Publish and Share

Developing using the Publish and Share Public API

Version 1.1
2020-17-01

Email: contact@its4land.com
Web: https://its4land.com/

Table of Contents

Introduction

The Publish and Share Public API is a HTTP based REST-ful API. What does this mean?

- It uses a client-server architecture
- Requests are *stateless*. Each request made by the client contains all information that is required to act upon it by the server, without using any of the stored context on the server. Session state is held in the client.
- Adheres to other REST principles of Caching, Uniform interface, Layered system and code on demand (https://www.ics.uci.edu/~fielding/pubs/dissertation/rest_arch_style.htm)
- HTTP methods GET/PUT/POST/DELETE/PATCH are used to perform the transition of state by the client.
- Resources on the server have a Uniform Resource Identifier (URI). A web URL is an example of a URