direct Link as returned by wSearchEOImage
- **sProductName** – Product Name as returned by wSearchEOImage
- **sBoundingBox** – product bounding box, optional
- **sProvider** – data provider, optional

Returns**sProcessObjId**

Identifier of the import process

wImportProductList`matlabwasdilib.wImportProductList(Wasdi, asProductLinks, asProductNames)`

Import an EO Image in WASDI. This is the asynchronous version Syntax `sProcessObjId=wAsynchImportProductList(Wasdi, asProductLinks, asProductNames)`

Parameters

- **Wasdi** – Wasdi object created after the wasdilib call
- **asProductLinks** – collection of Product Direct Link as returned by wSearchEOImage
- **asProductNames** – collection of Product Names as returned by wSearchEOImage

Returns**asStatuses**

list of statuses of the import processes

wLog`matlabwasdilib.wLog(Wasdi, sLogRow)`

Add a row to the application logs. Locally, just print on the console if VERBOSE. On the server, it logs on the WASDI interface. Syntax `wLog(Wasdi, sLogRow)`

Parameters

- **Wasdi** – Wasdi object created after the wasdilib call
- **sLogRow** – Text to log

wMosaic`matlabwasdilib.wMosaic(Wasdi, asInputFileNames, sOutputFile, sNoDataValue, sInputIgnoreValue)`

Mosaic input images in the output file Syntax `sStatus=wMosaic(Wasdi, asInputFileNames, sOutputFile, sNoDataValue, sInputIgnoreValue)`

Parameters

Wasdi – Wasdi object created after the wasdilib call **asInputFileNames**: Array of input file names **sOutputFile**: Name of the output file **sNoDataValue**: value to use as no data in the output file **sInputIgnoreValue**: value used as no data in the input file

Returns

sStatus
end status of the mosaic operation

wMultiSubset

matlabwasdilib.**wMultiSubset**(*Wasdi*, *sInputFile*, *asOutputFiles*, *adLatN*, *adLonW*, *adLatS*, *adLonE*)

Extracts subsets of an image given its name and the desired bounding boxes Syntax sReturn=wMultiSubset(Wasdi, sInputFile, asOutputFiles, adLatN, adLonW, adLatS, adLonE)

Parameters

- **Wasdi** – Wasdi object created after the wasdilib call
- **sInputFile** – the input file from where subsets must be extracted
- **asOutputFiles** – names to be given to output files
- **adLatN** – a collection of Northernmost latitudes
- **adLonW** – a collection of Westernmost longitudes
- **adLatS** – a collection of Southernmost latitudes
- **adLonE** – a collection of Easternmost longitudes

wOpenWorkspace

matlabwasdilib.**wOpenWorkspace**(*Wasdi*, *sWorkspaceName*)

Open a Workspace Syntax sWorkspaceId=wOpenWorkspace(Wasdi, sWorkspaceName);

Parameters

- **Wasdi** – Wasdi object created after the wasdilib call
- **sWorkspaceName** – Name of the workspace

Returns

sWorkspaceId
id of the workspace

wOpenWorkspaceById

matlabwasdilib.**wOpenWorkspaceById**(*Wasdi*, *sWorkspaceId*)

Opens a workspace given its ID Syntax sWorkspaceId=wOpenWorkspaceById(Wasdi,sWorkspaceId)

Parameters

- **Wasdi** – Wasdi object created after the wasdilib call
- **sWorkspaceId** – ID of the workspace to open

Returns

sWorkspaceId
the ID of the workspace if successfully opened, empty string otherwise

wPrintStatus

```
matlabwasdilib.wPrintStatus(Wasdi)
```

Prints the status Syntax: printStatus(Wasdi)

Parameters

Wasdi – Wasdi object created after the wasdilib call

wRefreshParameters

```
matlabwasdilib.wRefreshParameters(Wasdi)
```

Read again the parameters from the configured file Syntax sParameter = wRefreshParameters(Wasdi, sKey)

Parameters

Wasdi – Wasdi object created after the wasdilib call

wSearchEOImages

```
matlabwasdilib.wSearchEOImages(Wasdi, sPlatform, sDateFrom, sDateTo, dULLat, dULLon,  
                                dLRLat, dLRLon, sProductType, iOrbitNumber,  
                                sSensorOperationalMode, sCloudCoverage)
```

Search EO Images. Returns 3 parallel arrays: one with the names, one with the links and one with the footprints of the found products. The links and footprints can be used as input to the wImportProduct function, that imports the product in the active workspace Syntax [asProductNames, asProductLinks, asProductFootprints]=wSearchEOImages(Wasdi, sPlatform, sDateFrom, sDateTo, dULLat, dULLon, dLRLat, dLRLon, sProductType, iOrbitNumber, sSensorOperationalMode, sCloudCoverage);

Parameters

- **Wasdi** – Wasdi object created after the wasdilib call
- **sPlatform** – Satellite Platform. Accepts “S1”, “S2”
- **sDateFrom** – Starting date in format “YYYY-MM-DD”
- **sDateTo** – End date in format “YYYY-MM-DD”
- **dULLat** – Upper Left Lat Coordinate. Can be null.
- **dULLon** – Upper Left Lon Coordinate. Can be null.
- **dLRLat** – Lower Right Lat Coordinate. Can be null.
- **dLRLon** – Lower Right Lon Coordinate. Can be null.
- **sProductType** – Product Type. If Platform = “S1” -> Accepts “SLC”, “GRD”, “OCN”. If Platform = “S2” -> Accepts “S2MSI1C”, “S2MSI2Ap”, “S2MSI2A”. Can be null.
- **iOrbitNumber** – Sentinel Orbit Number. Can be null.
- **sSensorOperationalMode** – Sensor Operational Mode. ONLY for S1. Accepts -> “SM”, “IW”, “EW”, “WV”. Can be null. Ignored for Platform “S2”
- **sCloudCoverage** – sCloudCoverage Cloud Coverage. Sample syntax: [0 TO 9.4]

Returns

asProductNames

array of strings that are the names of the found products

asProductLinks

array of strings that are the links to download the products

asProductFootprints

array of strings that are the footprints of found products in WKT

wSetActiveWorkspaceId

`matlabwasdilib.wSetActiveWorkspaceId(Wasdi, sNewActiveWorkspaceId)`

Set the active workspace Syntax `wSetActiveWorkspaceId(Wasdi, sNewActiveWorkspaceId)`

Parameters

- **Wasdi** – Wasdi object created after the wasdilib call
- **sNewActiveWorkspaceId** – the workspace ID to open

wSetBasePath

`matlabwasdilib.wSetBasePath(Wasdi, sNewBasePath)`

Set the base path Syntax `wSetBasePath(Wasdi, sNewBasePath)`

Parameters

- **Wasdi** – Wasdi object created after the wasdilib call
- **sNewBasePath** – the new base path

wSetBaseUrl

`matlabwasdilib.wSetBaseUrl(Wasdi, sBaseUrl)`

Set the base URL Syntax `wSetBaseUrl(Wasdi, sBaseUrl)`

Parameters

- **Wasdi** – Wasdi object created after the wasdilib call
- **sBaseUrl** – the new base URL (must be valid)

wSetDownloadActive

`matlabwasdilib.wSetDownloadActive(Wasdi, iActive)`

Set the download active flag Syntax `wSetDownloadActive(Wasdi, iActive)`

Parameters

- **Wasdi** – Wasdi object created after the wasdilib call
- **iActive** – true/false

wSetIsOnServer

matlabwasdilib.**wSetIsOnServer**(*Wasdi*, *bIsOnServer*)

Set is on server flag Syntax wSetVerbose(*Wasdi*, *bIsOnServer*)

Parameters

- **Wasdi** – Wasdi object created after the wasdilib call
- **bIsOnServer** – true/false

wSetMyProcId

matlabwasdilib.**wSetMyProcId**(*Wasdi*, *sMyNewProcId*)

Set the processor ID Syntax wSetMyProcId(*Wasdi*, *sMyNewProcId*)

Parameters

- **Wasdi** – Wasdi object created after the wasdilib call
- **sMyNewProcId** – the new processor ID

wSetParameter

matlabwasdilib.**wSetParameter**(*Wasdi*, *sKey*, *sValue*)

Set the value of a parameter Syntax sParameter = wSetParameter(*Wasdi*, *sKey*, *sValue*)

Parameters

- **Wasdi** – Wasdi object created after the wasdilib call
- **sKey** – The KEY of the parameter to add or update
- **sValue** – The the value of the parameter

Returns

sParameter

The value (same as sValue in input)

wSetPassword

matlabwasdilib.**wSetPassword**(*Wasdi*, *sPassword*)

Set the WASDI user password Syntax wSetPassword(*Wasdi*, *sPassword*)

Parameters

- **Wasdi** – Wasdi object created after the wasdilib call
- **sPassword** – the password

wSetPayload

`matlabwasdilib.wSetPayload(Wasdi, sPayload)`

Writes a Payload in a process Syntax `sStatus = wSetProcessPayload(Wasdi, sProcessId, sData);`

Parameters

- **Wasdi** – Wasdi object created after the wasdilib call
- **sPayload** – the process payload

wSetProcessPayload

`matlabwasdilib.wSetProcessPayload(Wasdi, sProcessId, sData)`

Writes a Payload in a process Syntax `sStatus = wSetProcessPayload(Wasdi, sProcessId, sData);`

Parameters

- **Wasdi** – Wasdi object created after the wasdilib call
- **sProcessId** – Id of the process to update
- **sData** – Data to write as payload

Returns

sStatus

Process Status as a String: CREATED, RUNNING, STOPPED, DONE, ERROR

wSetSessionId

`matlabwasdilib.wSetSessionId(Wasdi, sSessionId)`

Set the session ID Syntax `wSetSessionId(Wasdi, sSessionId)`

Parameters

- **Wasdi** – Wasdi object created after the wasdilib call
- **sSessionId** – the session ID

wSetSubPid

`matlabwasdilib.wSetSubPid(Wasdi, sProcessId, iSubPid)`

Set the sub pid Syntax `sStatus = wSetSubPid(Wasdi, sProcessId, iSubPid)`

Parameters

- **Wasdi** – Wasdi object created after the wasdilib call
- **sProcessId** – the process ID
- **iSubPid** – the sub pid

wSetUpUploadActive

`matlabwasdilib.wSetUpUploadActive(Wasdi, iActive)`

Set the Upload active flag Syntax `wSetUpUploadActive(Wasdi, iActive)`

Parameters

- **Wasdi** – Wasdi object created after the wasdilib call
- **iActive** – true/false

wSetUser

`matlabwasdilib.wSetUser(Wasdi, sUser)`

Set the user Syntax `wSetUser(Wasdi, sUser)`

Parameters

- **Wasdi** – Wasdi object created after the wasdilib call
- **sUser** – the user

wSetVerbose

`matlabwasdilib.wSetVerbose(Wasdi, bVerbose)`

Set verbose flag Syntax `wSetVerbose(Wasdi, bVerbose)`

Parameters

- **Wasdi** – Wasdi object created after the wasdilib call
- **bVerbose** – true/false

wSetWorkspaceBaseUrl

`matlabwasdilib.wSetWorkspaceBaseUrl(Wasdi, sUrl)`

Set the workspace base URL Syntax `wSetWorkspaceBaseUrl(Wasdi, sUrl)`

Parameters

- **Wasdi** – Wasdi object created after the wasdilib call
- **sUrl** – the workspace base URL

wSubset

`matlabwasdilib.wSubset(Wasdi, sInputFile, sOutputFile, dLatN, dLonW, dLatS, dLonE)`

Make a Subset (tile) of an input image in a specified Lat Lon Rectangle Syntax `sStatus=wSubset(Wasdi, sInputFile, sOutputFile, dLatN, dLonW, dLatS, dLonE)`

Parameters

- **Wasdi** – Wasdi object created after the wasdilib call
- **sInputFile** – Name of the input file
- **sOutputFile** – Name of the output file

- **dLatN** – Lat North Coordinate
- **dLonW** – Lon West Coordinate
- **dLatS** – Lat South Coordinate
- **dLonE** – Lon East Coordinate

Returns**sStatus**

Status of the operation

wUpdateProcessStatus

`matlabwasdilib.wUpdateProcessStatus(Wasdi, sProcessId, sStatus, iPerc)`

Updates the status of a Process Syntax `sStatus = wUpdateProcessStatus(Wasdi, sProcessId, sStatus, iPerc);`

Parameters

- **Wasdi** – Wasdi object created after the wasdilib call
- **sProcessId** – Id of the process to update
- **sStatus** – updated status. Must be CREATED, RUNNING, STOPPED, DONE, ERROR
- **iPerc** – progress percentage of the process

Returns**sOutputStatus**

Process Status Updated as a String: CREATED, RUNNING, STOPPED, DONE, ERROR

wUpdateProgress

`matlabwasdilib.wUpdateProgress(Wasdi, iPerc)`

Updates the progress of the processor Syntax `sStatus = wUpdateProgress(Wasdi, iPerc);`

Parameters

- **Wasdi** – Wasdi object created after the wasdilib call
- **iPerc** – progress percentage of the own process

Returns**sOutputStatus**

Process Status as a String: CREATED, RUNNING, STOPPED, DONE, ERROR

wUpdateProgressPerc

`matlabwasdilib.wUpdateProgressPerc(Wasdi, iPerc)`

Updates the status of a process Syntax `sStatus = wUpdateProgressPerc(Wasdi, iPerc)`

Parameters

- **Wasdi** – Wasdi object created after the wasdilib call
- **iPerc** – the %of completion

Returns

sStatus

updated status as a String or '' if there was any problem

wUpdateStatus

`matlabwasdilib.wUpdateStatus(wasdi, sStatus, iPerc=[])`

updates the status and, optionally, the progress percent syntax: `sStatus = wUpdateStatus(Wasdi, sStatus, iPerc=[])`

Parameters

- **Wasdi** – Wasdi object created after the wasdilib call
- **sStatus** – the status to be set
- **iPerc** – optional, the progress percent

wUrlEncode

`matlabwasdilib.wUrlEncode(s)`

wWaitProcess

`matlabwasdilib.wWaitProcess(Wasdi, sProcessId)`

Wait for the end of a process Syntax `sStatus =wWaitProcess(Wasdi, sProcessId);`

Parameters

- **Wasdi** – Wasdi object created after the wasdilib call
- **sProcessId** – Id of the process to wait

Returns

sStatus

exit status of the process: CREATED, RUNNING, STOPPED, DONE, ERROR

wasdiHello

```
matlabwasdilib.wasdiHello(Wasdi)
```

Hello world in WASDI. Useful for testing the setup Syntax: wasdiHello(*Wasdi*)

Parameters

Wasdi – Wasdi object created after the wasdilib call

wasdiLog

```
matlabwasdilib.wasdiLog(Wasdi, sLine)
```

Logs a line Syntax wasdiLog(*Wasdi*, *sLine*)

Parameters

- **Wasdi** – Wasdi object created after the wasdilib call
- **sLine** – the string to be logged

5.4 Octave WasdiLib

5.4.1 Methods

startWasdi

```
octavewasdilib.startWasdi()
```

Initialize the Wasdi object

Parameters

config_path – The path to be used to import the configuration.

Returns

Wasdi The object to be used to invoke all the following call to Wasdi services

addFileToWASDI

```
octavewasdilib.addFileToWASDI(Wasdi, sFileName)
```

Ingest a new file in the Active WASDI Workspace waiting for the result The method takes a file saved in the workspace root (see `getSaveFilePath`) not already added to the WS o work be sure that the file is on the server Syntax `sStatus =addFileToWASDI(Wasdi, sFileName);`

Parameters

- **Wasdi** – Wasdi object created after the wasdilib call
- **sFileName** – Name of the file to add

Returns

sStatus Status of the Ingest Process as a String: CREATED, RUNNING, STOPPED, DONE, ERROR

executeWorkflow

`octavewasdilibr.executeWorkflow(Wasdi, sWorkflow, asInputFiles, asOutputFiles)`

Executes a SNAP workflow. The workflow has to be uploaded in WASDI: it can be public or private of a user. If it is private, it must be triggered from the owner. Syntax `sStatus = wExecuteWorkflow(Wasdi, sWorkflow, asInputFiles, asOutputFiles)`

Parameters

- **Wasdi** – Wasdi object created after the wasdilibr call
- **sWorkflow** – Name of the workflow
- **asInputFiles** – array of strings with the name of the input files. Must be one file for each Read Node of the workflow, in the exact order
- **asOutputFiles** – array of strings with the name of the output files. Must be one file for each Write Node of the workflow, in the exact order

Returns

`sStatus` Exit Workflow Process Status as a String: CREATED, RUNNING, STOPPED, DONE, ERROR

getFullProductPath

`octavewasdilibr.getFullProductPath(Wasdi, sFileName)`

Get the full local path of a product. If it is not present on the local PC and DownloadActive flag is true the product will be downloaded. Syntax `sFullPath = getFullProductPath(Wasdi, sFileName);`

Parameters

- **Wasdi** – Wasdi object created after the wasdilibr call
- **FileName** – Name of the file

Returns

`sFullPath` full local path

getProcessStatus

`octavewasdilibr.getProcessStatus(Wasdi, sProcessId)`

Get the status of a Process Syntax `sStatus = getProcessStatus(Wasdi, sProcessId);`

Parameters

- **Wasdi** – Wasdi object created after the wasdilibr call
- **sProcessId** – Id of the process to query

Returns

`sStatus` Process Status as a String: CREATED, RUNNING, STOPPED, DONE, ERROR

getProductsByWorkspace

octavewasdilibrb.getProductsByWorkspace(*Wasdi*, *sWorkspaceName*)

Get the List of Products in a Workspace Syntax asProducts=getProductsByWorkspace(*Wasdi*, *sWorkspaceName*); :param *Wasdi*: Wasdi object created after the wasdilibrb call :param *sWorkspaceName*: name of the workspace

returns

asProducts array of strings that are the names of the products

getSavePath

octavewasdilibrb.getSavePath(*Wasdi*)

Get the full local path where to save a product in the active workspace Syntax sSavePath =getSavePath(*Wasdi*);

Parameters

Wasdi – Wasdi object created after the wasdilibrb call

Returns

sSavePath the local path to use to save the file, including last /

getWorkflows

octavewasdilibrb.getWorkflows(*Wasdi*)

Get the List of Workflows of the actual User Syntax [asWorkflowNames, asWorkflowIds]=getWorkflows(*Wasdi*);

Parameters

Wasdi – Wasdi object created after the wasdilibrb call

Returns

asWorkflowNames array of strings that are the names of the workflows

Returns

asWorkflowIds array of strings that are the id of the workflows

getWorkspaces

octavewasdilibrb.getWorkspaces(*Wasdi*)

Get the List of Workspace of the actual User Syntax [asWorkspaceNames, asWorkspaceIds]=getWorkspaces(*Wasdi*);

Parameters

Wasdi – Wasdi object created after the wasdilibrb call

Returns

asWorkspaceNames array of strings that are the names of the workspaces

Returns

asWorkspaceIds array of strings that are the id of the workspaces

openWorkspace

octavewasdilib.**openWorkspace**(*Wasdi*, *sWorkspaceName*)

Open a Workspace Syntax `sWorkspaceId=openWorkspace(Wasdi, sWorkspaceName);`

INPUT :param Wasdi: Wasdi object created after the wasdilib call :param sWorkspaceName: Name of the workspace

Returns sWorkspaceId
id of the workspace

setProcessPayload

octavewasdilib.**setProcessPayload**(*Wasdi*, *sProcessId*, *sData*)

Writes a Payload in a process Syntax `sStatus=setProcessPayload(Wasdi, sProcessId, sData);`

Parameters

- **Wasdi** – Wasdi object created after the wasdilib call
- **sProcessId** – Id of the process to update
- **sData** – Data to write as payload

Returns

`sStatus` Process Status as a String: CREATED, RUNNING, STOPPED, DONE, ERROR

updateProcessStatus

octavewasdilib.**updateProcessStatus**(*Wasdi*, *sProcessId*, *sStatus*, *iPerc*)

Updates the status of a Process Syntax `sStatus=updateProcessStatus(Wasdi, sProcessId, sStatus, iPerc);`

Parameters

- **Wasdi** – Wasdi object created after the wasdilib call
- **sProcessId** – Id of the process to update
- **sStatus** – updated status. Must be CREATED, RUNNING, STOPPED, DONE, ERROR
- **iPerc** – progress percentage of the process

Returns

`sOutputStatus` Process Status Updated as a String: CREATED, RUNNING, STOPPED, DONE, ERROR

waitProcess

`octavewasdilib.waitProcess(Wasdi, sProcessId)`

Wait for the end of a process Syntax `sStatus =waitProcess(Wasdi, sProcessId);`

Parameters

- **Wasdi** – Wasdi object created after the wasdilib call
- **sProcessId** – Id of the process to wait

returns

`sStatus` exit status of the process: CREATED, RUNNING, STOPPED, DONE, ERROR

5.5 Python WasdiLib

5.5.1 Legal Notice

WASDI Sàrl

Disclaimer The library is provided “as-is” without warranty

Neither FadeOut Software (IT) Srl or any of its partners or agents shall be liable for any direct, indirect, incidental, special, exemplary, or consequential damages (including, but not limited to, breach of expressed or implied contract; procurement of substitute goods or services; loss of use, data or profits; business interruption; or damage to any equipment, software and/or data files) however caused and on any legal theory of liability, whether for contract, tort, strict liability, or a combination thereof (including negligence or otherwise) arising in any way out of the direct or indirect use of software, even if advised of the possibility of such risk and potential damage.

FadeOut Software (IT) Srl uses all reasonable care to ensure that software products and other files that are made available are safe to use when installed, and that all products are free from any known software virus. For your own protection, you should scan all files for viruses prior to installation.

WASDI

This is WASPY, the WASDI Python lib.

WASDI is an ESA GSTP Project sponsored by ASI in 2016. The system is a fully scalable and distributed Cloud based EO analytical platform. The system is cross-cloud and cross DIAS. WASDI is an operating platform that offers services to develop and deploy DIAS based EO on-line applications, designed to extract value-added information, made and distributed by EO-Experts without any specific IT/Cloud skills. WASDI offers as well to End-Users the opportunity to run EO applications both from a dedicated user-friendly interface and from an API based software interface, fulfilling the real-world business needs. EO-Developers can work using the WASDI Libraries in their usual programming languages and add to the platform these new blocks in the simplest possible way.

Note: the philosophy of safe programming is adopted as widely as possible, the lib will try to workaround issues such as faulty input, and print an error rather than raise an exception, so that your program can possibly go on. Please check the return statues

Version 0.6.2 Last Update: 10/03/2021

Tested with: Python 2.7, Python 3.7

5.5.2 Methods

addFileToWASDI

`wasdi.addFileToWASDI(sFileName, sStyle="")`

Add a file to the wasdi workspace

Parameters

- **sFileName** – Name (with extension) of the file to add
- **sStyle** – name of a valid WMS style

Returns

status of the operation

addParameter

`wasdi.addParameter(sKey, oValue)`

Adds a parameter

Parameters

- **sKey** – parameter key
- **oValue** – parameter value

getParameter

`wasdi.getParameter(sKey, oDefault=None)`

Gets a parameter using its key

Parameters

- **sKey** – parameter key
- **oDefault** – Default value to return if parameter is not present

Returns

parameter value

getParametersDict

`wasdi.getParametersDict()`

Get the full Params Dictionary

Returns

a dictionary containing the parameters

getParametersFilePath

`wasdi.getParametersFilePath()`

Get the local parameters file Path

Returns

local paramters file path

getSessionId

`wasdi.getSessionId()`

Get the WASDI Session

Returns

Session Id [String]

getPassword

`wasdi.getPassword()`

Get the WASDI Password

getUser

`wasdi.getUser()`

Get the WASDI User

getVerbose

`wasdi.getVerbose()`

Get Verbose Flag

Returns

True or False

getWorkflows

`wasdi.getWorkflows()`

Get the list of workflows for the user

Returns

None if there is any error; an array of WASDI Workspace JSON Objects if everything is ok.

The format is as follows:

```
{  
  "description":STRING, "name": STRING, "workflowId": STRING  
}
```

getFoundProductName

`wasdi.getFoundProductName(aoProduct)`

Get The name of a product from a Dictionary returned by Search EO Images

Parameters

aoProduct – dictionary representing the product as returned by Search EO Images

Returns

product name or “” if there was any error

getProductBBOX

`wasdi.getProductBBOX(sFileName)`

Gets the bounding box of a file

Parameters

sFileName – name of the file to query for bounding box

Returns

Bounding Box if available as a String comma separated in form SOUTH, WEST, EST, NORTH

getProcessorPath

`wasdi.getProcessorPath()`

Get the local path of the processor (where myProcessor.py is located)

Returns

Local path of the processor

getProcessesByWorkspace

`wasdi.getProcessesByWorkspace(iStartIndex=0, iEndIndex=20, sStatus=None, sOperationType=None, sName=None)`

Get a paginated list of processes in the active workspace

Parameters

- **iStartIndex** – start index of the process (0 by default is the last one)
- **iEndIndex** – end index of the process (20 by default)
- **sStatus** – status filter. None by default. Can be CREATED, RUNNING, STOPPED, DONE, ERROR, WAITING, READY
- **sOperationType** – Operation Type Filter. None by default. Can be RUNPROCESSOR, RUNIDL, RUNMATLAB, INGEST, DOWNLOAD, GRAPH, DEPLOYPROCESSOR
- **sName** – Name filter. The name meaning depends by the operation type. None by default. For RUNPROCESSOR, RUNIDL and RUNMATLAB is the name of the application

getBaseUrl

`wasdi.getBaseUrl()`

Get the WASDI API URL

Returns

WASDI API URL

setWorkspaceBaseUrl

`wasdi.setWorkspaceBaseUrl(sWorkspaceBaseUrl)`

Set the Workspace specific API URL

Parameters

sWorkspaceBaseUrl – Workspace API URL

getWorkspaceBaseUrl

`wasdi.getWorkspaceBaseUrl()`

Get the Workspace API URL

Returns

Workspace API URL

setIsOnServer

`wasdi.setIsOnServer(bIsOnServer)`

Set the Is on Server Flag: keep it false, as default, while developing

Parameters

bIsOnServer – set the flag to know if the processor is running on the server or on the local PC

getIsOnServer

`wasdi.getIsOnServer()`

Are we running on a WASDI Server?

Returns

True if it is running on server, False if it is running on the local Machine

setDownloadActive

`wasdi.setDownloadActive(bDownloadActive)`

When in development, set True to download locally files from Server. Set it to false to NOT download data. In this case the developer must check the availability of the files

Parameters

bDownloadActive – True (default) to activate autodownload. False to disactivate

getDownloadActive

`wasdi.getDownloadActive()`

Get the Download Active Flag

Returns

True if auto download is active, False if it is not active

setUploadActive

`wasdi.setUploadActive(bUploadActive)`

When in development, set True to upload local files on Server. Set it to false to NOT upload data. In this case the developer must check the availability of the files

Parameters

bUploadActive – True to activate Auto Upload, False to disactivate auto upload

getUploadActive

`wasdi.getUploadActive()`

Get the Upload Active Flag

Returns

True if Auto Upload is Active, False if it is NOT Active

setProclD

`wasdi.setProcId(sProcID)`

Own Proc Id

Parameters

sProcID – self processor identifier

getProclD

`wasdi.getProcId()`

Get the Own Proc Id

Returns

Own Processor Identifier

setActiveWorkspaceId

`wasdi.setActiveWorkspaceId(sActiveWorkspace)`

Set the Active Workspace Id

Parameters

sActiveWorkspace – Active Workspace Id

getActiveWorkspaceId

`wasdi.getActiveWorkspaceId()`

Get Active workspace Id

Returns

the WorkspaceId as a String, "" if there is any error

refreshParameters

`wasdi.refreshParameters()`

Refresh parameters, reading the file again

init

`wasdi.init(sConfigFilePath=None)`

Init WASDI Library. Call it after setting user, password, path and url or use it with a config file

Parameters

sConfigFilePath – local path of the config file. In None or the file does not exists, WASDI will ask for login in the console

Returns

True if login was successful, False otherwise

hello

`wasdi.hello()`

Hello Wasdi to test the connection.

Returns

the hello message as Text

getWorkspaces

`wasdi.getWorkspaces()`

Get List of user workspaces

Returns

an array of WASDI Workspace JSON Objects.

Each Object is like this {

```
    "ownerUserId":STRING,  "sharedUsers":[STRING],  "workspaceId":STRING,  "workspace-  
    Name":STRING
```

```
}
```

createWorkspace

`wasdi.createWorkspace(sName=None)`

Create a new workspaces and set it as ACTIVE Workspace

Parameters

sName – Name of the workspace to create. Null by default

Returns

Workspace Id as a String if it is a success, None otherwise

deleteWorkspace

`wasdi.deleteWorkspace(sWorkspaceId)`

Delete a workspace

Parameters

sWorkspaceId – Id of the workspace to delete

Returns

True if workspace could be deleted, False otherwise

getWorkspaceIdByName

`wasdi.getWorkspaceIdByName(sName)`

Get Id of a Workspace from the name

Parameters

sName – Workspace Name

Returns

the WorkspaceId as a String, '' if there is any error

getWorkspaceOwnerByName

`wasdi.getWorkspaceOwnerByName(sName)`

Get user Id of the owner of Workspace from the name

Parameters

sName – Name of the workspace

Returns

the userId as a String, '' if there is any error

getWorkspaceOwnerByWsId

`wasdi.getWorkspaceOwnerByWsId(sWsId)`

Get user Id of the owner of Workspace from the Workspace Id

Parameters

sWsId – Workspace Id

Returns

the userId as a String, '' if there is any error

getWorkspaceUrlByWsId

`wasdi.getWorkspaceUrlByWsId(sWsId)`

Get Base Url of a Workspace from the Workspace Id

Parameters

sWsId – Workspace Id

Returns

the Workspace Base Url as a String, “” if there is any error

openWorkspaceById

`wasdi.openWorkspaceById(sWorkspaceId)`

Open a workspace by Id

Parameters

sWorkspaceId – Workspace Id

Returns

the WorkspaceId as a String, “” if there is any error

openWorkspace

`wasdi.openWorkspace(sWorkspaceName)`

Open a workspace

Parameters

sWorkspaceName – Workspace Name

Returns

the WorkspaceId as a String, “” if there is any error

getProductsByWorkspace

`wasdi.getProductsByWorkspace(sWorkspaceName)`

Get the list of products in a workspace by Name

Parameters

sWorkspaceName – Name of the workspace

Returns

the list is an array of string. Can be empty if there is any error

getProductsByWorkspaceId

`wasdi.getProductsByWorkspaceId(sWorkspaceId)`

Get the list of products in a workspace by Id

Parameters

sWorkspaceId – Workspace Id

Returns

the list is an array of string. Can be empty if there is any error

getProductsByActiveWorkspace

`wasdi.getProductsByActiveWorkspace()`

Get the list of products in the active workspace

Returns

the list is an array of string. Can be empty if there is any error

getPath

`wasdi.getPath(sFile="")`

Get Local File Path. If the file exists and needed the file will be automatically downloaded. Returns the full local path where to read or write sFile

Param

sFile name of the file

Returns

Local path where to read or write sFile

getFullProductPath

`wasdi.getFullProductPath(sProductName)`

Get the full local path of a product given the product name. If auto download is true and the code is running locally, WASDI will download the image and keep the file on the local PC Use the output of this API to get the full path to open a file

Parameters

sProductName – name of the product to get the path open (WITH the final extension)

Returns

local path of the Product File

getSavePath

`wasdi.getSavePath()`

Get the local base save path for a product. To save use this path + fileName. Path already include '/' as last char

Returns

local path to use to save files (with '/' as last char)

getProcessStatus

`wasdi.getProcessStatus(sProcessId, sDestinationWorkspaceUrl=None)`

get the status of a Process

Parameters

- **sProcessId** – Id of the process to query
- **sDestinationWorkspaceUrl** – allow to ask for a status of a Process That is not in the actual Active Node

Returns

the status or 'ERROR' if there was any error

STATUS are CREATED, RUNNING, STOPPED, DONE, ERROR, WAITING, READY

deleteProduct

`wasdi.deleteProduct(sProduct)`

Delete a Product from a Workspace NOTE: the method DOES NOT delete the pyshical file on your local Disk if the app is running in your environment.

Parameters

sProduct – Name of the product to delete (WITH EXTENSION)

Returns

True if the file has been deleted, False if there was any error

mosaic

`wasdi.mosaic(asInputFiles, sOutputFile, iNoDataValue=None, iIgnoreInputValue=None, fPixelSizeX=None, fPixelSizeY=None, bAsynch=False)`

Creates a mosaic out of a set of images

Parameters

- **asInputFiles** – List of input files to mosaic
- **sOutputFile** – Name of the mosaic output file
- **iNoDataValue** – Value to use as noData. Use -1 to ignore
- **iIgnoreInputValue** – Value to ignore from the input files of the mosaic. Use -1 to ignore
- **fPixelSizeX** – double value of the output pixel X resolution
- **fPixelSizeY** – double value of the output pixel Y resolution
- **bAsynch** – True to return after the triggering, False to wait the process to finish

Returns

Process ID is asynchronous execution, end status otherwise. An empty string is returned in case of failure

printStatus

`wasdi.printStatus()`

Prints status

searchEOImages

`wasdi.searchEOImages(sPlatform, sDateFrom=None, sDateTo=None, fULLat=None, fULLon=None, fLRLat=None, fLRLon=None, sProductType=None, iOrbitNumber=None, sSensorOperationalMode=None, sCloudCoverage=None, sProvider=None, oBoundingBox=None, aoParams=None, sFileName=None)`

Search EO images

Parameters

- **sPlatform** – satellite platform:(S1|S2|S3|S5P|VIIRS|L8|ENV|ERA5)

- **sDateFrom** – initial date YYYY-MM-DD
- **sDateTo** – final date YYYY-MM-DD
- **fULLat** – Latitude of Upper-Left corner
- **fULLon** – Longitude of Upper-Left corner
- **fLRLat** – Latitude of Lower-Right corner
- **fLRLon** – Longitude of Lower-Right corner
- **sProductType** – type of EO product; Can be null. FOR “S1” -> “SLC”, “GRD”, “OCN”. FOR “S2” -> “S2MSI1C”, “S2MSI2Ap”, “S2MSI2A”. FOR “VIIRS” -> “VIIRS_1d_composite”, “VIIRS_5d_composite”. FOR “L8” -> “L1T”, “L1G”, “L1GT”, “L1GS”, “L1TP”. For “ENVI” -> “ASA_IM_0P”, “ASA_WS_0P”
- **iOrbitNumber** – orbit number
- **sSensorOperationalMode** – sensor operational mode
- **sCloudCoverage** – interval of allowed cloud coverage, e.g. “[0 TO 22.5]”
- **sProvider** – WASDI Data Provider to query (AUTO|LSA|ONDA|CREODIAS|SOBLOO|VIIRS|SENTINEL). None means default node provider = AUTO.
- **oBoundingBox** – alternative to the float lat-lon corners: an object expected to have these attributes: oBoundingBox[“northEast”][“lat”], oBoundingBox[“southWest”][“lng”], oBoundingBox[“southWest”][“lat”], oBoundingBox[“northEast”][“lng”]
- **aoParams** – dictionary of search keys to add to the query. The system will add key=value to the query sent to WASDI. The parameters for each collection can be found on the on line documentation
- **sFileName** – name of a specific file to search

Returns

a list of results represented as a Dictionary with many properties. The dictionary has the “fileName” and “relativeOrbit” properties among the others

setVerbose

`wasdi.setVerbose(bVerbose)`

Sets verbosity :param boolean bVerbose: False non verbose, True verbose :return:

setParametersDict

`wasdi.setParametersDict(aoParams)`

Get the full Params Dictionary

Parameters

aoParams – dictionary of Parameters

Returns

a dictionary containing the parameters

setUser

`wasdi.setUser(sUser)`

Sets the WASDI User

Parameters

sUser – WASDI UserID

Returns

setPassword

`wasdi.setPassword(sPassword)`

Set the WASDI Password

setSessionId

`wasdi.setSessionId(sSessionId)`

Set the WASDI Session

setParametersFilePath

`wasdi.setParametersFilePath(sParamPath)`

Set The Parameters JSON File Path

Param

sParamPath Local Path of the parameters file

setBasePath

`wasdi.setBasePath(sBasePath)`

Set the local Base Path for WASDI

Parameters

sBasePath – local WASDI base Path. If not set, by default WASDI uses [USERHOME].wasdi

getBasePath

`wasdi.getBasePath()`

Get the local Base Path for WASDI

Returns

local base path for WASDI

setBaseUrl

`wasdi.setBaseUrl(sBaseUrl)`

Set the WASDI API URL

Parameters

sBaseUrl – WASDI API URL

setProcessPayload

`wasdi.setProcessPayload(sProcessId, data)`

Saves the Payload of a process

Parameters

- **sProcessId** – Id of the process
- **data** – data to write in the payload. Suggestion to use a JSON

Returns

the updated status as a String or “ if there was any problem

setPayload

`wasdi.setPayload(data)`

Sets the payload of the current running process. The payload is saved only when run on Server. In local mode is just a print.

Parameters

data – data to save in the payload. Suggestion is to use JSON

return None

getProcessorPayload

`wasdi.getProcessorPayload(sProcessObjId, bAsJson=False)`

Retrieves the payload

Parameters

- **sProcessObjId** – a valid processor obj id
- **bAsJson** – flag to indicate whether the payload is a json object: if True, then a dictionary is returned

Returns

the processor payload if present, None otherwise

getProcessorPayloadAsJson

`wasdi.getProcessorPayloadAsJson(sProcessObjId)`

Retrieves the payload in json format using `getProcessorPayload`

Parameters

sProcessObjId – a valid processor obj id

Returns

the processor payload if present as a dictionary, None otherwise

setSubPid

`wasdi.setSubPid(sProcessId, iSubPid)`

Set the sub pid

Parameters

- **sProcessId** – Id of the process
- **iSubPid** – PID of the physical process

Returns

the updated status as a String or “ if there was any problem

saveFile

`wasdi.saveFile(sFileName)`

Ingest a new file in the Active WASDI Workspace. The method takes a file saved in the workspace root (see `getSaveFilePath`) not already added to the WS To work be sure that the file is on the server

Param

Name of the file to add to the workspace

Returns

Status of the operation

updateProgressPerc

`wasdi.updateProgressPerc(iPerc)`

Update the actual progress Percentage of the processor

Parameters

iPerc – new Percentage. Use a value between 0 and 100 to set it. The value must be an integer

Returns

updated status of the process or “ if there was any error

updateProcessStatus

`wasdi.updateProcessStatus(sProcessId, sStatus, iPerc=-1)`

Update the status of a process

Parameters

- **sProcessId** – Id of the process to update.
- **sStatus** – Status of the process. Can be CREATED, RUNNING, STOPPED, DONE, ERROR, WAITING, READY
- **iPerc** – percentage of complete of the processor. Use -1 to ignore Percentage. Use a value between 0 and 100 to set it.

Returns

the updated status as a String or “ if there was any problem

updateStatus

`wasdi.updateStatus(sStatus, iPerc=-1)`

Update the status of the running process

Parameters

- **sStatus** – new status. Can be CREATED, RUNNING, STOPPED, DONE, ERROR, WAITING, READY
- **iPerc** – new Percentage.-1 By default, means no change percentage. Use a value between 0 and 100 to set it.

Returns

the updated status as a String or “ if there was any problem

waitProcess

`wasdi.waitProcess(sProcessId, sDestinationWorkspaceUrl=None)`

Wait for a process to End

Parameters

sProcessId – Id of the process to wait

Returns

output status of the process

waitProcesses

`wasdi.waitProcesses(asProcIdList)`

Wait for a list of processes to wait. The list of processes is an array of strings, each with a proc id to wait

Parameters

asProcIdList – list of strings, procId, to wait

Returns

list of strings with the same number of elements in input, with the exit status of the processes

_downloadFile

`wasdi._downloadFile(sFileName)`

Download a file from WASDI

Parameters

sFileName – file to download

Returns

None

wasdiLog

`wasdi.wasdiLog(sLogRow)`

Write one row of Log

Parameters

sLogRow – text to log

Returns

None

fileExistsOnWasdi

`wasdi.fileExistsOnWasdi(sFileName)`

checks if a file already exists on WASDI or not

Parameters

sFileName – file name with extension

Returns

True if the file exists, False otherwise

importProductByFileUrl

`wasdi.importProductByFileUrl(sFileUrl=None, sName=None, sBoundingBox=None, sProvider=None, sVolumeName=None, sVolumePath=None)`

Imports a product from a Provider in WASDI, starting from the File URL.

Parameters

- **sFileUrl** – url of the file to import
- **sName** – Name of the file to import as returned by the Data Provider
- **sBoundingBox** – declared bounding box of the file to import
- **sProvider** – WASDI Data Provider to use. Use None for Default
- **sVolumeName** – if the file is in a Volume, the name of the volume
- **sVolumePath** – if the file is in a Volume, the path of the file in the volume

Returns

execution status as a STRING. Can be DONE, ERROR, STOPPED.

asynchImportProductByFileUrl

`wasdi.asynchImportProductByFileUrl(sFileUrl=None, sName=None, sBoundingBox=None, sProvider=None, sVolumeName=None, sVolumePath=None)`

Asynch Import of a product from a Provider in WASDI, starting from file URL

Parameters

- **sFileUrl** – url of the file to import as returned by the data provider
- **sName** – Name of the file to import as returned by the Data Provider
- **sBoundingBox** – declared bounding box of the file to import
- **sProvider** – WASDI Data Provider. Use None for default
- **sVolumeName** – if the file is in a Volume, the name of the volume
- **sVolumePath** – if the file is in a Volume, the path of the file in the volume

Returns

ProcessId of the Download Operation, “DONE” if the file is imported or “ERROR” if there is any problem

importProduct

`wasdi.importProduct(oProduct, sProvider=None)`

Imports a product from a Provider in WASDI starting from the object returned by searchEOImages

Parameters

- **oProduct** – product dictionary as returned by searchEOImages
- **sProvider** – WASDI Data Provider. Use None for default

Returns

execution status as a STRING. Can be DONE, ERROR, STOPPED.

asynchImportProduct

`wasdi.asynchImportProduct(oProduct, sProvider=None)`

Asynch Import a product from a Provider in WASDI starting from the object returned by searchEOImages

Parameters

- **oProduct** – product dictionary as returned by searchEOImages
- **sProvider** – WASDI Data Provider. Use None for default

Returns

ProcessId of the Download Operation or “ERROR” if there is any problem

importProductList

`wasdi.importProductList(aoProducts, sProvider=None)`

Imports a list of product from a Provider in WASDI starting from an array of objects returned by searchEOImages

Parameters

- **aoProducts** – Array of product dictionary as returned by searchEOImages
- **sProvider** – WASDI Data Provider. Use None for default

Returns

execution status as an array of STRINGS, one for each product in input. Can be CREATED, DONE, ERROR, STOPPED, WAITING, READY

asynchImportProductList

`wasdi.asynchImportProductList(aoProducts, sProvider=None)`

Asynch Import a list of product from a Provider in WASDI starting from an array of objects returned by searchEOImages

Parameters

- **aoProducts** – Array of product dictionary as returned by searchEOImages
- **sProvider** – WASDI Data Provider. Use None for default

Returns

array of the ProcessId of the Download Operations. An element can be “ERROR” if there was any problem

asynchAddFileToWASDI

`wasdi.asynchAddFileToWASDI(sFileName, sStyle='')`

Triggers the ingestion of File Name in the workspace

Param

sFileName: Name (with extension) of the file to add

Parameters

sStyle – name of a valid WMS style

Returns

Process Id of the ingestion

importAndPreprocess

`wasdi.importAndPreprocess(aoImages, sWorkflow, sPreProcSuffix='_proc.tif', sProvider=None)`

Imports in WASDI and apply a SNAP Workflow to an array of EO Images as returned by searchEOImages

Parameters

- **aoImages** – array of images to import as returned by searchEOImages
- **sWorkflow** – name of the workflow to apply to each imported images
- **sProvider** – WASDI Data Provider. Use None for default
- **sPreProcSuffix** – suffix to use for the name of the output of the workflows

Returns**asynchExecuteProcessor**

`wasdi.asynchExecuteProcessor(sProcessorName, aoParams={})`

Legacy: use `executeProcessor` Executes a WASDI Processor asynchronously. The method try up to three time if there is any problem.

Parameters

- **sProcessorName** – WASDI processor name
- **aoParams** – a dictionary of parameters for the processor

Returns

the Process Id if every thing is ok, '' if there was any problem

executeProcessor

`wasdi.executeProcessor(sProcessorName, aoProcessParams)`

Executes a WASDI Processor asynchronously. The method try up to three time if there is any problem.

Parameters

- **sProcessorName** – WASDI processor name
- **aoParams** – a dictionary of parameters for the processor

Returns

the Process Id if every thing is ok, '' if there was any problem

_uploadFile

`wasdi._uploadFile(sFileName)`

Uploads a file to WASDI

Parameters

sFileName – name of file inside working directory OR path to file RELATIVE to working directory

Returns

True if succeeded, False otherwise

subset

`wasdi.subset(sInputFile, sOutputFile, dLatN, dLonW, dLatS, dLonE)`

Creates a Subset of an image:

Parameters

- **sInputFile** – Input file
- **sOutputFile** – Output File
- **dLatN** – Latitude north of the subset
- **dLonW** – Longitude west of the subset

- **dLatS** – Latitude South of the subset
- **dLonE** – Longitude Est of the subset

multiSubset

`wasdi.multiSubset(sInputFile, asOutputFiles, adLatN, adLonW, adLatS, adLonE, bBigTiff=False)`

Creates a Many Subsets from an image. MAX 10 TILES PER CALL

Parameters

- **sInputFile** – Input file
- **sOutputFile** – Array of Output File Names
- **dLatN** – Array of Latitude north of the subset
- **dLonW** – Array of Longitude west of the subset
- **dLatS** – Array of Latitude South of the subset
- **dLonE** – Array of Longitude Est of the subset

executeWorkflow

`wasdi.executeWorkflow(asInputFileNames, asOutputFileNames, sWorkflowName, aoTemplateParams=None)`

Execute a SNAP Workflow available in WASDI (you can use WASDI to upload your SNAP Graph XML and use from remote)

Parameters

- **asInputFileNames** – array of the inputs of the workflow. Must correspond to the number of inputs of the workflow.
- **asOutputFileNames** – array of the outputs of the workflow. Must correspond to the number of inputs of the workflow.
- **sWorkflowName** – Name of the workflow to run
- **aoTemplateParams** – Dictionary with strings KEY-VALUE that will be used to fill potential parameters in the Workflow XML. Wasdi will search the XML for the strings in the keys and replace with the value here provided

Returns

final status of the executed Workflow

asynchExecuteWorkflow

`wasdi.asynchExecuteWorkflow(asInputFileNames, asOutputFileNames, sWorkflowName, aoTemplateParams=None)`

Trigger the asynch execution of a SNAP Workflow available in WASDI (you can use WASDI to upload your SNAP Graph XML and use from remote)

Parameters

- **asInputFileNames** – array of the inputs of the workflow. Must correspond to the number of inputs of the workflow.
- **asOutputFileNames** – array of the outputs of the workflow. Must correspond to the number of inputs of the workflow.

- **sWorkflowName** – Name of the workflow to run
- **aoTemplateParams** – Dictionary with strings KEY-VALUE that will be used to fill potential parameters in the Workflow XML. Wasdi will search the XML for the strings in the keys and replace with the value here provided

Returns

Process Id of the started workflow

asynchMosaic

```
wasdi.asynchMosaic(asInputFiles, sOutputFile, iNoDataValue=None, iIgnoreInputValue=None,  
                  fPixelSizeX=None, fPixelSizeY=None)
```

Start a mosaic out of a set of images in asynch way

Parameters

- **asInputFiles** – List of input files to mosaic
- **sOutputFile** – Name of the mosaic output file
- **iNoDataValue** – Value to use as noData. Use -1 to ignore
- **iIgnoreInputValue** – Value to ignore from the input files of the mosaic. Use -1 to ignore
- **fPixelSizeX** – double value of the output pixel X resolution
- **fPixelSizeY** – double value of the output pixel Y resolution

Returns

Process ID is asynchronous execution, end status otherwise. An empty string is returned in case of failure

copyFileToSftp

```
wasdi.copyFileToSftp(sFileName, bAsynch=None, sRelativePath=None)
```

Copy a file from a workspace to the WASDI user's SFTP Folder

Parameters

- **sFileName** – File name (with extension, without path) to copy in the SFTP folder
- **bAsynch** – True to return after the triggering, False to wait the process to finish

Returns

Process ID is asynchronous execution, end status otherwise. An empty string is returned in case of failure

_log

`wasdi._log(sLog)`

Internal Log function

Parameters

sLog – text row to log

_getStandardHeaders

`wasdi._getStandardHeaders()`

Get the standard headers for a WASDI API Call, setting also the session token

Returns

dictionary of headers to add to the REST API

_loadConfig

`wasdi._loadConfig(sConfigFilePath)`

Loads configuration from given file

Parameters

sConfigFilePath – a string containing a path to the configuration file

_loadParams

`wasdi._loadParams()`

Loads parameters from file, if specified in configuration file

_unzip

`wasdi._unzip(sAttachmentName, sPath)`

Unzips a file

Parameters

- **sAttachmentName** – filename to unzip
- **sPath** – both the path where the file is and where it must be unzipped

Returns

None

_waitForResume

_normPath

`wasdi._normPath(sPath)`

Normalizes path by adjusting separator

Parameters

sPath – a path to be normalized

Returns

the normalized path

`_internalAddFileToWASDI`**`_internalExecuteWorkflow`**

`wasdi._internalExecuteWorkflow(asInputFileNames, asOutputFileNames, sWorkflowName, bAsynch=False, aoTemplateParams=None)`

Internal call to execute workflow

Parameters

- **asInputFileNames** – name of the file in input (string WITH extension) or array of strings of the files in input (WITH extension)
- **asOutputFileNames** – name of the file in output (string WITH extension) or array of strings of the files in output (WITH extension)
- **sWorkflowName** – name of the SNAP workflow uploaded in WASDI
- **bAsynch** – true to run asynch, false to run synch
- **aoTemplateParams** – Dictionary with strings KEY-VALUE that will be used to fill potential parameters in the Workflow XML. Wasdi will search the XML for the strings in the keys and replace with the value here provided

Returns

processID if asynch, status of the executed process if synch, empty string in case of failure

`_fileOnNode`

`wasdi._fileOnNode(sFileName)`

checks if a file already exists on the node of the workspace or not

Parameters

sFileName – file name with extension

Returns

True if the file exists, False otherwise

`_getDefaultCRS`

5.5.3 Changelog

5.6 Javascript WasdiLib

5.6.1 Methods

loadConfig

loadConfig(*configFile*, *parametersFile*)

Loads configuration and parameters. If no filename is specified, the method attempts to load config.json and parameters.json files from the root level URL of the developed application. The config file can be also hosted on an external URL.

Arguments

- **configFile** – a JSON containing all the required information to login to WASDI, please check repository for a complete example
- **parametersFile** – a JSON containing the parameters that can be used during the launch of the application

loadParameters**loadParameters**(*filename*)

Loads a json containing the parameters which are then imported in a dedicated field. If filename is not specified the methods search for “parameters.json”, as default

Arguments

- **filename** – the file name of the JSON containing the parameters

login**login**(*sUserName*, *sPassword*)

Api call for the login to WASDI services. Valid credential must be available.

Arguments

- **sUserName** – The username, corresponding to the e-mail used during registration
- **sPassword** – The selected password

checkBaseUrl**checkBaseUrl**()

Util methods to check initialization of the current base URL. Can be used to verify the WASDI service on the selected node.

Returns

boolean – true, if the connection is ok, false instead

helloWasdiWorld**helloWasdiWorld**(*noOutput*)

Test method to check wasdi instance, with a tiny bit of developer’s traditions. Used across this library to check the connection state.

Arguments

- **noOutput** – if true, the method doesn’t output the response on the console

openWorkspace

openWorkspace(workspaceID)

Open a workspace and set it as active workspace, by using its Id

Arguments

- **workspaceID** – The id of the selected workspace

Returns

Object|number|string|Object|* –

createWorkspace

createWorkspace(wsName)

Create a new workspace for the active user.

Arguments

- **wsName** – The workspace name, if the name is already used WADI will append a further numeric identifier (like the OS for new folders)

openWorkspaceById

openWorkspaceById(workspaceID)

Opens a workspace and set it as active workspace. The active workspace is the one used for the following operations, like launch a processor or execute a workflow.

Arguments

- **workspaceID** – The id of the selected workspace

Returns

Object|number|string|Object|* –

getWorkspaces

getWorkspaces()

Retrieves the list of Workspace of the current logged user.

getProductsByActiveWorkspace

getProductsByActiveWorkspace()

Retrieve a list of workspace of the current logged user.

executeProcessor

executeProcessor(*appName, jsonParameters*)

Launch a process in the current workspace. Check `getDeployed` method to obtain a list of the available processors

Arguments

- **appName** – a String containing the name of the selected application
- **jsonParameters** – a JSON containing the parameters for the application, please check the app on WASDI for a specific reference

getProcessStatus

getProcessStatus(*processId*)

Retrieves the process status of a process, identified by its `processId`. The response contains, among other data, the status and the progress percentage

Arguments

- **processId** – the string containing the `processId` selected

Returns

the process Object

setProcessPayload

setProcessPayload(*sProcessId, data*)

Set the payload of a process, identified by its `processId`

Arguments

- **sProcessId** – the `processId` to add the payload
- **data** – JSON string containing the payload

getDeployed

getDeployed()

Retrieves a list of applications available on the WASDI marketplace. The response is an array of strings that can be used to launch the particular application

publishBand

publishBand(*fileName, bandName*)

Publish a band of the particular product selected. To obtain a list of available bands, a function that retrieves the product details can be used.

Arguments

- **fileName** – The product name in the current workspace
- **bandName** – The band that needs to be published

getLayerWMS

getLayerWMS(*sProductName*, *sBand*)

Returns an object containing 2 string with WMS parameters: - server e.g. “[https://\[URL\]/geoserver/ows?](https://[URL]/geoserver/ows?)” if using Geoserver - layerId

Arguments

- **sProductName** – the Product name
- **sBand** – The band required

5.7 Create a config.json file

For developing with WASDI in python, you need to create a config.json file.

Note: The configuration file contains your credentials and some additional information to get WASDI started: never share it with others! It is required only for developing on your PC, so do not upload it to WASDI when deploying or updating an application

5.7.1 Prerequisites

To run this code you need:

- A running Python 3.x Environment
- A valid WASDI Account

If this is not clear, you probably need to take a look to the [Python Tutorial](#) before.

5.7.2 Recipe

The basic config.json file must contain:

- your username, which usually corresponds to the email you used
- your password (only you are supposed to know this)
- the workspace ID (suggested) or the workspace name you intend to work in. The workspace ID can be obtained looking at the URL in the editor, it's going to look like this: <https://www.wasdi.net/#!/71805896-654b-468c-8fc5-5d2ad6ba61f3/editor> -> in this case, *71805896-654b-468c-8fc5-5d2ad6ba61f3* is the string you are looking for
- path to the parameters file. Again, this is another JSON file used just for development purposes. Assuming it is called *params.json*, and that you saved it in the top folder with you your *myProcessor.py* and *config.json* files, the value would be “*./params.json*”

```
{
  "USER": "yourusername@goes.here",
  "PASSWORD": "your secret password goes here",
  "WORKSPACEID": "71805896-654b-468c-8fc5-5d2ad6ba61f3",
  "PARAMETERSFILEPATH": "./params.json"
}
```

Note: Quick, wasn't it? Try, and [reach out if you need help](#).

5.8 Python Application Skeleton

The following code is the basic structure of a Python Application.

5.8.1 Prerequisites

To run this code you need:

- A running Python 3.x Environment
- A valid WASDI Account
- A valid Config file

If this is not clear, you probably need to take a look to the [Python Tutorial](#) before.

5.8.2 Recipe

This is the basic structure of a WASDI Application.

Note: The main file **MUST** be called `myProcessor.py`. You can then add all the libraries, files and module you may want to code or include

```
import wasdi

def run():
    wasdi.wasdiLog("Here I can start to code")

if __name__ == '__main__':
    wasdi.init("./config.json")
    run()
```

What it does:

- import the library
- handle `__main__` in the file
- define a `run()` method
- initialize the lib
- call the `run()` method

Note: This structure is mandatory if you plan to deploy your application in WASDI. To use the library only as a client, this is not necessary.

5.9 Read Parameters

The following code is the snippet to read parameters in WASDI.

5.9.1 Prerequisites

To run this code you need:

- A running Python 3.x Environment
- A valid WASDI Account
- A valid Config file
- A valid params.json file

If this is not clear, you probably need to take a look at the [Python Tutorial](#) before.

5.9.2 Recipe

Note: Assume we have a params.json file (configured in config.json)

This is our sample params.json file.

```
{
  "STRING_PARAM": "SAMPLE",
  "INT_PARAM": 2,
  "BBOX": {
    "northEast": {
      "lat": 20.1,
      "lng": -71.4
    },
    "southWest": {
      "lat": 17.9,
      "lng": -74.9
    }
  },
  "DATE": "2024-01-01"
}
```

This is the code used to read Parameters.

```
# Read the String Parameter
sStringParameter = wasdi.getParameter("STRING_PARAM")
# Read the String Parameter, with a Default value if the param is missing in the params.
# ↪ json file
sStringParameterWithDefault = wasdi.getParameter("STRING_PARAM", "My Default")
# Read the Area of Interest
oBbox = wasdi.getParameter("BBOX", None)
# Read the integer value without any default
iIntegerValue = wasdi.getParameter("INT_PARAM")
# Read the string-formatted Date
```

(continues on next page)

(continued from previous page)

```
sDateString = wasdi.getParameter("DATE")

#This method return a Key-Value Dictionary with all your parameters
aoAllParametersDictionary = wasdi.getParametersDict()
```

What it does:

- reads different parameters
- reads the full parameters dictionary at once

Note: The developer can decide whatever is needed in the params.json file. If you decide to use the [WASDI User Interface](#) your parameters will be generated automatically by WASDI.

Note: With the [WASDI User Interface](#) you can use the [renderAsStrings](#) flag to ask WASDI to get all your parameters in String Format. In this case, you will be responsible to convert your data in your code.

Note: The Boundig Box Format used here is the same one used by the User Interface when [renderAsStrings](#) is missing or false. The Boundig Box fromat when [renderAsStrings](#): true is **“NORTH,WEST,SOUTH,EAST”**.

Note: The Date is formatted by the User Interface as “YYYY-MM-DD”.

5.10 Search Sentinel-1 Images

The following code shows how to search S1 Images.

5.10.1 Prerequisites

To run this code you need:

- A running Python 3.x Environment
- A valid WASDI Account
- A [valid Config file](#)

If this is not clear, you probably need to take a look to the [Python Tutorial](#) before.

5.10.2 Recipe

Note: Assume you have at least one workspace and you have configured it in the config.json file.

These are different samples of Sentinel 1 Search. The mandatory fields to search are:

- Mission Type
- Start Date
- End Date
- Product Type
- Bounding Box

```
# Create the Bounding Box Object: usually you will take it from the parameters
oBBBox = wasdi.getParameter("BBOX", None)

# If it is null we show here how to initialize manually
if oBBBox is None:
    oBBBox = {"northEast": {}, "southWest": {}}
    oBBBox["northEast"]["lat"] = 20.1
    oBBBox["northEast"]["lng"] = 44.4
    oBBBox["southWest"]["lat"] = 19.3
    oBBBox["southWest"]["lng"] = 43.2

# Set Start Date
sStartDate = wasdi.getParameter("START_DATE", "2023-01-01")
# Set End Date
sEndDate = wasdi.getParameter("END_DATE", "2023-01-31")

# Start Search GRD Images
aoProductsFoundArray = wasdi.searchEOImages("S1", sStartDate, sEndDate, sProductType="GRD",
↳, oBoundingBox=oBBBox)

# The result is an array of Objects. Each Object is a Dictionary.

# If we have results
if len(aoProductsFoundArray) > 0:

    # We just loop on the results and explore some properties
    for oFoundImage in aoProductsFoundArray:
        # This is where to read the relative Orbit
        iOrbit = oFoundImage["properties"]["relativeorbitnumber"]
        # This is the name of the file
        sFileName = oFoundImage["fileName"]
        # There are many other properties, depending by the Provider and the Mission,
↳that can be explored

# Now lets search SLC Images
aoSLCFoundArray = wasdi.searchEOImages("S1", sStartDate, sEndDate, sProductType="SLC",
↳oBoundingBox=oBBBox)
wasdi.wasdiLog("Found " + str(len(aoSLCFoundArray)) + " SLC Images")
```

(continues on next page)

(continued from previous page)

```

# For Sentinel 1, we can also filter the Relative Orbit
iRelativeOrbit = 43
aoSLCPerOrbitFoundArray = wasdi.searchEOImages("S1", sStartDate, sEndDate, sProductType=
↳ "SLC", oBoundingBox=oBBox, iOrbitNumber=iRelativeOrbit)
wasdi.wasdiLog("Found " + str(len(aoSLCPerOrbitFoundArray)) + " SLC Images in orbit " +
↳ str(iRelativeOrbit))

# If we have a String Bounding Box...
sBBox = "20.1,43.2,19.3,44.4"
# We can convert it in the object
oBoundingBox = wasdi.bboxStringToObject(sBBox)
# Or we can also use directly lat and lon in the search:
aoSLCWithLatLonFound = wasdi.searchEOImages("S1", sStartDate, sEndDate, fULLat=20.1,
↳ fULLon=43.2, fLRLat=19.3, fLRLon=44.4, sProductType="SLC")
wasdi.wasdiLog("Found " + str(len(aoSLCWithLatLonFound)) + " SLC Images")

```

What it does:

- Initializes the input variable needed.
- Starts searching for S1 GRD Images
- Loops over the results and accesses some properties
- Searches for SLC Images
- Searches for SLC Images adding the relative orbit filter
- Searches for GRD Images using the lat lon values and not the Bounding Box Object

Note: The developer can decide what is needed in the params.json file. If you decide to use the [WASDI User Interface](#) your parameters will be generated automatically by WASDI.

Note: With the [WASDI User Interface](#) you can use the `renderAsStrings` flag to ask WASDI to get all your parameters in String Format. In this case you will be responsible to convert your data in your code.

Note: The Bounding Box Format used here is the one used by the User Interface when `renderAsStrings` is missing or false. The Bounding Box format when `renderAsStrings: true` is “NORTH, WEST, SOUTH, EAST”.

Note: The Date is formatted by the User Interface as “YYYY-MM-DD”.

5.11 Search Sentinel-2 Images

The following code shows how to search S2 Images

5.11.1 Prerequisites

To run this code you need:

- A running Python 3.x Environment
- A valid WASDI Account
- A [valid Config file](#)

If this is not clear, you probably need to take a look to the [Python Tutorial](#) before.

5.11.2 Recipe

Note: Assume you have at least one workspace and you configured it in the config.json file

These are different samples of Sentinel 2 Search. The mandatory fields to search are:

- Mission Type
- Start Date
- End Date
- Product Type
- Bounding Box

```
# Create the Bounding Box Object: usually you will take it from the parameters
oBBox = wasdi.getParameter("BBOX", None)

# If it is null we show here how to initialize manually
if oBBox is None:
    oBBox = {"northEast": {}, "southWest": {}}
    oBBox["northEast"]["lat"] = 20.1
    oBBox["northEast"]["lng"] = 44.4
    oBBox["southWest"]["lat"] = 19.3
    oBBox["southWest"]["lng"] = 43.2

# Set Start Date
sStartDate = wasdi.getParameter("START_DATE", "2023-01-01")
# Set End Date
sEndDate = wasdi.getParameter("END_DATE", "2023-01-10")

# Start Search S2 MSI1C Images (Level 1)
aoProductsFoundArray = wasdi.searchEOImages("S2", sStartDate, sEndDate, sProductType=
↳ "S2MSI1C", oBoundingBox=oBBox)

# The result is an array of Objects. Each Object is a Dictionary.
```

(continues on next page)

(continued from previous page)

```

# If we have results
if len(aoProductsFoundArray) > 0:

    # We just loop on the results and log file names
    for oFoundImage in aoProductsFoundArray:
        # This is the name of the file
        sFileName = oFoundImage["fileName"]
        wasdi.wasdiLog("Found " + sFileName)
        # There are many other properties, depending by the Provider and the Mission,
        ↪ that can be explored

# Now lets search L2 Images
aoL2FoundArray = wasdi.searchEOImages("S2", sStartDate, sEndDate, sProductType="S2MSI2A",
    ↪ oBoundingBox=oBBox)
wasdi.wasdiLog("Found " + str(len(aoL2FoundArray)) + " S2MSI2A Images")

# For Sentinel 1, we can also filter on the Cloud Coverage
sCloudCover = "[0 TO 50]"
aoL2CloudCoverFound = wasdi.searchEOImages("S2", sStartDate, sEndDate, sProductType=
    ↪ "S2MSI2A", oBoundingBox=oBBox, sCloudCoverage=sCloudCover)
wasdi.wasdiLog("Found " + str(len(aoL2CloudCoverFound)) + " S2MSI2A Images with
    ↪ CloudCover = " + sCloudCover)

# If we have a String Bounding Box...
sBBox = "20.1,43.2,19.3,44.4"
# We can convert it in the object
oBoundingBox = wasdi.bboxStringToObject(sBBox)
# Or we can also use directly lat and lon in the search:
aoL2LatLonFound = wasdi.searchEOImages("S2", sStartDate, sEndDate, fULLat=20.1,
    ↪ fULLon=43.2, fLRLat=19.3, fLRLon=44.4, sProductType="S2MSI2A")
wasdi.wasdiLog("Found " + str(len(aoL2LatLonFound)) + " S2MSI2A Images")

```

What it does:

- Initializes the input variable needed.
- Starts searching S2 L1 Images
- Loop the results and print file names
- Searches L2 Images
- Searches L2 Images; adding the cloud coverage
- Searches L2 Images; using the lat lon values and not the Bounding Box Object

Note: The developer can decide whatever is needed in the params.json file. If you decide to use the [WASDI User Interface](#) your parameters will be generated automatically by WASDI.

Note: With the [WASDI User Interface](#) you can use the [renderAsStrings](#) flag to ask WASDI to get all your parameters in String Format. In this case you will be responsible fir converting your data in your code.

Note: The Bounding Box Format used here is the one used by the User Interface when `renderAsStrings` is missing or false. The Bounding Box format when `renderAsStrings: true` is **“NORTH, WEST, SOUTH, EAST”**.

Note: The Date is formatted by the User Interface as “YYYY-MM-DD”.

5.12 Search Sentinel-3 Images

The following code shows how to search S3 Images

5.12.1 Prerequisites

To run this code you need:

- A running Python 3.x Environment
- A valid WASDI Account
- A valid Config file

If this is not clear, you probably need to take a look to the [Python Tutorial](#) before.

5.12.2 Recipe

Note: Assume you have at least one workspace and you configured it in the `config.json` file

These are different samples of Sentinel 2 Search. The mandatory fields to search are:

- Mission Type
- Start Date
- End Date
- Product Type
- Bounding Box

```
# Create the Bounding Box Object: usually you will take it from the parameters
oBBBox = wasdi.getParameter("BBOX", None)
```

```
# If it is null we show here how to initialize manually
```

```
if oBBBox is None:
```

```
    oBBBox = {"northEast": {}, "southWest": {}}
    oBBBox["northEast"]["lat"] = 20.1
    oBBBox["northEast"]["lng"] = 44.4
    oBBBox["southWest"]["lat"] = 19.3
    oBBBox["southWest"]["lng"] = 43.2
```

```
# Set Start Date getting the parameter from parameters file, fallbacks to 2023-01-01 if_
```

(continues on next page)

(continued from previous page)

```

↪value is not specified
sStartDate = wasdi.getParameter("START_DATE", "2023-01-01")
# Set End Date getting the parameter from parameters file, fallbacks to 2023-01-10 if
↪value is not specified
sEndDate = wasdi.getParameter("END_DATE", "2023-01-10")

# Start Search S3 Images using the automatic provider selection
aoProductsFoundArray = wasdi.searchEOImages("S3", sDateFrom=sStartDate, sDateTo=sEndDate,
↪ sProvider="AUTO",
                                oBoundingBox=oBBox)

# The result is an array of Objects. Each Object is a Dictionary.

# Let's see how many products correspond to our query
wasdi.wasdiLog(f'Your query identified {len(aoProductsFoundArray)} products')

# If we have results
if len(aoProductsFoundArray) > 0:
    # as an example, let's print the filename of the first product we found
    wasdi.wasdiLog(f'{aoProductsFoundArray[0]["fileName"]}')

```

What it does:

- Initializes the input variable needed.
- Start searching for Sentinel-3 Images - automatically selecting the provider
- Loops over the results and prints the file names of the files reporting Land Surface Temperature ([reference data](#))

Note: The developer can decide whatever is needed in the params.json file. If you decide to use the [WASDI User Interface](#) your parameters will be generated automatically by WASDI.

Note: With the [WASDI User Interface](#) you can use the `renderAsStrings` flag to ask WASDI to get all your parameters in String Format. In this case you will be responsible for converting your data in your code.

Note: The Bounding Box Format used here is the one used by the User Interface when `renderAsStrings` is missing or false. The Bounding Box format when `renderAsStrings: true` is **“NORTH, WEST, SOUTH, EAST”**.

Note: The Date is formatted by the User Interface as **“YYYY-MM-DD”**.

5.13 Search Sentinel-5p products

The following code shows how to search Sentinel-5p products in WASDI

5.13.1 Prerequisites

To run this code you need:

- A running Python 3.x Environment
- A valid WASDI Account
- A valid Config file

If this is not clear, you probably need to take a look to the [Python Tutorial](#) before.

5.13.2 Recipe

Note: Assume you have at least one workspace and you configured it in the config.json file

These are different samples of Sentinel-5p Search. The mandatory fields to search are:

- Mission Type
- Start Date
- End Date
- Product Type
- Bounding Box

```
# Create the Bounding Box Object: usually you will take it from the parameters
oBBox = wasdi.getParameter("BBOX", None)

# If it is null we show here how to initialize manually
if oBBox is None:
    oBBox = {"northEast": {}, "southWest": {}}
    oBBox["northEast"]["lat"] = 20.1
    oBBox["northEast"]["lng"] = 44.4
    oBBox["southWest"]["lat"] = 19.3
    oBBox["southWest"]["lng"] = 43.2

# Set Start Date getting the parameter from parameters file, fallbacks to 2023-01-01 if
↳ value is not specified
sStartDate = wasdi.getParameter("START_DATE", "2023-01-01")
# Set End Date getting the parameter from parameters file, fallbacks to 2023-01-10 if
↳ value is not specified
sEndDate = wasdi.getParameter("END_DATE", "2023-01-10")

# product type: Nitrogen Dioxide (NO2)
sProductType = 'L2__NO2__'

# Start Search S3 Images using the automatic provider selection
```

(continues on next page)

(continued from previous page)

```

aoProductsFoundArray = wasdi.searchEOImages("S5P", sDateFrom=sStartDate,
↪sDateTo=sEndDate,
                                     sProductType=sProductType,
                                     sProvider="AUTO", oBoundingBox=oBBox)

# The result is an array of Objects. Each Object is a Dictionary.

# Let's see how many products correspond to our query
wasdi.wasdiLog(f'Your query identified {len(aoProductsFoundArray)} products')

# If we have results
if len(aoProductsFoundArray) > 0:
    # as an example, let's print the filename of the first product we found
    wasdi.wasdiLog(f'{aoProductsFoundArray[0]["fileName"]}')

```

What it does:

- Initialize the input variable needed.
- Start Searching for Sentinel-5p products choosing automatically the provider
- prints the number of results
- as an example, prints the filename of the first product found

Note: The developer can decide whatever is needed in the params.json file. If you will use the [WASDI User Interface](#) your parameters will be generated automatically by WASDI.

Note: With the [WASDI User Interface](#) you can use the `renderAsStrings` flag to ask WASDI to get all your parameters in String Format. In this case you will be responsible to convert your data in your code

Note: The Bounding Box Format Here Used is the one used by the User Interface when `renderAsStrings` is missing or false. The Bounding Box format when `renderAsStrings: true` is “NORTH,WEST,SOUTH,EAST”

Note: The Date is formatted by the User Interface as “YYYY-MM-DD”

5.14 Search Copernicus Marine products

The following code shows how to search for Copernicus Marine products in WASDI

5.14.1 Prerequisites

To run this code you need:

- A running Python 3.x Environment
- A valid WASDI Account
- A valid [Config file](#)

If this is not clear, you probably need to take a look to the [Python Tutorial](#) before.

5.14.2 Recipe

Note: Assume you have at least one workspace and you have configured it in the config.json file.

To search for Copernicus Marine products, the following fields are mandatory:

- Collection: this is always going to be 'CM'
- Start Date
- End Date
- Bounding Box
- product type

Then, it's also mandatory to provide search parameters specific for Copernicus Marine. CM products are usually organized with the following hierarchy:

- dataset
- variables

Additional parameters may apply, in particular, the protocol: SUBS usually tend to work in most cases, but there may be exceptions, so check case by case and don't hesitate to [reach out for support](#).

Let's see an example

```
# Create the Bounding Box Object: usually you will read it from the parameters
oBBox = wasdi.getParameter("BBOX", None)

# If it is null we show here how to initialize manually
if oBBox is None:
    oBBox = {
        "northEast": {
            "lat": 44.2879447888337,
            "lng": 9.5
        },
        "southWest": {
            "lat": 43.5,
            "lng": 8.4
        }
    }

# Set Start Date
sStartDate = wasdi.getParameter("START_DATE", "2023-07-01")
# Set End Date
```

(continues on next page)

(continued from previous page)

```
sEndDate = wasdi.getParameter("END_DATE", "2023-07-31")

sProductType = 'OCEANCOLOUR_MED_BGC_HR_L3_NRT_009_205 - TDS'
aoParams = {
    "dataset": "cmems_obs_oc_med_bgc_tur-spm-chl_nrt_l3-hr-mosaic_PID-m",
    "variables": "CHL",
    "protocol": "SUBS"
}

aoProductsFoundArray = wasdi.searchEOImages(
    sPlatform='CM',
    oBoundingBox=oBBBox, sDateFrom=sStartDate, sDateTo=sEndDate,
    sProvider='AUTO',
    sProductType=sProductType, aoParams=aoParams
)

# Usually, the result is a list of Dictionaries
# In the case of Copernicus Marine, however, the list contain only one element,
↳ encompassing all the data we required:
wasdi.wasdiLog(f'Your query identified {len(aoProductsFoundArray)} products')

# If we have results
if len(aoProductsFoundArray) > 0:
    # let's see the filename corresponding to the product we found:
    wasdi.wasdiLog(f'{aoProductsFoundArray[0]["fileName"]}')

```

What it does:

- Initializes the input variables
- Searches for the corresponding results
- count the results (should always be 1 for CM)
- access a field in the results

Note: The developer can decide what is needed in the params.json file. If you decide to use the [WASDI User Interface](#) your parameters will be generated automatically by WASDI.

Note: With the [WASDI User Interface](#) you can use the `renderAsStrings` flag to ask WASDI to get all your parameters in String Format. In this case you will be responsible to convert your data in your code.

Note: The Bounding Box Format used here is the one used by the User Interface when `renderAsStrings` is missing or false. The Bounding Box format when `renderAsStrings: true` is “NORTH, WEST, SOUTH, EAST”.

Note: The Date is formatted by the User Interface as “YYYY-MM-DD”.

5.15 Search ECOSTRESS products

The following code shows how to search for ECOSTRESS products in WASDI

5.15.1 Prerequisites

To run this code you need:

- A running Python 3.x Environment
- A valid WASDI Account
- A valid Config file

If this is not clear, you probably need to take a look to the [Python Tutorial](#) before.

5.15.2 Recipe

Note: Assume you have at least one workspace and you have configured it in the config.json file.

To search for ECOSTRESS products, the following fields are mandatory:

- Collection: this is always going to be 'ECOSTRESS'
- Start Date
- End Date
- Bounding Box

Another additional search parameters, specific for ECOSTRESS products, is mandatory:

- dataset

The other search parameters, that you can optionally specify, are:

- relativeorbitnumber
- parameterName
- dayNightFlag

Note: The search filters: dataset, relativeorbitnumber and dayNightFlag should be added to a dictionary, that you can then pass to the method `wasdi.searchEOImages` through the optional parameter `aoParams`

Let's see an example of code to search for ECOSTRESS products, by specifying all the available filters.

```
def get_ecostress_products():
    wasdi.wasdiLog("** get_ecostress_products **")

    # Create the Bounding Box Object: usually you will read it from the parameters
    oBBBox = wasdi.getParameter("BBOX", None)

    # If it is null we show here how to initialize manually
    if oBBBox is None:
```

(continues on next page)

(continued from previous page)

```

oBBox = {
    "northEast": {
        "lat": 50,
        "lng": 50
    },
    "southWest": {
        "lat": 44,
        "lng": 44
    }
}

sStartDate = wasdi.getParameter("START_DATE", "2018-01-10")
sEndDate = wasdi.getParameter("END_DATE", "2018-12-19")

oParameters = {'dataset': "L1B_RAD",
               'relativeorbitnumber': "523",
               'parameterName': "L1B_RAD",
               'dayNightFlag': 'Day'}

images = wasdi.searchEOImages(
    sPlatform='ECOSTRESS',
    oBoundingBox=oBBox,
    sDateFrom=sStartDate,
    sDateTo=sEndDate,
    sProvider='AUTO',
    aoParams=oParameters)

wasdi.wasdiLog(f"get_ecostress_products: Found {len(images)} images")

# let's print name of the first image found
if len(images) > 0:
    print(images[0]['id'])

```

What it does:

- Initializes the input variables
- Searches for the corresponding results
- checks that at least one file is returned
- accesses a field in the results

Note: The developer can decide what is needed in the params.json file. If you decide to use the [WASDI User Interface](#) your parameters will be generated automatically by WASDI.

Note: With the [WASDI User Interface](#) you can use the `renderAsStrings` flag to ask WASDI to get all your parameters in String Format. In this case you will be responsible to convert your data in your code.

Note: The Bounding Box Format used here is the one used by the User Interface when `renderAsStrings` is missing or

false. The Bounding Box format when renderAsStrings: true is “NORTH,WEST,SOUTH,EAST”.

Note: The Date is formatted by the User Interface as “YYYY-MM-DD”.

5.16 Search ERA5 products

The following code shows how to search for ERA5 products in WASDI

5.16.1 Prerequisites

To run this code you need:

- A running Python 3.x Environment
- A valid WASDI Account
- A valid Config file

If this is not clear, you probably need to take a look to the [Python Tutorial](#) before.

5.16.2 Recipe

Note: Assume you have at least one workspace and you have configured it in the config.json file.

To search for ERA5 products, the following fields are mandatory:

- Collection: this is always going to be ‘ERA5’
- Start Date
- End Date
- Bounding Box

Two additional search parameters, specific for ERA5 products, are also mandatory:

- dataset
- aggregation: this is used to specify how you want your data to be stored into the downloaded products. “daily” will produce one file per day, while “monthly” will produce one file per each calendar month

The other search parameters that you need to specify will depend from the specific dataset you are interested in. For ERA5 pressure levels, ERA5 single leels and ERA5 Land, you will need to speficy:

- product type
- variables
- format

Additionally, for ERA5 pressure levels, you need to specify:

- pressure levels.

Note: The aforementioned search filters: dataset, aggregation, variables, format and pressure levels, should be added to a dictionary, that you can then pass to the method `wasdi.searchEOImages` through the optional parameter `aoParams`

Let's see an example of code to search for ERA5 pressure levels products, which is the more comprehensive with respect to search parameters involved:

```
def get_pressure_levels_products():
    wasdi.wasdiLog("*** get_pressure_levels_products ***")

    # Create the Bounding Box Object: usually you will read it from the parameters
    oBBox = wasdi.getParameter("BBOX", None)

    # If it is null we show here how to initialize manually
    if oBBox is None:
        oBBox = {
            "northEast": {
                "lat": 44.2879447888337,
                "lng": 9.5
            },
            "southWest": {
                "lat": 43.5,
                "lng": 8.4
            }
        }

    # Set Start Date
    sStartDate = wasdi.getParameter("START_DATE", "2023-07-01")

    # Set End Date
    sEndDate = wasdi.getParameter("END_DATE", "2023-07-31")

    aoParams = {'dataset': "reanalysis-era5-pressure-levels",
                "pressureLevels": "1000",
                'variables': "RH",
                'format': "netcdf",
                'aggregation': "daily"}

    aoProductsFoundArray = wasdi.searchEOImages(
        sPlatform='ERA5',
        oBoundingBox=oBBox,
        sDateFrom=sStartDate,
        sDateTo=sEndDate,
        sProvider='AUTO',
        sProductType='reanalysis',
        aoParams=aoParams)

    if len(aoProductsFoundArray) > 0:
        # let's see the file name corresponding to the product we found:
        wasdi.wasdiLog(aoProductsFoundArray[0]["title"])
```

What it does:

- Initializes the input variables
- Searches for the corresponding results
- checks that at least one file is returned
- accesses a field in the results

Note: The developer can decide what is needed in the params.json file. If you decide to use the [WASDI User Interface](#) your parameters will be generated automatically by WASDI.

Note: With the [WASDI User Interface](#) you can use the [renderAsStrings](#) flag to ask WASDI to get all your parameters in String Format. In this case you will be responsible to convert your data in your code.

Note: The Bounding Box Format used here is the one used by the User Interface when renderAsStrings is missing or false. The Bounding Box format when renderAsStrings: true is **“NORTH,WEST,SOUTH,EAST”**.

Note: The Date is formatted by the User Interface as “YYYY-MM-DD”.

5.17 Import Images after a Search

The following code shows how to import the results of a search to a workspace.

5.17.1 Prerequisites

To run this code you need:

- A running Python 3.x Environment
- A valid WASDI Account
- A valid Config file
- A valid params.json file

If this is not clear, you probably need to take a look to the [Python Tutorial](#) before.

5.17.2 Recipe

Note: We will use Sentinel-1 for this sample but the same code can be used for all the missions and image types.

Note: In the code, you will see different options. Most likely, you will want to choose the one that best fits your needs.

This is our sample params.json file.

```
{
  "START_DATE": "2023-01-01",
  "END_DATE": "2023-01-01",
  "BBOX": {
    "northEast": {
      "lat": 20.1,
      "lng": 44.4
    },
    "southWest": {
      "lat": 19.3,
      "lng": 43.2
    }
  },
  "MISSION": "S1",
  "PRODUCT_TYPE": "GRD"
}
```

```
# Read Bounding Box, Start and End Date
oBBox = wasdi.getParameter("BBOX", None)
sStartDate = wasdi.getParameter("START_DATE", "2023-01-01")
sEndDate = wasdi.getParameter("END_DATE", "2023-01-31")

# Read Mission and Product Type
sMission = wasdi.getParameter("MISSION", "S1")
sProductType = wasdi.getParameter("PRODUCT_TYPE", "GRD")

# Search Images
aoProductsFoundArray = wasdi.searchEOImages(sMission, sStartDate, sEndDate,
↳sProductType=sProductType, oBoundingBox=oBBox)

# OPTION 1: Import a single image and wait for the image to be available
if len(aoProductsFoundArray) > 0:
    wasdi.importProduct(aoProductsFoundArray[0])

# OPTION 2: Import a single image WITHOUT waiting for it:
if len(aoProductsFoundArray) > 0:
    sProcessId = wasdi.asyncImportProduct(aoProductsFoundArray[0])
    # Here you can do what you want
    wasdi.wasdiLog("Started Import of one image, the associated process id is " +
↳sProcessId)
    # Call this if you need to wait
    wasdi.waitProcess(sProcessId)

# OPTION 3: Import All Products and wait for the images to be available
wasdi.importProductList(aoProductsFoundArray)

# OPTION 4: Import All Products without waiting
sProcessId = wasdi.asyncImportProductList(aoProductsFoundArray)
# Here you can do what you want
wasdi.wasdiLog("Started Import of all images, the associated process id is " +
↳sProcessId)
# Call this if you need to wait
wasdi.waitProcess(sProcessId)
```

What it does:

- Reads Input Parameters
- Starts searching for S1 GRD Images
- Imports 1 Product
- Asynchronously imports 1 Product
- Imports All Products
- Asynchronously imports all Products

5.18 Import And Pre-Process

You may need to pre-process your images before being able to work with them. This snippets shows a convenient method to automatically pre-process all imported images.

5.18.1 Prerequisites

To run this code you need:

- A running Python 3.x Environment
- A valid WASDI Account
- A valid Config file
- A valid params.json file

If this is not clear, you probably need to take a look to the [Python Tutorial](#) before.

5.18.2 Recipe

Note: We will use Sentinel-1 GRD and the public LISTSinglePreproc2 workflow for this snippet, but this method can be used with any mission and any compatible workflow.

Note: The LISTSinglePreproc2 is designed to georeference a Sentinel-1 GRD Image (apply orbit, radiometric calibration, terrain correction...)

This is our sample params.json file:

```
{
  "START_DATE": "2023-01-01",
  "END_DATE": "2023-01-01",
  "BBOX": {
    "northEast": {
      "lat": 20.1,
      "lng": 44.4
    },
    "southWest": {
```

(continues on next page)

(continued from previous page)

```

        "lat": 19.3,
        "lng": 43.2
    }
},
"MISSION": "S1",
"PRODUCT_TYPE": "GRD",
"WORKFLOW": "LISTSinglePreproc2"
}

```

```

# Read Bounding Box, Start and End Date
oBBBox = wasdi.getParameter("BBOX", None)
sStartDate = wasdi.getParameter("START_DATE", "2023-01-01")
sEndDate = wasdi.getParameter("END_DATE", "2023-01-31")

# Read Mission and Product Type
sMission = wasdi.getParameter("MISSION", "S1")
sProductType = wasdi.getParameter("PRODUCT_TYPE", "GRD")
sWorkflow = wasdi.getParameter("WORKFLOW", "LISTSinglePreproc2")

# Search Images
aoProductsFound = wasdi.searchEOImages(sMission, sStartDate, sEndDate,
↳sProductType=sProductType, oBoundingBox=oBBBox)

if len(aoImagesToProcess)>0:
    # Import and pre-process all the images: '_preproc.tif' is the suffix added to the
↳original file name that will be used as output name of the workflow
    wasdi.importAndPreprocess(aoProductsFound, sWorkflow, '_preproc.tif')

```

What it does:

- Reads Input Parameters
- Starts searching for S1 GRD Images
- Imports and runs the workflow on all the images

5.19 Run Snap Workflow

This snippet show how to run SNAP worklows by code

5.19.1 Prerequisites

To run this code you need:

- A running Python 3.x Environment
- A valid WASDI Account
- A valid Config file
- A valid params.json file

If this is not clear, you probably need to take a look to the [Python Tutorial](#) before.

5.19.2 Recipe

Note: We will use Sentinel-1 GRD and the public LISTSinglePreproc2 workflow for this snippet, but this method can be used with any mission and any compatible workflow.

Note: Remember that you can upload in WASDI your own SNAP Workflows. For further explanation take a look [here](#).

This is our sample params.json file.

```
{
  "START_DATE": "2023-01-01",
  "END_DATE": "2023-01-01",
  "BBOX": {
    "northEast": {
      "lat": 20.1,
      "lng": 44.4
    },
    "southWest": {
      "lat": 19.3,
      "lng": 43.2
    }
  },
  "MISSION": "S1",
  "PRODUCT_TYPE": "GRD",
  "WORKFLOW": "LISTSinglePreproc2"
}
```

```
# Read Bounding Box, Start and End Date
oBBox = wasdi.getParameter("BBOX", None)
sStartDate = wasdi.getParameter("START_DATE", "2023-01-01")
sEndDate = wasdi.getParameter("END_DATE", "2023-01-31")

# Read Mission and Product Type
sMission = wasdi.getParameter("MISSION", "S1")
sProductType = wasdi.getParameter("PRODUCT_TYPE", "GRD")

# Read the name of the workflow to start
sWorkflow = wasdi.getParameter("WORKFLOW", "LISTSinglePreproc2")

# Search Images
aoProductsFound = wasdi.searchEOImages(sMission, sStartDate, sEndDate,
↪sProductType=sProductType, oBoundingBox=oBBox)

if len(aoImagesToProcess)>0:
  # Import the first image
  wasdi.importProduct(aoProductsFound[0])

  # Get the name of the image
  sImageName = aoProductsFound[0]["fileName"]
```

(continues on next page)

(continued from previous page)

```

# In general, workflows can have multiple inputs
asInputImages = [sImageName]

# We need to decide the name of the output file: here you may add a more smart code_
↪ (add a suffix to the original name for example)
sOutputImage = "preprocessed.tif"

# In general, workflows can have also multiple outputs
asOutputImages = [sOutputImage]

#OPTION 1: run the workflow and wait for the result
wasdi.executeWorkflow(asInputImages, asOutputImages, sWorkflow)

#ALTERNATIVE OPTION 2: run asynch
sProcessId = wasdi.asynchExecuteWorkflow(asInputImages, asOutputImages, sWorkflow)
# Here you are free to do what you want
wasdi.wasdiLog("I started a workflow")
# Call this if you need to wait for it to finish
wasdi.waitProcess(sProcessId)
else:
    wasdi.wasdiLog("No file to pre-process found")

```

What it does:

- Reads Input Parameters
- Starts searching for S1 GRD Images
- Runs the workflow waiting for it
- Runs the workflow without waiting for it

5.20 Run Another WASDI Application

This snippet shows how to run another WASDI Application from your code.

5.20.1 Prerequisites

To run this code you need:

- A running Python 3.x Environment
- A valid WASDI Account
- A valid Config file

If this is not clear, you probably need to take a look to the [Python Tutorial](#) before.

5.20.2 Recipe

Note: We will use the hellowasdi app, but the same code can be used to call any WASDI Application that you have access to.

Note: To call an App, you need to understand it. Look to the help section of the app or the Json Parameter sample in the store. Additionally, you can contact the developer (if they have provided a contact email address) to understand the inputs (and outputs) required.

```
# Read the name we want to pass to the hellowasdi
sName = wasdi.getParameter("NAME", "my Test")

# Create the dictionary with the parameters to pass to the application
aoApplicationParameters = { "name": sName }

# Run the application: Applications are ALWAYS executed in asynchronous way
sProcessId = wasdi.executeProcessor("hellowasdiworld", aoApplicationParameters)

# Here you are free to do what you want
wasdi.wasdiLog("I started an app and I can do what I want")

# Call this if you need to wait for it to finish
wasdi.waitProcess(sProcessId)

wasdi.wasdiLog("Here I know the application is finished")
```

What it does:

- Reads Input Parameters
- Creates the dictionary with the params to pass in input to our application
- Runs the application
- Waits for the application to finish

5.21 Save Payload

This snippet demonstrates how to save a payload as additional output of your application. WASDI apps are mainly meant to create new products that will be added to the workspace. But often you may want to save also other information: the payload is the solution. Payloads are just a string you can save. Usually, the payload is in JSON format. The user can view the payload in the WASDI Editor.

5.21.1 Prerequisites

To run this code you need:

- A running Python 3.x Environment
- A valid WASDI Account
- A valid Config file

If this is not clear, you probably need to take a look to the [Python Tutorial](#) before.

5.21.2 Recipe

Note: We will save a JSON Payload. The payload is really saved only when the app is running in WASDI. When running locally the payload is not really saved.

Note: It is not mandatory, but is good practice in the payload to also save the inputs received.

```
# Read the input parameters
aoInputParameters = wasdi.getParametersDict()

# Declare the payload
aoPayload = {}

# Add the inputs as a member of the payload
aoPayload["inputs"] = aoInputParameters

# Do your own code here...

# Here we add some sample values
aoPayload["item_found"] = 3
aoPayload["max_value"] = 1893
aoPayload["selected_color"] = "red"

# Save the payload
wasdi.setPayload(aoPayload)
```

What it does:

- Reads Input Parameters
- Adds some elements to the payload
- Saves the payload

5.22 Get list of S2 tiles in an area of interest

The following code shows how to list the different S2 tiles that intersects the area of interest

5.22.1 Prerequisites

To run this code you need:

- A running Python 3.x Environment
- A valid WASDI Account
- A valid Config file

If this is not clear, you probably need to take a look to the [Python Tutorial](#) before.

5.22.2 Recipe

Note: Assume you have at least one workspace and you configured it in the config.json file

This snippet is meant to get a list of unique S2 Tiles as a result of a search. For more info about the search of S2 please see [Search Sentinel-2 Images](#) .

```
# Read the Bounding Box Object: usually you will take it from the parameters
oBBox = wasdi.getParameter("BBOX", None)
# Set Start Date
sStartDate = wasdi.getParameter("START_DATE", "2023-01-01")
# Set End Date
sEndDate = wasdi.getParameter("END_DATE", "2023-01-10")

# Start Search S2 MSI1C Images (Level 1)
aoProductsFoundArray = wasdi.searchEOImages("S2", sStartDate, sEndDate, sProductType=
↳ "S2MSI1C", oBoundingBox=oBBox)

# The result is an array of Objects. Each Object is a Dictionary.

# Here we will put our list of tiles
asTiles = []
try:
    for oS2L2Image in aoProductsFoundArray:
        sImage = oS2L2Image["title"]
        sTile = sImage.split('_')[5]
        if sTile not in asTiles:
            asTiles.append(sTile)
except Exception as oE:
    wasdi.wasdiLog(f'Error listing the tiles: {type(oE)}: {oE}')
    wasdi.updateStatus('ERROR', 0)
    return

wasdi.wasdiLog("Involved Tiles " + str(asTiles))
```

What it does:

- Read Bounding Box and Dates from parameters
- Starts searching S2 L1 Images
- Loop the results and extract unique Tiles

Note: The same snippet can be used also for Level 2 Data.

5.23 Use Library as client

This snippet shows how to use the WASDI Lib as client, to run applications and get results.

5.23.1 Prerequisites

To run this code you need:

- A running Python 3.x Environment
- A valid WASDI Account
- A valid Config file

If this is not clear, you probably need to take a look to the [Python Tutorial](#) before.

5.23.2 Recipe

We are going to init the WASDI Lib, call an application and get back the result

Note: Often WASDI is used to deploy application, but in the same lib can be used also as a Client of WASDI.

Note: In this sample we use hellowasdiworld application that DOES NOT produce any file, so this snippet as it is will not download locally any file. In the real life you will probably use an app that will add some file in the workspace.

```
# Initialize the lib: you must set the right path to your config file
wasdi.init('myconfig.json')

# Create a workspace where we will run our application
wasdi.createWorkspace('NAME')

# Create the dictionary with the params to pass to the application
aoParams = {}
aoParams['NAME'] = "Test"

# Run the application: Applications are ALWAYS executed in asynchronous way
sProcessId = wasdi.executeProcessor("hellowasdiworld", aoParams)

# Here you are free to do what you want
```

(continues on next page)

(continued from previous page)

```
wasdi.wasdiLog("I started an app and I can do what I want")

# Call this when you need to wait for it to finish
wasdi.waitProcess(sProcessId)

# Get the List of Files in Workspace
asFilesProduced = wasdi.getFileByActiveWorkspace()

# For all the files produced
for sFile in asFilesProduced
    # Since you are running out of wasdi, this will take the produced files locally for
    ↪ you
    wasdi.getPath(file)
```

What it does:

- Initializes the lib
- Creates a workspace
- Creates the parameters for the app
- Starts the app
- Waits for it
- Get the files in the workspace
- Get a local copy of all the produced files

5.24 Change HTTP request timeouts

The following code shows how to set a custom HTTP request timeout to the wasdi lib. Specifically, the new value will affect the HTTP connection timeout and the response timeout. The default value is set to 120 seconds.

5.24.1 Prerequisites

To run this code you need:

- A running Python 3.x Environment
- A valid WASDI Account
- A [valid Config file](#)

If this is not clear, you probably need to take a look to the [Python Tutorial](#) before.

5.24.2 Recipe

You can change the default value of the HTTP request timeout by calling the method *setRequestsTimeout*. That method takes an integer as parameter to represent the timeout value, expressed in seconds.

The following code snippet provides an example of how to set the HTTP request timeout.

```
wasdi.wasdiLog(f"Default timeout value: {wasdi.geRequestsTimeout()}")

iNewTimeOut = wasdi.getParameter("REQUEST_TIMEOUT", 240)

if iNewTimeOut is not None:
    wasdi.setRequestsTimeout(iNewTimeOut)

wasdi.wasdiLog(f"New timeout value: {wasdi.geRequestsTimeout()}")
```

What it does:

The code first prints the current HTTP request timeout value, got using the method *getRequestsTimeout*.

Then, it tries to read the new timeout value from the parameters file, looking for a field “REQUEST_TIMEOUT”. If such a parameter is not present, then it takes 240 as fallback value (for an overview on how to use the parameters file, have a look at [this recipe](#)).

The new value is passed as a parameter to the *setRequestsTimeoutValue* and the value of the request timeout is printed again, to verify that it changed accordingly.

5.25 Add a Data Provider to WASDI

5.25.1 Introduction

A Data Provider is an external service that can be used to query and import Data in WASDI. The main business entities involved in this operation are:

- Platform: this is the type of data. Usually identified as a Satellite Mission. Platforms are for example Sentinel1, Sentinel2, ENVISat etc. Each Platform, in general, can be found in more data providers.
- Query Executors / catalogue: the Query executor is the WASDI hierarchy used to query the Data Provider Catalogue
- Provider Adapters: Objects used by the launcher to download/import files from an external service.

Note: This tutorial assumes that you already have the WASDI Project up and running in your environment

5.25.2 Getting Started

To start adding a Data Provider the first thing to check is the Platform's support. Supported platforms are listed in the class:

```
wasdi.shared.queryexecutors.Platforms
```

Each platform is represented by a static String that declares the Platform Code.

In case of a new platform, add also the support to:

```
wasdi.shared.utils.MissionUtils.getPlatformFromSatelliteImageFileName
```

This method must be able to infer to platform type from the file name.

If the Data Provider support a new Platform, the code must be added in the Platforms class.

5.25.3 Client Filter

The client search page is configured using the JSON config file `config/appconfig.json`, which contains an array of "missions" objects. Each mission is a JSON similar to this one:

```
{
  "name": "S1",
  "indexname": "platformname",
  "indexvalue": "Sentinel-1",
  "selected": true,
  "filters": [
    {
      "indexname": "filename",
      "indexlabel": "Satellite Platform",
      "indexvalues": "S1A_*|S1B_*",
      "indexvalue": "S1A_*"
      "regex": ".*"
    },
    {
      "indexname": "producttype",
      "indexlabel": "Product Type",
      "indexvalues": "SLC|GRD|OCN",
      "indexvalue": "GRD"
      "regex": ".*"
    }
  ]
}
```

name will be used to create the corresponding tab in the WASDI search section.

Filters can be added to the search form of the data provider. Each filter has an **indexname** that represents the name of the filter and a **indexvalue** will contain the value of filter selected by the user.

indexname-indexvalues is an array used to create a new variable from the client to the server.

indexname:"platformname" is a filter that *must* be used to set the Platform Code as defined in the Java Platforms object. Accordingly, the **indexvalue** for a ceratin *platformname* will correspond to the code of the Platform in WASDI.

You can add as many filters as required/supported by the Data Provider.

WASDI automatically handles the date interval, the bounding box, platformname and, if supplied, the producttype.

Other filters can be added and will have to be supported server side by your own QueryTranslator.

5.25.4 Query Executor, Query Translator and Response Translator

This section is needed to make WASDI search the new Data Provider. WASDI receives always the query as string that must be translated in for the provider. Results must then be converted to the WASDI format.

When the user wants to download a file, QueryExecutor will pass to the ProviderAdapter the link and the file name that must be imported.

In general, the name is the key element: since WASDI supports automatic data provider selection, the system will search the highest priority provider adapter that supports that platform. The Download Operation will use the QueryExecutor to obtain the url to use for the download from the filename. Since a platform can be supported by many Data Providers, this method assures to get always the right file, even from different sources.

In the particular situation where a single platform is supported only by One Data Provider, in the name and in the link, the developer can decide to store more complete informations that may be needed to interoperate with the external API.

To create a new QueryExecutor, add a new package in

```
wasdi.shared.queryexecutors
```

Create 3 objects:

- The new QueryExecutor deriving from QueryExecutor
- The new QueryTranslator deriving from QueryTranslator
- The new ResponseTranslator deriving from ResponseTranslator

Query Executor

The QueryExecutor MUST define its own unique code in the variable m_sProvider, inside the constructor. Usually, it must also instantiate its own QueryTranslator and ResponseTranslator in the constructor.

```
public QueryExecutorPLANET() {
    m_sProvider="PLANET";
    this.m_oQueryTranslator = new QueryTranslatorPLANET();
    this.m_oResponseTranslator = new ResponseTranslatorPLANET();
}
```

QueryExecutor must implement:

```
public int executeCount(String sQuery): receive in input the WASDI query, must return
↳ the number of results for the provider
public List<QueryResultViewModel> executeAndRetrieve(PaginatedQuery oQuery, boolean
↳ bFullViewModel): receive in input the WASDI query, must return the list of provider's
↳ results as a list of QueryResultViewModel.
```

The boolean parameter bFullViewModel taken as an input by the method executeAndRetrieve depends on the type of search being executed. Indeed, WASDI supports two types of search:

- the paginated search, used by the WASDI Web client and implemented by the REST endpoint /search/query

- the not paginated search, returning the whole list of results at once, which is used by the WASDI libraries and implemented by the REST endpoint `/search/querylist`.

When the type of search being executed is a not-paginated one, the value of the boolean is set to *false*, in order to add only the essential information to the list of `QueryResultViewModel` objects returned by the method. By contrast, the boolean is set to *true* when the search is paginated. In that case, the `QueryResultViewModel` objects will contain exhaustive information about each product returned by the search.

In the `QueryResultViewModel` the most important fields are:

- *title* : name of the file
- *link* : url for the direct download of the file

`QueryExecutor` base class implements:

```
public String getUriFromProductName(String sProduct, String sProtocol, String
↪ sOriginalUrl)
```

This method is very important for the auto data provider selection: it takes the name of the product returned by any catalogue that supports that platform, the original url returned by the same catalogue and must return the URI to access the file for the Provider Adapter. URI is usually an http link but can be a file path or a ftp link or other, depending on the linked `DataProvider` that takes the file with that URI in the `executeDownloadFile` method.

The basic implementation just performs a query filtering by the exact product name and uses the result to get the relative URI: it MUST be overridden if this does not work.

There are at least 2 `QueryExecutors` base classes that can be used other than the abstract one:

QueryExecutorHttpGet

Each `Query Executor` that uses standard get http calls, should derive from this class and implement the abstracts methods of `QueryTranslator` to get Search and Count URL and of `Response Translator` to convert the return of the search query in WASDI View Models `executeCount` steps are:

- Check if the platform is supported
- call `QueryTranslator.getCountUrl`
- execute std http get call with that url
- call `m_oResponseTranslator.getCountResult` to get the number of results.

`executeAndRetrieve` steps are:

- Check if the platform is supported
- call `QueryTranslator.getSearchUrl`
- execute std http get call with that url
- call `m_oResponseTranslator.translateBatch` to get the number of results.

QueryExecutorOpenSearch

Base class for Providers supporting Open Search.

QueryTranslator

QueryTranslator has the goal to convert the WASDI query in a valid provider query. The user must implement 2 methods:

```
String getCountUrl(String sQuery)
String getSearchUrl(PaginatedQuery oQuery)
```

The base class contains the `parseWasdiClientQuery` method.

```
QueryViewModel oQuery = parseWasdiClientQuery(sQuery);
```

In its implementation, the WASDI query is parsed and transformed in the corresponding view model. If the Platform or the Data Provider have special filters, these must be supported (parsed) there.

It is important to CHECK that `parseWasdiClientQuery` is able to detect the `platformName` attribute of `QueryViewModel`, since it is mandatory.

ResponseTranslator

The ResponseTranslator must translate the results of the API call made to the data provider into the WASDI format.

```
public class ResponseTranslatorPLANET extends ResponseTranslator {

    @Override
    public List<QueryResultViewModel> translateBatch(String sResponse, boolean_
↪bFullViewModel) {
        return null;
    }

    @Override
    public int getCountResult(String sQueryResult) {
        return 0;
    }
}
```

The WASDI format is a list of `QueryResultViewModel` objects. The basic information contained in those objects is:

- *title* : name of the file,
- *summary* : description. Supports a sort of std like: “Date: 2021-12-25T18:25:03.242Z, Instrument: SAR, Mode: IW, Satellite: S1A, Size: 0.95 GB” but is not mandatory,
- *id* : provider unique id,
- *link* : link to download the file,
- *footprint* : bounding box in WKT,
- *provider* : provider used to get this information.

The `QueryResultViewModel` object also contains a field `properties`, which is a dictionary filled with all the properties supported by the data provider. It can be seen with the “info” button in the client.

Some commonly used properties, shown in the client, are:

- *date* : reference date,
- *Satellite* : platform,

- *instrument* : used instrument,
- *sensorMode* : sensing mode,
- *size* : image size as string,
- *relativeOrbit*: relative orbit of the acquisition.

To add the query executor to WASDI, remember to add it to the factory

```
wasdi.shared.queryexecutors.QueryExecutorFactory
```

```
static {
    Utils.debugLog("QueryExecutorFactory");
    final Map<String, Supplier<QueryExecutor>> aoMap = new HashMap<>();

    aoMap.put("ONDA", QueryExecutorONDA::new);
    aoMap.put("SENTINEL", QueryExecutorSENTINEL::new);
    aoMap.put("SOBLOO", QueryExecutorSOBLOO::new);
    aoMap.put("EODC", QueryExecutorEODC::new);
    aoMap.put("CREODIAS", QueryExecutorCREODIAS::new);
    aoMap.put("LSA", QueryExecutorLSA::new);
    aoMap.put("VIIRS", QueryExecutorVIIRS::new);
    aoMap.put("CDS", QueryExecutorCDS::new);
    aoMap.put("PROBAV", QueryExecutorPROBAV::new);
    aoMap.put("PLANET", QueryExecutorPLANET::new);

    s_aoExecutors = Collections.unmodifiableMap(aoMap);

    Utils.debugLog("QueryExecutorFactory.static constructor, s_aoExecutors content:
↪");
    for (String sKey : s_aoExecutors.keySet()) {
        Utils.debugLog("QueryExecutorFactory.s_aoExecutors key: " + sKey);
    }
}
```

5.25.5 Provider Adapter

The ProviderAdapter has the goal to ingest the file, either being it a downloaded file or a file copy. Each ProviderAdapter is linked to the relative QueryExecutor using the same DataProviderCode.

WASDI supports automatic DataProvider selection, consequently each ProviderAdapter must be able to get the URI link of a file from the file name. The ProviderAdapter must also be able to declare its “score” in the ability to fetch a file: this score will be used by WASDI to select the best DataProvider for the file to be downloaded.

Scores are defined as integers by the enum in

```
wasdi.dataproviders.DataProviderScores
```

A higher number means the best possibility to get the file. At the moment values are: FILE_ACCESS(100), SAME_CLOUD_DOWNLOAD(90), DOWNLOAD(80), SLOW_DOWNLOAD(50), LTA(10);

The typical empty implementation of a ProviderAdapter is:

```
public class PLANETProviderAdapter extends ProviderAdapter {
```

(continues on next page)

(continued from previous page)

```

public PLANETProviderAdapter() {
    super();
    m_sDataProviderCode = "PLANET";
}

public PLANETProviderAdapter(LoggerWrapper logger) {
    super(logger);
    m_sDataProviderCode = "PLANET";
}

@Override
protected void internalReadConfig() {
}

@Override
public long getDownloadFileSize(String sFileURL) throws Exception {
    return 0;
}

@Override
public String executeDownloadFile(String sFileURL, String sDownloadUser, String
↪sDownloadPassword,
                                String sSaveDirOnServer, ProcessWorkspace oProcessWorkspace, int
↪iMaxRetry) throws Exception {
    return null;
}

@Override
public String getFileName(String sFileURL) throws Exception {
    return null;
}

@Override
protected int internalGetScoreForFile(String sFileName, String sPlatformType) {
    return 0;
}
}

```

In the constructor, the provider MUST set its own code in `m_sDataProviderCode`, that must correspond to the code used by the linked `QueryExecutor`.

The methods of the class are:

- `internalReadConfig` can be used to read from `WasdiConfig` specific configurations.
- `getDownloadFileSize` receives the file URI and must return the size of the file. It is useful to give progress to the user.
- `executeDownloadFile`. It is the main method: it receives the `sFileURL` OBTAINED BY THE LINKED DATA PROVIDER, the credentials, the local folder, the process workspace and the max number of retry allowed. Must return the valid file full path or "" if the download was not possible.
- `getFileName` extracts the file name from the URL
- `internalGetScoreForFile` returns the score auto-evaluated by the Provider Adapter to download `sFileName`

of `sPlatformType`.

The base class has many utility functions ready for many common cases:

- `downloadViaHttp`: std http download
- `getFileSizeViaHttp`: request file size to http
- `copyStream`: copy a stream to another
- `localFileCopy`: makes a local file copy
- `getFileNameViaHttp`: extracts name from http call
- `isWorkspaceOnSameCloud`: state if the workspace is on the same cloud of the `DataProvider` (useful for score)

The provider adapter MUST be added to:

```
wasdi.dataproviders.ProviderAdapterFactory
```

When executing the download of a file, many methods implemented by the provider adapter will be used by the class

```
wasdi.operations.Download
```

Specifically:

- `executeOperation` relies on the data provider implementation to retrieve the name of the file to download as well as its size and executing the actual download of the file;
- `getBestProviderAdapter` implements the scoring mechanism to select the best data provider to download a file,
- `doesProviderAdapterFindFile` is used to double-check the availability of the file in the data provider, before executing the actual download.

Configuration

Each Data provider is listed in the `dataProviders` section of `wasdiConfig.json`. An example is:

```
{
  "name": "LSA",
  "description": "LSA DATA CENTER",
  "link": "https://www.collgs.lu/",
  "searchListPageSize": "50",
  "defaultProtocol": "https://",
  "parserConfig": "/tmp/lisaParserConfig.json",
  "user": "USER",
  "password": "PASSWORD",
  "localFilesBasePath": "/mount/data/",
  "urlDomain": "https://collgs.lu/repository/",
  "connectionTimeout": "",
  "readTimeout": "",
  "adapterConfig": "",
  "cloudProvider": "AdwaisEO",
  "supportedPlatforms": ["Sentinel-1", "Sentinel-2"]
}
```

- **name** is the unique code of the data provider

- **parserConfig** and **adapterConfig** are 2 possible specific config file that can be used by the Data Provider, one for the QueryExecutor and the other for the Provider Adapter.
- **user** and **password**, if present, are the credentials of the Data Provider.
- **cloudProvider** is the unique code of the cloud where the DataProvider is hosted. Can be used to set the score of the performance for a specific file download.
- **supportedPlatforms** is an array if strings. Each String is a valid entry of the Plaforms supported by WASDI: here is written the list of plaforms that this DataProvider supports.

Since each Platform can be supported by many data providers, among which we can select the best data provider, WASDI also defines the best catalogue to use to query that specific Platform. This is done in the *catalogues* section of *wasdiConfig*.

```
"catalogues": [
  {
    "platform": "Sentinel-1",
    "catalogues": ["LSA", "CREODIAS", "SENTINEL", "ONDA", "EODC"]
  }
]
```

In the example, we see that the Platform Sentinel-1 is supported by 6 catalogues (DataProviders) and the priority one is LSA Data Center.

To enable the new data provider to download products, we also need to add and configure a new queue to the arrays *schedulers*, under the JSON field *scheduler*.

```
"schedulers": [
  {
    "name": "DOWNLOAD.PROVIDERNAME",
    "maxQueue": "10",
    "timeoutMs": "1111111",
    "opTypes": "DOWNLOAD",
    "opSubType": "PROVIDERNAME",
    "enabled": "1"
  }
]
```

- **name** is the unique code of the queue, following the pattern DOWNLOAD.NAME_OF_THE_DATAPROVIDER.
- **maxQueue** is the number of elements that can be put in the queue.
- **timeOutMS** is the default queue timeout, in milliseconds.
- **opTypes** it is a comma separated list of OperationTypes supported by the queue. In this specific case of adding a new data provider, the operation supported by the queue should be “DOWNLOAD”.
- **opSubType** must be a valid subtype of opType. In this case, the field is used to store the name of the data provider the queue refers to.
- **enabled** is a flag to enable (“1”) or disable (“0”) the queue.

Welcome to Space, Have fun!

5.26 Add Application User Interface controls

5.26.1 Introduction

This tutorial is done to show how to add a new user control to the WASDI Application interface.

All the WASDI Applications takes in input a generic dictionary of parameters. The developer knows the meaning of the parameter of his own application and can document it with a readme.md file. In the backend of the wasdi app, it is also possible to create an automatic user interface to show the application in the marketplace. Look the [How to create a User Interface \(UI\)](#) for more general info about this.

Here we see how to create and add a new control that WASDI App developers will be able to add their own user interface.

5.26.2 Existing Controls

The concept of User Interface control is very old in the IT history; it is present for sure from the first versions of Windows and Apple operating systems. The user controls are a component of the user interface where the user can interact to insert, read or update a specific kind of data.

The most easy and common control is probably a TextBox: a box where the user can insert a text.

Common controls, already supported in WASDI, are for example:

- TextBox: a box to insert strings
- ComboBox: a list of elements that can be hidden. The user can choose one element of the list
- NumericSlider: a slider that let the user insert an integer number
- NumericField: similar to a textbox, but accepts only floating numbers and not generic strings
- DateTimePicker: a control desinged to insert a date
- Checkbox: usually a sort of switch to input a boolean value

Controls more specific for WASDI are:

- SelectArea: a map that let the user insert a bounding box
- SearchEOImage: a mini-search engine for EO Images
- ProductComboBox: a combobox auto populated with the names of a product in a workspace

In this tuturiel we will add the ListBox control: a list of elements where the user can choose zero, one or more elements.

Note: This tutorial requires the WASDI Client project already configured in your environment

5.26.3 WASDI UI definition language

WASDI UI are described by Json Files. Each control has a minimum structure:

```
{
  "param": "PARAM_NAME",
  "type": "textbox",
  "label": "Description",
  "tooltip": "Quick help",
  "required": false
}
```

- param: name of the param that will be given to the app
- type: type of the control. When we add a new control, we add a new type.
- label: what we show as description of the parameter to the user
- tooltip: a quick help that will be shown when the user will hover the mouse over the control
- required: true if the input of this param is mandatory otherwise false

Every control MUST have these data as minimum. Then, the designer, can decide to add more parameters specific of his own control.

For example for our ListBox we define:

```
{
  "param": "PARAM_NAME",
  "type": "listbox",
  "label": "description",
  "values": [],
  "required": false,
  "tooltip": ""
}
```

In respect to the general control, we added our **values** parameter:

- values: array of strings. Each string will be an element of the list. The user will be able to choose one or more of these values.

5.26.4 View Element Factory

In the **lib/factories** folder there is the **ViewElementFactory.js** file.

This file contains the definition of all the View Elements. Each control is a View Element. This file contains a class for each control supported. So the first step is to add our class to the ViewElementFactory:

```
/**
 * List Box Control Class
 * @constructor
 */
class ListBox extends UIComponent { constructor() {
super();

this.aoElements = [];
this.aoSelected = [];
```

(continues on next page)

(continued from previous page)

```

/**
 * Return the selected product
 * @returns {}
 */
this.getValue = function () {
    return this.aoSelected;
}

/**
 * Return the name of the selected product
 * @returns {}
 */
this.getStringValue = function () {

    let sReturn = "";
    let iSel = 0;

    if (this.aoSelected != undefined) {
        if (this.aoSelected != null) {
            for (iSel = 0; iSel<this.aoSelected.length; iSel++) {
                if (iSel>0) sReturn = sReturn + ",";
                sReturn = sReturn + this.aoSelected[iSel];
            }
        }
    }
    return sReturn;
}
};

```

Our class derive from the base `UIComponent` one.

The class must implement:

- `this.getValue = function () { ... }`: here the class must be able to take the input of the user and return it in the native format (date for dates, number for slider, string for text...)
- `this.getStringValue = function () { ... }`: here the class must be able to take the input of the user and return it in a string representation

The two methods are needed because not all the languages can support the input of native parameters so there is the option to translate all the input of the user in strings. The developer will later convert the strings in the code of the WASDI app.

The class CAN implement:

- `this.isValid = function (asMessage) { ... }`: must return true or false, making a validation of the user input. The `asMessages` parameter is an array of strings: if the validation is not ok the control can add here a message that will be shown to the user.

After the class has been created, we must move in the `createViewElement` method in the same file.

This method will be called by the Marketplace UI window to initialize the user interface. It receives in input the json part of the actual control that must be created.

Here there is a cascade of if-else to detect the type of control: we need to add our control:

```

else if (oControl.type === "listbox") {
  // List Box
  oViewElement = new ListBox();

  let iValues = 0;

  oViewElement.aoElements = [];
  oViewElement.aoSelected = [];

  for (; iValues < oControl.values.length; iValues++) {
    oViewElement.aoElements.push(oControl.values[iValues]);
  }
}

```

All the default properties (label, type, paramName, required) are set by the function. In our branch of the if we just need to:

- Create our class
- Read the “extended” properties we defined in our json definition and use it to initialize our class

This step, is obviously strongly dependant by the control we are implementing: here for example we read the values list of string and we save it in elements of our class. We also initialize another array, the one of selected elements, that will be filled by our directive...

5.26.5 Directive

In the **directives/wasdiApp** folder there are all the Angular Directive that are the physical implementation of the user control. Every time you create a new control you will create also a new directive.

Note: When you add your directive you will have to include the js file in the index.html and declare it in app.js module. If you have it, you will also have to include the .scss file in the style.scss. To build, remember also to add the require of the js file in the directive.js file.

Each control, or ViewElement, has a corresponding directive. The directive is left up to the developer, it has to represent the specific type of input you want to add to WASDI. It is supposed to interact with the ViewElement class, we will see how in short.

For the moment here an example of our ListBox Directive:

The controller:

```

angular.module('wasdi.wapListBox', [])
  .directive('waplistbox', function () {
    "use strict";
    return {
      restrict: "E",
      scope: {
        optionsDirective: '=options',
        //options: '=',
        selectedDirective: '=selected'
        // * Text binding ('@' or '@?') *
        // * One-way binding ('<' or '<?') *
      }
    };
  });

```

(continues on next page)

(continued from previous page)

```

        // * Two-way binding ('=' or '?') *
        // * Function binding ('&' or '&?') *
    },

    templateUrl:"directives/wasdiApps/wapListBox/wapListBox.html",
    link: function(scope, elem, attrs) {

        scope.pushOptionInSelectedList = function(sBandInput)
        {

            if(utilsIsStrNullOrEmpty(sBandInput) == true)↵

↵return false;

            var iNumberOfSelectedBand = scope.
↵selectedDirective.length;

            var bFinded = false;
            for(var iIndexBand = 0; iIndexBand <↵
↵iNumberOfSelectedBand; iIndexBand++)
            {

                if(scope.selectedDirective[iIndexBand]↵
↵== sBandInput)
                {

                    scope.selectedDirective.

                    bFinded=true;
                    break;

                }

            }

            if(bFinded == false)
            {
                scope.selectedDirective.push(sBandInput);
            }
            return true;
        };

        scope.isOptionSelected = function(sBandInput)
        {

            if(utilsIsStrNullOrEmpty(sBandInput) == true)↵

↵return false;

            var bResult=utilsFindObjectInArray(scope.
↵selectedDirective ,sBandInput);

            if(utilsIsObjectNullOrUndefined(bResult) ==↵
↵true) return false;

            if(bResult == -1)
            {

                return false;

            }

            return true;
        };
    };

```

(continues on next page)

(continued from previous page)

```

    }
  };
});

```

The view:

```

<div class="waplistbox-directive">
  <input type="text" class="form-control" placeholder="Search..." ng-model=
  ↪ "textFilter">

  <div class="list-group" >
    <a href="" class="list-group-item" ng-repeat="option in optionsDirective.
    ↪ | filter:textFilter track by $index " ng-class="{active: isOptionSelected(option)}" ng-
    ↪ click="pushOptionInSelectedList(option);">
      {{option}}
    </a>
  </div>
</div>

```

The style:

```

.waplistbox-directive
{

  .list-group
  {
    overflow-y: auto;
    max-height: 160px;
    min-height: 50px;

    border: 1px solid #43516A;
    border-top-left-radius: 4px;
    border-top-right-radius: 4px;
    a
    {
      //color: $wasdi-blue-logo;
      //border-color: $wasdi-blue-logo;
      border-color: transparent;
    }
    a:hover
    {
      background-color: darkgrey;
    }
  }

  .list-group-item.active, .list-group-item.active:focus, .list-group-item.active:hover {
    z-index: 2;
    color: #fff;
    background-color: #009036;
    border-color: white; // #337ab7;
  }
}

```

Is out of the scope of this tutorial to go in the details of the Angular code of this directive. Just recap that it suppose to receive an attribute called **options**, with the array of strings to display; also an attribute called **selected**, again an array, where it will push all the selected elements.

5.26.6 Add your Directive to the User Interface

Now we have all the elements, we need to add our control to the Marketplace Application User Interface page. The file is in **partials/wasdiapplicationui.html**. This page just show in a cycle all the controls requested by the developer in the UI of the app. The page will take care to show or hide the different controls. All we have to do is add in the “TAB CONTENT” section, our directive:

```
<!--list box-->
<div class="col-xs-12 col-md-10 col-lg-8 border-bottom py-2"
    ng-if="viewElement.type === 'listbox'">
  <div class="input-text-label pt-2">{{viewElement.label}}</div>
  <waplistbox options="viewElement.aoElements" selected="viewElement.aoSelected"
    tooltip="viewElement.tooltip"></waplistbox>
</div>
```

As you can notice, the directive receive in input the ViewElement instanced with the code we wrote before. So here is request to us to use our own directive with our own ViewElement Object, to initialize the control and retrieve back the value inserted by the user.

In our case is done in this snippet:

```
<waplistbox options="viewElement.aoElements" selected="viewElement.aoSelected">
```

Where we put as options of the directive the elements we got from the JSON and we ask to save the output in aoSelected, that will be used by our ListBox class in the getValue method.

5.26.7 Add a button to the online editor

To help our developers, there is a very basic on line editor of the UI. There every control has a button that the Developer can click to see a mockup of the json required to define that control.

The file is a dialog and can be found:

dialogs/processor/TabUIProcessor.html

and

dialogs/processor/ProcessorController.js

In the html we just need to add our button:

```
<div class="addUIElementCommand" ng-click="m_oController.addUIElement('listbox')">List_
  ↪Box</div>
```

In the JS, is again simple: just add in the **addUIElement** your pre-defined json sample:

```
else if (sElementType === "listbox") {
  sTextToInsert = '\n\t{\n\t\t"param": "PARAM_NAME",\n\t\t"type": "listbox",\n\t\t\t
  ↪"label": "description",\n\t\t\t"values": [],\n\t\t\t"required": false,\n\t\t\t"tooltip": ""\n\t
  ↪t},';
}
```

That's it, you created a new User Interface Control for WASDI Applications!!
Welcome to Space, Have fun!

TERMS AND CONDITIONS

Please, before start using WASDI, check our terms and condition.

6.1 EULA

6.1.1 1. Introduction

1.1. WASDI is a cloud service provided by WASDI Sarl (“WASDI”, “we”, or “us”). By using the WASDI platform services (“Services”), you agree to be bound by the following Terms of Service and any future modifications (collectively, the “Terms”).

1.2. **Please read these Terms carefully:** This end user agreement represents a legal between (a) you (either an individual or single entity) and (b) use (WASDI) that governs the use of our Services. **If you use the Services, you agreed to these Terms.**

6.1.2 2. Signing Up

2.1. You must register for a WASDI account to use the Services. You are responsible for all use of the Services under your account, whether or not authorized. At our discretion, we may make limited exceptions to this policy for unauthorized use of your account if you notify us of the problem in a timely manner.

2.2. If you are entering into this agreement on behalf of your company or another legal entity, you represent that you have the authority to bind that entity to these Terms.

6.1.3 3. Our Services

3.1. WASDI Services are available to you in accordance with these Terms. We grant you a non-exclusive, revocable license and right to: a. Use the Services within 3rd party applications; and b. Use the Services to develop your own WASDI applications; c. Use the Services with a Machine-to-Machine connection with third party systems;

3.2. Within the trial period and within non-commercial plans you may only use services for non-commercial purposes and for research. With paid subscription plans, you may use services for both commercial and non-commercial purposes.

6.1.4 4. Using WASDI Subscriptions

- 4.1. Starting from 18 MAY 2023 WASDI has implemented a subscription-based agreement for using WASDI services. From 18 MAY 2023 onwards, when a user creates a new WASDI account they will be granted rights to a FREE 3 MONTH (90 DAYS) subscription at the STANDARD LEVEL (for information on subscription levels, see sections 5.2 and 5.3). The subscription begins from the first subscription-only action the user executes and continues for the specified amount of time calculated in milliseconds (for information on subscription-only actions, see section 4.7).
- 4.2. Users who have purchased a Standard Subscription understand and acknowledge that their Node is shared and priority may be granted to Professional Subscription holders at any time in the course of the user's WASDI actions.
- 4.3. The User must have an active project and valid subscription in WASDI to perform any subscription-only actions. When a subscription profile is created, an active project for that subscription is created automatically.
- 4.4. Subscriptions are shareable between users within an organization, but they are not TRANSFERABLE. If you need to transfer a long term subscription (1 Month or 1 Year) please contact WASDI directly.
- 4.5. By purchasing and using a WASDI subscription, you are acknowledging that if you are found to be in violation of any of WASDI's terms of service particularly as outlined in section 9, the privileges associated with your subscription will be revoked and no refund will be issued. See section 12 for further information on Account Termination or Suspension.
- 4.6. If there is a major interruption during the course of a short term subscription (1 Day or 1 Week) the amount of time the interruption caused will be added to the remaining time of the subscription. For example, if the last two hours of a 1 Day subscription were interrupted, those 2 hours will be added to the time remaining in that specific subscription.
- 4.7. Subscription-only actions are actions in WASDI that are solely available to holders of VALID subscriptions with an active project. The first execution of any of these actions will trigger your subscription to begin. These actions are as follows: a. Executing a Processor in a workspace (an existing workspace or a new one); b. Opening a Jupyter Notebook in a workspace; c. Importing a product to a workspace; d. Executing a workflow in a workspace WASDI retains the right to add subscription-only actions to this list at any time.

6.1.5 5. Purchasing WASDI Subscriptions

- 5.1. After the 3 months of free access, WASDI offers two levels of subscription, STANDARD (STND) and PROFESSIONAL (PRO). For information on the differentiation, please visit our pricing page at either [our information page](#) or log into an existing WASDI account and navigate to the [Subscriptions Page](#).
- 5.2. Short Term and STANDARD (STND) Subscriptions are purchased through our payment partner, STRIPE. When the user completes a payment through Stripe, they are not only agreeing to act in compliance with WASDI's terms of service, but also that of Stripe.
- 5.3. PROFESSIONAL (PRO) Subscriptions are only available through contacting WASDI. The clauses pertaining to refunding (4.6, and 4.7) are not applicable to PROFESSIONAL subscriptions. Users wishing to purchase a PROFESSIONAL subscription must contact WASDI directly for pricing and user obligations.
- 5.4. VAT is calculated at checkout through STRIPE based on the user's location. The VAT calculation shown on the WASDI website is based on a Luxembourgish user. If the purchasing user is located elsewhere, you are acknowledging that your VAT rate may differ. When you complete a transaction on our Payment Partner's website, you are confirming that all information, including your location is true and accurate.
- 5.5. When checking out through our payment partner, you are agreeing to WASDI's terms of use for subscriptions which are outlined in SECTION 4.
- 5.6. STANDARD Subscriptions can be purchased in increments of 1 Day (24 Hours), 1 Week (7 Days), 1 Month (30 Days), or 1 Year (365 days). These times are calculated in milliseconds beginning at the first execution of a subscription-only action.

5.7. Short term subscription plans (i.e., 1 Day and 1 Week) will be refundable if they are PAID but unused. An unused subscription is one where no subscription-only action has been executed. This refund can be claimed within 14 days of payment. After 14 days, issuing a refund is at the discretion of WASDI.

5.8. Executed and partially complete subscriptions of any kind are not refundable. If you have executed a subscription-only action you have agreed to waive your right to a refund.

6.1.6 6. Service Hours and Exceptions

6.1. The Service is delivered 24 hours per day, 7 days per week (i.e., 365 days or 8,760 hours per year), to seamlessly support business operations.

6.2. Planned and announced interruptions may reduce the effective operating time of the Service.

6.3. The following exceptions with respect to the Service Operations apply: a. Planned and agreed interruptions (e.g., for maintenance) are not considered as unavailability of the service, since they are not part of the effective operating time. Maintenance windows or other planned interruptions will be announced with a lead time of at least 3 working days. b. Outages or interruptions from third party services on which the Service Operations rely are not considered as unavailability of the service, since these outages are out of the control of WASDI Sarl.

6.4. Additionally, WASDI reserves the right to temporarily suspend the service in whole or in part in case of: a. Detected security threats or vulnerability of the Service or individual Service Components; b. Evidence of fraudulent intent or misuse of the Service or individual Service Components; c. Infringements with respect to third party agreements imposed on the Service or Service Components as well as violations to the present agreement; d. Adversely affecting other Service Provider services, services of the Service Provider customers or any Service Provider customer activities; and e. Violations with reference to payment obligations inherent with the Service delivery. f. In the event of a temporary suspension of the Service, you remain responsible for all fees and charges incurred during the period of suspension.

6.1.7 7. Service Guarantees

7.1. The Service will be available with a minimum availability of 95%. However, a single service interruption will have a duration of fewer than 72 hours (weekdays only). Scheduled maintenance downtime is announced with a notification period of 3 business days. For further information as to what constitutes a Service interruption, see section 6.3.

7.2. If the minimum availability (§7.1) is not provided by the Service Provider, the Service Provider will grant a service credit in the form of time lost added to the end of any short-term (1 Day or 1 Week Subscription).

7.3. To receive a credit, the Service User must contact the Service Provider within 30 days following the end of the unavailability via email at the address provided in §7.6 and include the dates and times of unavailability.

7.4. If the Service Provider confirms that the uptime percentage covered by the Service User request is below the minimum availability (§7.1), the Service Provider will issue the Service User a service credit. The service credit is added to the end of the Service User's term for the Service, and cannot be exchanged for, or converted to, monetary compensation.

7.5. The WASDI Service Level Agreement (SLA) includes the provision of a chat support in a public Discord Channel from Monday to Friday, from 9:00AM to 7:00PM CET.

7.6. Direct support requests should be sent via e-mail to: info@wasdi.cloud

6.1.8 8. Technical and Performance

8.1. WASDI runs on different cloud environments. The performance and availability of each WASDI node is regulated by the SLA of the cloud provider that is hosting the node. WASDI can only guarantee the backup of the history of the operations done in that node. Users are able to find out in which cloud environment the services are running and access that cloud provider's SLA with a link if made available by the provider.

8.2. In case of any accident in a cloud environment, WASDI cannot be considered responsible and will guarantee only what stated in (§6.1). Regardless, WASDI will contact the cloud provider to get the best possible conditions to repair the damage. In case of a re-found from the Cloud Provider this will be proportionally distributed to the users impacted by the incident, in terms of processors that had been started in that node and files that were stored in that node.

6.1.9 9. Unlawful or Unauthorized Uses

9.1. You may not use the Services for any unlawful purpose. Your use of the Services must comply with all local rules regarding online conduct and acceptable content.

9.2. You may not use the Services in any manner that could damage or overburden the Services or interfere with any other party's use of the Services.

9.3. You may not engage in other unacceptable uses of the Services, which include but are not limited to: a. Disseminating material that is abusive, obscene, pornographic, defamatory, harassing, grossly offensive, vulgar, threatening or malicious; b. Aiding or implementing practices violating basic human rights or civil liberties; c. Disseminating or storing material that infringes the copyright, trademark, patent, trade secret, or other intellectual property rights of any person; d. Creating a false identity or otherwise attempting to mislead others as to the identity or origin of any communication; e. Exporting, re-exporting, or permitting downloading of any content in violation of any export or import law, regulation, or restriction of the European Union and its agencies or authorities, or without all required approvals, licenses, or exceptions; f. Interfering with or attempting to gain unauthorized access to any computer network; g. Transmitting viruses, Trojan horses, or any other malicious code or program; or h. Engaging in any other activity deemed by WASDI to be in conflict with the spirit or intent of these Terms.

6.1.10 10. User-Supplied Applications

10.1. Any user-supplied application remains a full property of the user.

10.2. Limited to the purpose of hosting your content so that we can provide the Services to you, you hereby grant WASDI a non-exclusive, worldwide, royalty-free, transferable right and license (with the right to sublicense), to host, copy and back-up your code.

10.3. If you decide to set your application as public, or you share it with other Users, you also grant WASDI a to non-exclusive, worldwide, royalty-free, transferable right and license (with the right to sublicense), to use, copy, cache, publish, display, distribute and store such content. This right and license enables WASDI to host and mirror your content on its distributed platform. You warrant, represent, and agree that you have the right to grant WASDI these aforementioned rights.

10.4. On termination of your account WASDI will make all reasonable efforts to promptly remove from the site and cease use of your content; however, you recognize and agree that caching of or references to the content may not be immediately removed.

6.1.11 11. Third-Party Applications

11.1. WASDI itself hosts and offers the platform to the client (user). Different third-party applications can be found in the platform. Each application has its own SLA. WASDI is not responsible for the performance and the results of any of the hosted applications.

11.2. WASDI Sarl, as the other third parties value adders, will release the SLA of its own applications.

11.3. The Intellectual Property of the applications uploaded in WASDI will remain of the user that uploaded the application. The publisher can choose if the application will be private, shared with other users or public. The publisher will choose if the application is free or has a cost.

11.4. Other than your content, all content accessible through the Services, including text, graphics, maps, logos, images, illustrations, software or source code, audio and video, and animations, are all property of WASDI and/or third parties and are protected by Luxembourgish and international copyright law. You may be held liable for any unauthorized copying or disclosure of this content. You agree that WASDI's licensors shall be third-party beneficiaries to these Terms and that these companies may directly enforce, and may rely upon, any provision of the Terms that confers a benefit on them or grants rights in their favor.

11.5. All logos and product names appearing on or in connection with the Services are proprietary to WASDI and/or its licensors and/or suppliers. You may not remove any proprietary notices or product identification labels from the Services' software, maps, or other content.

11.6. In case of a third application sold in WASDI, a revenue sharing mechanism is foreseen. The amount of the application will go to the developer and a percentage to WASDI.

6.1.12 12. Account Termination or Suspension

12.1. Your WASDI account may be terminated by you at any time. However, we do not give pro-rated refunds for unused time if you cancel during a billing cycle.

20.2. The limited license granted by this agreement terminates automatically, without notice to you, if you breach any of these Terms.

12.3. Additionally, WASDI may cancel or suspend your account for any reason by providing you with thirty days' advance notice. Upon cancellation or suspension, your right to use the Services will cease immediately. You may not have access to data that you had stored on the site after we cancel or suspend your account. You are responsible for backing up data that you use with the Services. If we cancel your account in its entirety without cause, we will refund you on a pro-rata basis the amount of your payment corresponding to the portion of your Service remaining right before we cancelled your account.

6.1.13 13. Changes to Terms of Service

13.1. We reserve the right to modify these Terms at any time by posting the changed terms on the WASDI website. All changes shall be effective immediately upon posting. Please check these Terms periodically for changes. Your continued use of the Services after we post any changes constitutes your binding acceptance of the new terms.

13.2. We may change the features and functions of the Services and the terms of the SLA may change over time.

6.1.14 14. Indemnification

14.1. By using the platform, the user agrees to hold harmless WASDI, its subsidiaries, affiliates, officers, agents, partners and employees for any claim or demand, including reasonable attorneys' fees arising out of: i. Your use of the Services; ii. Your violation of these Terms; iii. Your end users' use of the Services in or through an application or service that you provide; iv. Content you or your end users submit, post to, extracts from, or transmit through the Services.

6.1.15 15. Data Handling and Retention

15.1. The platform keeps a backup of the database which contains the history of all the operations that the user has done in WASDI.

15.2. The platform does not save any backup copy of the users' workspaces or of the files contained in the workspaces.

15.3. WASDI offers a "WASDI-ASSURANCE" service that is able to re-create the workspaces in case of an accident. This service does not include files uploaded directly by the user and all the files that can be derived through elaboration from files uploaded directly by the user.

15.4. On the **free** account type, WASDI reserves the right to delete the user workspaces after a reasonable period of 2 months.

15.5. WASDI Sarl keeps the user's email and the user-supplied name as personal data. The email is the user id and is used to reconstruct the history of the processes ran by the user, the list of its workspaces, workflows, applications, and files.

15.6. WASDI reserves the right to notify the user in case of Foreseen Maintenance Downtimes and/or Major Updates by email. This newsletter is elective, thus the user may choose not to receive it.

15.7. WASDI makes a backup copy of users' processors and workflows. The backup runs once per day and processors and workflows are copied on at least one node in a different cloud environment. The result cannot in any way be guaranteed and WASDI strongly suggests that users create a local backup copy of their own applications and workflows.

6.1.16 16. Disclaimer

16.1. YOU EXPRESSLY AGREE THAT THE USE OF THE SITE IS AT YOUR SOLE RISK. THE SITE AND ITS SOFTWARE, SERVICES, MAPS, AND OTHER CONTENT, INCLUDING ANY THIRD-PARTY SOFTWARE, SERVICES, MEDIA, OR OTHER CONTENT MADE AVAILABLE IN CONJUNCTION WITH OR THROUGH THE SITE, ARE PROVIDED ON AN "AS IS", "AS AVAILABLE", "WITH ALL FAULTS" BASIS AND WITHOUT WARRANTIES OR REPRESENTATIONS OF ANY KIND, EITHER EXPRESS OR IMPLIED.

16.2. TO THE FULLEST EXTENT PERMISSIBLE PURSUANT TO APPLICABLE LAW, WASDI DISCLAIMS ALL WARRANTIES, STATUTORY, EXPRESS OR IMPLIED, INCLUDING IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, TITLE, AND NON- INFRINGEMENT OF PROPRIETARY RIGHTS. NO ADVICE OR INFORMATION, WHETHER ORAL OR WRITTEN, OBTAINED BY YOU FROM WASDI OR THROUGH THE SITE, WILL CREATE ANY WARRANTY NOT EXPRESSLY STATED HEREIN.

16.3. WASDI DOES NOT WARRANT THAT THE SITE, INCLUDING ANY SOFTWARE, SERVICES, MAPS, OR CONTENT OFFERED ON OR THROUGH THE SITE OR ANY THIRD-PARTY SITES REFERRED TO ON OR BY THE SITE WILL BE UNINTERRUPTED, OR FREE OF ERRORS, VIRUSES, OR OTHER HARMFUL COMPONENTS AND DOES NOT WARRANT THAT ANY OF THE FOREGOING WILL BE CORRECTED.

16.4. WHEN USING THE SERVICES, YOU MAY BE EXPOSED TO USER SUBMISSIONS AND OTHER THIRD-PARTY CONTENT ("NON-WASDI CONTENT"), AND SOME OF THIS CONTENT MAY BE INACCURATE, OFFENSIVE, INDECENT, OR OTHERWISE OBJECTIONABLE. WE DO NOT ENDORSE ANY NON-WASDI CONTENT. UNDER NO CIRCUMSTANCES WILL WASDI BE LIABLE FOR OR IN CONNECTION WITH THE NON-

WASDI CONTENT, INCLUDING FOR ANY INACCURACIES, ERRORS, OR OMISSIONS IN ANY NON-WASDI CONTENT, ANY INTELLECTUAL PROPERTY INFRINGEMENT WITH REGARD TO ANY NON-WASDI CONTENT, OR FOR ANY LOSS OR DAMAGE OF ANY KIND INCURRED AS A RESULT OF THE USE OF ANY NON-WASDI CONTENT.

16.5. WASDI DOES NOT WARRANT OR MAKE ANY REPRESENTATIONS REGARDING THE USE OR THE RESULTS OF THE USE OF THE SITE OR ANY THIRD-PARTY SITES REFERRED TO ON OR BY THE SITE IN TERMS OF CORRECTNESS, ACCURACY, RELIABILITY, OR OTHERWISE.

16.6. YOU UNDERSTAND AND AGREE THAT YOU USE, ACCESS, DOWNLOAD, OR OTHERWISE OBTAIN SOFTWARE, SERVICES, MAPS, OR CONTENT TO YOUR OWN DISCRETION AND RISK AND THAT YOU WILL BE SOLELY RESPONSIBLE FOR ANY DAMAGE TO YOUR PROPERTY (INCLUDING YOUR COMPUTER SYSTEM) OR LOSS OF DATA THAT RESULTS FROM SUCH DOWNLOAD OR USE.

16.7. CERTAIN JURISDICTIONS DO NOT ALLOW LIMITATIONS ON IMPLIED WARRANTIES OR THE EXCLUSION OR LIMITATION OF CERTAIN DAMAGES. IF YOU RESIDE IN SUCH A JURISDICTION, SOME OR ALL OF THE ABOVE DISCLAIMERS, EXCLUSIONS, OR LIMITATIONS MAY NOT APPLY TO YOU, AND YOU MAY HAVE ADDITIONAL RIGHTS. THE LIMITATIONS OR EXCLUSIONS OF WARRANTIES, REMEDIES, OR LIABILITY CONTAINED IN THESE TERMS APPLY TO YOU TO THE FULLEST EXTENT SUCH LIMITATIONS OR EXCLUSIONS ARE PERMITTED UNDER THE LAWS OF THE JURISDICTION IN WHICH YOU ARE LOCATED.

6.1.17 17. Limitation of Liability

17.1. UNDER NO CIRCUMSTANCES, AND UNDER NO LEGAL THEORY, INCLUDING NEGLIGENCE, SHALL WASDI OR ITS AFFILIATES, CONTRACTORS, EMPLOYEES, AGENTS, OR THIRD-PARTY PARTNERS OR SUPPLIERS, BE LIABLE FOR ANY SPECIAL, INDIRECT, INCIDENTAL, CONSEQUENTIAL, OR EXEMPLARY DAMAGES (INCLUDING LOSS OF PROFITS, DATA, OR USE OR COST OF COVER) ARISING OUT OF OR RELATING TO THESE TERMS OR THAT RESULT FROM YOUR USE OR THE INABILITY TO USE THE SITE, INCLUDING SOFTWARE, SERVICES, MAPS, CONTENT, USER SUBMISSIONS, OR ANY THIRD-PARTY SITES REFERRED TO ON OR BY THE SITE, EVEN IF WASDI OR A WASDI AUTHORIZED REPRESENTATIVE HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

17.2. IN NO EVENT SHALL THE TOTAL LIABILITY OF WASDI OR ITS AFFILIATES, CONTRACTORS, EMPLOYEES, AGENTS, OR THIRD-PARTY PARTNERS, LICENSORS, OR SUPPLIERS TO YOU FOR ALL DAMAGES, LOSSES, AND CAUSES OF ACTION ARISING OUT OF OR RELATING TO THESE TERMS OR YOUR USE OF THE SITE (WHETHER IN CONTRACT, TORT (INCLUDING NEGLIGENCE), WARRANTY, OR OTHERWISE) EXCEED THE GREATER OF ONE HUNDRED EURO (100 EUR) OR FEES PAID OR PAYABLE TO WASDI IN THE TWELVE MONTHS PERIOD PRIOR TO THE DATE ON WHICH THE DAMAGE OCCURRED.

17.3. THESE LIMITATIONS SHALL ALSO APPLY WITH RESPECT TO DAMAGES INCURRED BY REASON OF ANY PRODUCTS OR SERVICES SOLD OR PROVIDED ON ANY THIRD-PARTY SITES REFERRED TO ON OR BY THE SITE OR OTHERWISE BY THIRD PARTIES OTHER THAN WASDI AND RECEIVED BY YOU THROUGH OR ADVERTISED ON THE SITE OR RECEIVED BY YOU THROUGH ANY THIRD-PARTY SITES.

17.4 YOU AND WASDI AGREE THAT ANY CAUSE OF ACTION ARISING OUT OF THESE TERMS OR RELATED TO WASDI MUST COMMENCE WITHIN ONE (1) YEAR AFTER THE CAUSE OF ACTION ACCRUES. OTHERWISE, SUCH CAUSE OF ACTION IS PERMANENTLY BARRED.

6.2 Privacy policy

6.2.1 What Personal Data We Collect and Why We Collect it

WASDI Sarl (“WASDI”) is committed to preserving the privacy of all users of our platform, attendees of our workshop, and any other individuals who have entrusted us with their personal data. This privacy policy is constructed in plain language to help you understand how any of your personally identifiable information is collected, stored, and disclosed when you engage with us. Before engaging with us in any capacity (e.g., registering your information or submitting any content to the platform) we ask that you ensure your understanding of this privacy policy as well as our End User Licensing Agreement (EULA).

In general, when engaging with WASDI, you are under no obligation to submit any personal information, however, your refusal to do so may limit our ability to provide you with certain services as well as access to the WASDI platform. WASDI does not collect, use, store, or disclose any personal information without your consent, except as required by law.

WASDI will identify the purposes for which personal information is collected at or before the time it is collected. If we seek to use collected personal information for other purposes, we will seek your consent prior to use, unless it is otherwise permitted or required by law.

If you have any objections to the terms of this Privacy Policy, you should not use our Services or provide us with any of your personal information, as providing your personal information to us you will be deemed to have consented to the processing of such data. If you have any hesitations or are unsure about any of the terms we invite you to contact us with questions at the contact details provided below.

6.2.2 PART A - COLLECTION OF PERSONAL INFORMATION

1. Information You Provide

We collect the personal information that you provide to us, which may include, but is not limited to: basic personal information, including your name and contact information;

- information about third parties with whom you are sharing content using our Services and information about your use of our Services;
- analytical information including by not limited to, your login data, browser type and version, time zone setting and location, browser plug-in and versions, operating system and platform and other technology on the devices you use to access our website and application;
- information collected in a dispute resolution if applicable;
- any feedback you provide regarding our products and services; and
- geographic and special areas that you monitor
- satellite data sources you use.

2. Automatic Information

- We may collect and store anonymous information (“Automatic Information”) about you as you engage with our Services. This information may be collected using various technologies including, without limitation, cookies, Internet tags and navigational data. Automatic Information will not be published or made public through our Services.
- Some of this information is automatically transmitted to us through your browser, such as the URL that you just visited, the browser version that your computer uses, the date etc. Cookies are alphanumeric identifiers that are stored on your device with which you access the Services. These cookies allow us to adjust our Services to meet your personal browsing preferences. If your browser is not set to allow cookies, you may not have access to all areas or features of the website.
- You can set your browser to refuse all or some browser cookies, or to alert you when websites set or access cookies. If you disable or refuse cookies, please note that some parts of our website may become inaccessible or not function properly.

6.2.3 PART B – USE OF YOUR INFORMATION

To accomplish any of the purposes listed below, we may combine personal information, Automatic Information, and non-personal information collected through your use of our Services. This information may be used in a variety of ways including, but not limited to:

- to connect you to members of our team;
- to enforce our Terms;
- to prevent, detect and investigate potentially prohibited or illegal activities, fraud and security breaches;
- to collect data on our Services;
- to improve our Services;
- to contact you for the purpose of marketing, promotional offers, updates and any other purpose set out in our Privacy Policy and as permitted by the preferences you have indicated;
- to analyze trends, administer the Services, track your website navigation and gather broad demographic information;
- personalize your experience while using our Services; and
- to provide you with any additional information you may request.

Except as set forth above, your personal information will not be provided to any other users of our Services. We reserve the right to use personal information and Automatic Information as set forth in our Terms, if applicable.

6.2.4 PART C - DISCLOSURE OF YOUR INFORMATION

1. To Help with Business Operations

We may disclose your personal information to our representatives, affiliates, suppliers or other third parties with whom we do business (our “Business Partners”) when it is deemed necessary for our business operations. For example, from time to time, we may engage companies or individuals for certain services on our behalf including, without limitation, conducting audits, performing legal services, processing credit card payments, collecting feedback, and providing customer service. Our Business Partners and any other third-party service providers will only have incidental access to personal information to the extent required to perform their services. They are prohibited from using your personal information for any purpose other than assisting with our business operations.

2. Safety and Security

We may disclose your information as necessary to protect the safety or security of users of our Services, to detect and prevent fraud or abusive behaviour, or when otherwise required or permitted by law.

3. Legal Requirements

We may disclose your information, if required, to:

- supervisory authorities, tax authorities, police and other regulatory authorities as required by law or in response to a subpoena, court order, or other legally valid inquiry or order; or
- to either prevent/minimize a criminal offense or to protect a person's rights and personal and or financial health.

4. Consent

In addition to permissions already granted under this Privacy Policy by you, we may provide, disclose or transfer your personal information with your consent provided either as a part of the Services or in writing by you otherwise.

6.2.5 PART D - PROTECTING, RETAINING, AND STORING YOUR INFORMATION

1. PROTECTION

In order to help prevent unauthorized access to, maintain data accuracy and ensure the correct use of your information, we have put in place commercially reasonable and industry accepted physical, electronic and managerial procedures to safeguard and secure the information we collect through our Services. That said, we cannot guarantee that information provided to us through our Services will remain private, or that the information you transmit to us over the Internet will not be intercepted.

2. STORAGE

Your information is currently stored in various servers located in the European Union as well other jurisdictions in which our Services' technology platform resides. However, we retain the right to transfer your information to an affiliate or third party to protect the information or for internal business reasons as we see fit. BY AGREEING TO THIS PRIVACY POLICY, YOU AGREE AND CONSENT TO US DOING SO WITHOUT YOUR FURTHER CONSENT.

3. IDENTIFICATION

To help prevent unauthorized access to your personal information you are responsible for keeping your username and password private and confidential. You are solely responsible for preventing the unauthorized use of your ID. If you suspect that your information has been compromised or that your account has been accessed without your consent, please immediately contact us at the contact information provided below.

4. RETENTION

We may store your personal information until the agreement which governs your use of the Services is terminated, or as necessary to comply with our legal obligations, resolve disputes and enforce our agreements. We will use commercially reasonable efforts to delete your personal information and any other information you may have provided to us in a reasonable time frame.

5. WITHDRAWAL OF CONSENT

If you wish to cancel your account or revoke your consent for the collection, use or disclosure of your personal information at any time, please contact us at the contact information provided below. Your withdrawal of consent is not retroactive, since we may already have used your personal information for the purposes described herein, it will be applied on a go-forward basis.

6.2.6 PART E - ELECTRONIC COMMUNICATIONS

Should you submit an inquiry through our Services, or otherwise communicate with us electronically, you consent to us responding to such inquiries electronically.

In addition, with your consent we may use personal information to contact you with our customer support materials or with notices about promotions, sales, new features made available, service interruptions (planned and unplanned). If you would like to opt out of future promotional materials from us, you may indicate this preference by selecting the unsubscribe instructions at the end of our electronic communications. Please note that if you opt out of messages from us, we may continue to send you account-related updates so as to continue to support your account.

6.2.7 PART F - ACCESSING AND MODIFYING YOUR PERSONAL AND ACCOUNT INFORMATION

You can access and modify your personal and account information using the tools provided through our Services. We will not modify your personal or account information.

At any time, you may request access to or removal of your personal information by contacting us at the contact information provided below. We will respond to your request within a reasonable time period, and process it as appropriate under applicable law. Your request may be subject to a processing fee as allowed under law.

6.2.8 PART G - GDPR COMPLIANCE

As WASDI is headquartered in Luxembourg, if we collect, track, use or process in some other way your personal information collected from you or through your use of our website, or we transfer that personal information, we will do so in accordance with this Privacy Policy, our End User Licensing Agreement, and in compliance with applicable requirements of the General Data Protection Regulation (EU 2016/679) (“GDPR”).

1. Transfers of Personal Information.

WASDI is a data controller and responsible for your personal information, which WASDI processes and stores in the European Union. WASDI uses the following safeguards when transferring your personal information to a country that is not within the EEA:

- (a) Only transfer your personal data only to countries that have been deemed to provide an adequate level of protection for personal information by the European Commission.
- (b) Where your personal data is transferred to a country that is not deemed to have an adequate level of protection, we will ensure that our service providers contractually agree to implement measures that will ensure that your personal data has the same protection it has in the EU.

2. Opt-in

If you are an EU resident, we may only collect your data using cookies and similar devices, and then track and use your personal information where you have first consented to that. We will not automatically collect personal information from you as described above unless you have consented to us doing so. If you consent to our use of cookies and similar devices, you may at a later date disable them (please see above). Your Legal Rights Under certain circumstances, you may have rights under the data protection laws in relation to your personal information, including the right to:

- Request access to your personal information.
- Request correction of your personal information.
- Request erasure of your personal information.
- Object to processing of your personal information.
- Request restriction of processing your personal information.
- Request transfer of your personal information.
- Right to withdraw consent.

If you wish to exercise any of these rights, please contact our Privacy Officer to find out more information about what we may need from you and the time in which we should respond. Data Protection Officer We are required by the GDPR to have a data protection officer. The person who has that role is our Privacy Officer whose details are set out below.

6.2.9 PART H - CHANGES TO THIS PRIVACY POLICY

We reserve the right to change this Privacy Policy, and any other policies and procedures concerning our practices for managing personal information, at any time without prior notice to you. If this Privacy Policy is modified, we will post the most current version to our website (www.wasdi.cloud). At the top of the modified Privacy Policy we will include the date upon which it was last updated. Any changes that are made to this Privacy Policy will apply to both personal information that we hold prior to the effective date of the amended Privacy Policy and to any personal information collected on or after such effective date. Our successors and assigns may collect and use your personal information for substantially similar purposes as described in this Privacy Policy.

6.2.10 PART I - HOW TO CONTACT OUR PRIVACY OFFICER

Any questions, comments or concerns relating to this Privacy Policy, and any requests to correct or access personal information collected during your use of the Services, should be directed to the Privacy Officer at:

WASDI Sarl 100 route de Volmerange L-3593 Dudelange Luxembourg Attention: Privacy Officer Email: info@wasdi.cloud

Telephone Number: +352 206005 6301

Last Revised: 7 November 2023

PYTHON MODULE INDEX

W

wasdi, [364](#)

Symbols

[_fileOnNode\(\)](#) (in module *wasdi*), 364
[_getStandardHeaders\(\)](#) (in module *wasdi*), 363
[_internalExecuteWorkflow\(\)](#) (in module *wasdi*), 364
[_loadConfig\(\)](#) (in module *wasdi*), 363
[_loadParams\(\)](#) (in module *wasdi*), 363
[_log\(\)](#) (in module *wasdi*), 363
[_normPath\(\)](#) (in module *wasdi*), 363
[_unzip\(\)](#) (in module *wasdi*), 363

A

[addFileToWASDI\(\)](#) (in module *octavewasdilib*), 337
[AddFileToWASDI\(string\)](#) (Java method), 258, 274
[addFileToWASDI\(String\)](#) (Java method), 290
[AddFileToWASDI\(string, string\)](#) (Java method), 273
[AddParam\(string, string\)](#) (Java method), 262
[addParam\(String, String\)](#) (Java method), 290
[AsynchAddFileToWASDI\(string\)](#) (Java method), 274
[asynchAddFileToWASDI\(String\)](#) (Java method), 291
[AsynchAddFileToWASDI\(string, string\)](#) (Java method), 273
[AsynchExecuteProcessor\(string, Dictionary\)](#) (Java method), 284
[asynchExecuteProcessor\(String, HashMap\)](#) (Java method), 291
[AsynchExecuteProcessor\(string, string\)](#) (Java method), 284
[asynchExecuteProcessor\(String, String\)](#) (Java method), 291
[AsynchExecuteWorkflow\(List, List, string\)](#) (Java method), 270
[asynchExecuteWorkflow\(String\[\], String\[\], String\)](#) (Java method), 291
[AsynchImportProduct\(Dictionary\)](#) (Java method), 278
[AsynchImportProduct\(Dictionary, string\)](#) (Java method), 278
[AsynchImportProduct\(string, string\)](#) (Java method), 280
[AsynchImportProduct\(string, string, string\)](#) (Java method), 280

[AsynchImportProduct\(string, string, string, string\)](#) (Java method), 280
[AsynchImportProductList\(List, List\)](#) (Java method), 281
[AsynchImportProductListWithMaps\(List\)](#) (Java method), 281
[asynchMosaic\(List, String\)](#) (Java method), 292
[asynchMosaic\(List, String, String, String\)](#) (Java method), 292
[asynchMosaic\(List, String, String, String, List\)](#) (Java method), 292
[asynchMosaic\(List, String, String, String, List, double, double\)](#) (Java method), 292
[asynchMosaic\(List, String, String, String, List, double, double, String\)](#) (Java method), 293
[asynchMosaic\(List, String, String, String, List, double, double, String, double, double, double, double, String, boolean, String, String, boolean, boolean, String\)](#) (Java method), 293
[AsynchPreprocessProductsOnceDownloaded\(List, string, string, List\)](#) (Java method), 268
[AsynchPreprocessProductsOnceDownloadedWithNames\(List, string, string, List\)](#) (Java method), 269

C

[checkBaseUrl\(\)](#) (built-in function), 365
[checkSession\(String\)](#) (Java method), 294
[CheckSession\(string, string\)](#) (Java method), 263
[CopyFileToSftp\(string\)](#) (Java method), 288, 289
[CopyFileToSftp\(string, string\)](#) (Java method), 289
[copyStream\(InputStream, OutputStream\)](#) (Java method), 294
[copyStreamAndClose\(InputStream, OutputStream\)](#) (Java method), 294
[CreateSession\(string, string\)](#) (Java method), 263
[createWorkspace\(\)](#) (built-in function), 366
[CreateWorkspace\(string\)](#) (Java method), 285

CreateWorkspace(string, string) (Java method),
285

D

DeleteProduct(string) (Java method), 285
deleteProduct(String) (Java method), 294
DeleteWorkspace(string) (Java method), 286
DownloadFile(string) (Java method), 288
downloadFile(String) (Java method), 295

E

executeProcessor() (built-in function), 367
ExecuteProcessor(string, Dictionary) (Java
method), 284
ExecuteProcessor(string, string) (Java method),
284
executeWorkflow() (in module octavewasdilib), 338
executeWorkflow(String[], String[], String)
(Java method), 295

G

GetActiveWorkspace() (Java method), 259
getActiveWorkspace() (Java method), 295
GetBasePath() (Java method), 261
getBasePath() (Java method), 295
GetBaseUrl() (Java method), 260
getBaseUrl() (Java method), 295
GetDefaultProvider() (Java method), 258
getDeployed() (built-in function), 367
GetDownloadActive() (Java method), 261
getDownloadActive() (Java method), 295
GetFoundProductFootprint(Dictionary) (Java
method), 278
GetFoundProductFootprint(QueryResult) (Java
method), 278
GetFoundProductLink(Dictionary) (Java method),
277
getFoundProductLink(Map) (Java method), 296
GetFoundProductLink(QueryResult) (Java method),
277
GetFoundProductName(Dictionary) (Java method),
277
getFoundProductName(Map) (Java method), 296
GetFoundProductName(QueryResult) (Java method),
277
getFullProductPath() (in module octavewasdilib),
338
getFullProductPath(String) (Java method), 296
GetIsOnServer() (Java method), 260
getIsOnServer() (Java method), 296
getLayerWMS() (built-in function), 368
GetMyProcId() (Java method), 261
getMyProcId() (Java method), 296
GetParam(string) (Java method), 263

getParam(String) (Java method), 297
GetParametersFilePath() (Java method), 263
getParametersFilePath() (Java method), 297
GetParams() (Java method), 262
getParams() (Java method), 297
GetParamsAsJsonString() (Java method), 262
GetPassword() (Java method), 259
getPassword() (Java method), 297
GetPath(string) (Java method), 269
getPath(String) (Java method), 297
GetProcessesByWorkspaceAsListOfJson(int,
Int32, string, string, string) (Java
method), 286, 287
GetProcessesStatus(List) (Java method), 271
GetProcessesStatusAsList(List) (Java method),
271
GetProcessorPath() (Java method), 285
getProcessorPath() (Java method), 298
GetProcessorPayload(string) (Java method), 287
GetProcessorPayloadAsJSON(string) (Java
method), 287
getProcessStatus() (built-in function), 367
getProcessStatus() (in module octavewasdilib), 338
GetProcessStatus(string) (Java method), 270
getProcessStatus(String) (Java method), 297
GetProcessWorkspacesByWorkspaceId(string)
(Java method), 286
GetProductBbox(string) (Java method), 288
GetProductName(Dictionary) (Java method), 267
getProductsByActiveWorkspace() (built-in func-
tion), 366
getProductsByActiveWorkspace() (Java method),
298
getProductsByWorkspace() (in module octavewas-
dilib), 339
GetProductsByWorkspace(string) (Java method),
266
getProductsByWorkspace(String) (Java method),
298
GetProductsByWorkspaceId() (Java method), 267
GetProductsByWorkspaceId(string) (Java method),
266
getSavePath() (in module octavewasdilib), 339
GetSavePath() (Java method), 269
getSavePath() (Java method), 298
GetSessionId() (Java method), 260
getSessionId() (Java method), 298
getStandardHeaders() (Java method), 298
getStreamingHeaders() (Java method), 299
GetUser() (Java method), 258
getUser() (Java method), 299
GetVerbose() (Java method), 262
getVerbose() (Java method), 299
getWorkflows() (in module octavewasdilib), 339

[GetWorkflows\(\)](#) (Java method), 270
[getWorkflows\(\)](#) (Java method), 299
[GetWorkspaceBaseUrl\(\)](#) (Java method), 264
[getWorkspaceBaseUrl\(\)](#) (Java method), 299
[GetWorkspaceIdByName\(string\)](#) (Java method), 264
[getWorkspaceIdByName\(String\)](#) (Java method), 299
[GetWorkspaceNameById\(string\)](#) (Java method), 265
[GetWorkspaceOwnerByName\(string\)](#) (Java method), 265
[getWorkspaceOwnerByName\(String\)](#) (Java method), 299
[GetWorkspaceOwnerByWSId\(string\)](#) (Java method), 265
[getWorkspaceOwnerByWSId\(String\)](#) (Java method), 300
[getWorkspaces\(\)](#) (built-in function), 366
[getWorkspaces\(\)](#) (in module *octavewasdilb*), 339
[GetWorkspaces\(\)](#) (Java method), 264
[getWorkspaces\(\)](#) (Java method), 300
[GetWorkspaceUrlByWsId\(string\)](#) (Java method), 265

H

[Hello\(\)](#) (Java method), 264
[helloWasdiWorld\(\)](#) (built-in function), 365
[httpGet\(String, Map\)](#) (Java method), 300
[httpPost\(String, String, Map\)](#) (Java method), 300

I

[ImportAndPreprocess\(List, string, string\)](#) (Java method), 268
[ImportAndPreprocess\(List, string, string, string\)](#) (Java method), 268
[ImportAndPreprocessWithLinks\(List, List, string, string\)](#) (Java method), 267
[ImportAndPreprocessWithLinks\(List, List, string, string, string\)](#) (Java method), 267
[ImportProduct\(Dictionary\)](#) (Java method), 279
[importProduct\(Map\)](#) (Java method), 301
[ImportProduct\(QueryResult\)](#) (Java method), 279
[importProduct\(String\)](#) (Java method), 301
[ImportProduct\(string, string\)](#) (Java method), 279
[importProduct\(String, String\)](#) (Java method), 301
[ImportProduct\(string, string, string\)](#) (Java method), 279
[ImportProduct\(string, string, string, string\)](#) (Java method), 280
[ImportProductList\(List, List\)](#) (Java method), 281

[ImportProductListWithMaps\(List\)](#) (Java method), 281
[init\(\)](#) (Java method), 302
[Init\(string\)](#) (Java method), 257
[init\(String\)](#) (Java method), 301
[internalAddFileToWASDI\(String, Boolean\)](#) (Java method), 302
[internalExecuteWorkflow\(String\[\], String\[\], String, Boolean\)](#) (Java method), 302
[InternalGetFullProductPath\(string\)](#) (Java method), 269
[internalInit\(\)](#) (Java method), 302
[InternalInit\(string\)](#) (Java method), 258
[internalMosaic\(boolean, List, String\)](#) (Java method), 302
[internalMosaic\(boolean, List, String, String, String\)](#) (Java method), 303
[internalMosaic\(boolean, List, String, String, String, List\)](#) (Java method), 303
[internalMosaic\(boolean, List, String, String, String, List, double, double\)](#) (Java method), 303
[internalMosaic\(boolean, List, String, String, String, List, double, double, String\)](#) (Java method), 304
[internalMosaic\(boolean, List, String, String, String, List, double, double, String, double, double, String, boolean, String, String, boolean, boolean, String\)](#) (Java method), 304

L

[loadConfig\(\)](#) (built-in function), 364
[loadParameters\(\)](#) (built-in function), 365
[log\(String\)](#) (Java method), 305
[login\(\)](#) (built-in function), 365
[login\(String, String\)](#) (Java method), 305

M

[m_sUser](#) (Java field), 257
[module](#)
 [wasdi](#), 363, 364
[Mosaic\(List, string\)](#) (Java method), 274, 275
[mosaic\(List, String\)](#) (Java method), 305
[Mosaic\(List, string, string, string\)](#) (Java method), 274, 275
[mosaic\(List, String, String, String\)](#) (Java method), 306
[Mosaic\(List, string, string, string, double, double\)](#) (Java method), 275, 276

mosaic(List, String, String, String, List)
(Java method), 306

mosaic(List, String, String, String, List,
double, double) (Java method), 306

mosaic(List, String, String, String, List,
double, double, String) (Java method),
307

mosaic(List, String, String, String, List,
double, double, String, double,
double, double, double, String,
boolean, String, String, boolean,
boolean, String) (Java method), 307

MultiSubset(string, List, List, List, List,
List) (Java method), 282, 283

MultiSubset(string, List, List, List, List,
List, bool) (Java method), 282, 283

O

openWorkspace() (built-in function), 366

openWorkspace() (in module octavewasdilib), 340

OpenWorkspace(string) (Java method), 266

openWorkspace(String) (Java method), 308

openWorkspaceById() (built-in function), 366

OpenWorkspaceById(string) (Java method), 266

P

PrintStatus() (Java method), 290

publishBand() (built-in function), 367

R

RefreshParameters() (Java method), 273

refreshParameters() (Java method), 308

S

s_oMapper (Java field), 290

SearchEOImages(string, string, string,
Double, Double, Double, Double,
string, int, string, string) (Java
method), 276

searchEOImages(String, String, String,
Double, Double, Double, Double,
String, Integer, String, String)
(Java method), 308

SetActiveWorkspace(string) (Java method), 259

setActiveWorkspace(String) (Java method), 309

SetBasePath(string) (Java method), 261

setBasePath(String) (Java method), 309

SetBaseUrl(string) (Java method), 260

setBaseUrl(String) (Java method), 309

SetDefaultProvider(string) (Java method), 258

SetDownloadActive(bool) (Java method), 261

setDownloadActive(Boolean) (Java method), 309

SetIsOnServer(bool) (Java method), 260

setIsOnServer(Boolean) (Java method), 309

SetMyProcId(string) (Java method), 261

setMyProcId(String) (Java method), 309

SetParametersFilePath(string) (Java method),
263

setParametersFilePath(String) (Java method),
310

SetPassword(string) (Java method), 259

setPassword(String) (Java method), 310

SetPayload(string) (Java method), 273

setProcessPayload() (built-in function), 367

setProcessPayload() (in module octavewasdilib), 340

SetProcessPayload(string, string) (Java
method), 273

setProcessPayload(String, String) (Java
method), 310

SetSessionId(string) (Java method), 260

setSessionId(String) (Java method), 310

SetSubPid(string, int) (Java method), 289

SetUser(string) (Java method), 259

setUser(String) (Java method), 310

SetVerbose(bool) (Java method), 262

setVerbose(Boolean) (Java method), 311

SetWorkspaceBaseUrl(string) (Java method), 264

setWorkspaceBaseUrl(String) (Java method), 311

startWasdi() (in module matlabwasdilib), 313

startWasdi() (in module octavewasdilib), 337

Subset(string, string, double, double,
double, double) (Java method), 282

subset(String, String, double, double,
double, double) (Java method), 311

U

updateProcessStatus() (in module octavewasdilib),
340

UpdateProcessStatus(string, string, int)
(Java method), 272

updateProcessStatus(String, String, int)
(Java method), 311

UpdateProgressPerc(int) (Java method), 272

updateProgressPerc(int) (Java method), 312

UpdateStatus(string) (Java method), 271

updateStatus(String) (Java method), 312

UpdateStatus(string, int) (Java method), 271

updateStatus(String, int) (Java method), 312

UploadFile(string) (Java method), 288

uploadFile(String) (Java method), 312

W

wAddFileToWASDI() (in module matlabwasdilib), 313

wAddParam() (in module matlabwasdilib), 314

waitForResume() (Java method), 313

waitProcess() (in module octavewasdilib), 341

WaitProcess(string) (Java method), 272

- `waitProcess(String)` (Java method), 313
- `WaitProcesses(List)` (Java method), 272
- `wasdi`
 - module, 363, 364
- `Wasdi` (Java class), 257
- `Wasdi()` (Java constructor), 257
- `wasdiHello()` (in module *matlabwasdilib*), 337
- `WasdiLib` (Java class), 290
- `WasdiLib()` (Java constructor), 290
- `wasdiLog()` (in module *matlabwasdilib*), 337
- `WasdiLog(string)` (Java method), 285
- `wAsynchAddFileToWASDI()` (in module *matlabwasdilib*), 314
- `wAsynchCopyFileToSftp()` (in module *matlabwasdilib*), 314
- `wAsynchExecuteProcessor()` (in module *matlabwasdilib*), 315
- `wAsynchExecuteWorkflow()` (in module *matlabwasdilib*), 315
- `wAsynchImportProduct()` (in module *matlabwasdilib*), 315
- `wAsynchImportProductList()` (in module *matlabwasdilib*), 316
- `wAsynchMosaic()` (in module *matlabwasdilib*), 316
- `wAsynchMultiSubset()` (in module *matlabwasdilib*), 317
- `wCopyFileToSftp()` (in module *matlabwasdilib*), 317
- `wCreateWorkspace()` (in module *matlabwasdilib*), 317
- `wDeleteProduct()` (in module *matlabwasdilib*), 318
- `wDeleteWorkspace()` (in module *matlabwasdilib*), 318
- `wExecuteProcessor()` (in module *matlabwasdilib*), 318
- `wExecuteWorkflow()` (in module *matlabwasdilib*), 319
- `wGetActiveWorkspace()` (in module *matlabwasdilib*), 319
- `wGetBasePath()` (in module *matlabwasdilib*), 319
- `wGetBaseUrl()` (in module *matlabwasdilib*), 320
- `wGetDownloadActive()` (in module *matlabwasdilib*), 320
- `wGetFullProductPath()` (in module *matlabwasdilib*), 320
- `wGetMyProcId()` (in module *matlabwasdilib*), 320
- `wGetParameter()` (in module *matlabwasdilib*), 321
- `wGetParametersFilePath()` (in module *matlabwasdilib*), 321
- `wGetParams()` (in module *matlabwasdilib*), 321
- `wGetPassword()` (in module *matlabwasdilib*), 321
- `wGetPath()` (in module *matlabwasdilib*), 322
- `wGetProcessesByWorkspace()` (in module *matlabwasdilib*), 322
- `wGetProcessorPath()` (in module *matlabwasdilib*), 323
- `wGetProcessorPayload()` (in module *matlabwasdilib*), 323
- `wGetProcessorPayloadAsJSON()` (in module *matlabwasdilib*), 323
- `wGetProcessStatus()` (in module *matlabwasdilib*), 322
- `wGetProductBbox()` (in module *matlabwasdilib*), 323
- `wGetProductsByActiveWorkspace()` (in module *matlabwasdilib*), 324
- `wGetProductsByWorkspace()` (in module *matlabwasdilib*), 324
- `wGetSavePath()` (in module *matlabwasdilib*), 324
- `wGetSessionId()` (in module *matlabwasdilib*), 325
- `wGetUploadActive()` (in module *matlabwasdilib*), 325
- `wGetUser()` (in module *matlabwasdilib*), 325
- `wGetVerbose()` (in module *matlabwasdilib*), 325
- `wGetWorkflows()` (in module *matlabwasdilib*), 326
- `wGetWorkspaceBaseUrl()` (in module *matlabwasdilib*), 326
- `wGetWorkspaceIdByName()` (in module *matlabwasdilib*), 326
- `wGetWorkspaceOwnerByName()` (in module *matlabwasdilib*), 326
- `wGetWorkspaces()` (in module *matlabwasdilib*), 327
- `wGetWorkspaceUrlByWsId()` (in module *matlabwasdilib*), 327
- `wImportAndPreprocess()` (in module *matlabwasdilib*), 327
- `wImportProduct()` (in module *matlabwasdilib*), 327
- `wImportProductList()` (in module *matlabwasdilib*), 328
- `wLog()` (in module *matlabwasdilib*), 328
- `wMosaic()` (in module *matlabwasdilib*), 328
- `wMultiSubset()` (in module *matlabwasdilib*), 329
- `wOpenWorkspace()` (in module *matlabwasdilib*), 329
- `wOpenWorkspaceById()` (in module *matlabwasdilib*), 329
- `wPrintStatus()` (in module *matlabwasdilib*), 330
- `wRefreshParameters()` (in module *matlabwasdilib*), 330
- `wSearchEOImages()` (in module *matlabwasdilib*), 330
- `wSetActiveWorkspaceId()` (in module *matlabwasdilib*), 331
- `wSetBasePath()` (in module *matlabwasdilib*), 331
- `wSetBaseUrl()` (in module *matlabwasdilib*), 331
- `wSetDownloadActive()` (in module *matlabwasdilib*), 331
- `wSetIsOnServer()` (in module *matlabwasdilib*), 332
- `wSetMyProcId()` (in module *matlabwasdilib*), 332
- `wSetParameter()` (in module *matlabwasdilib*), 332
- `wSetPassword()` (in module *matlabwasdilib*), 332
- `wSetPayload()` (in module *matlabwasdilib*), 333
- `wSetProcessPayload()` (in module *matlabwasdilib*), 333
- `wSetSessionId()` (in module *matlabwasdilib*), 333
- `wSetSubPid()` (in module *matlabwasdilib*), 333
- `wSetUploadActive()` (in module *matlabwasdilib*), 334
- `wSetUser()` (in module *matlabwasdilib*), 334
- `wSetVerbose()` (in module *matlabwasdilib*), 334

`wSetWorkspaceBaseUrl()` (*in module matlabwasdilib*),
334
`wSubset()` (*in module matlabwasdilib*), 334
`wUpdateProcessStatus()` (*in module matlabwasdilib*),
335
`wUpdateProgress()` (*in module matlabwasdilib*), 335
`wUpdateProgressPerc()` (*in module matlabwasdilib*),
336
`wUpdateStatus()` (*in module matlabwasdilib*), 336
`wUrlEncode()` (*in module matlabwasdilib*), 336
`wWaitProcess()` (*in module matlabwasdilib*), 336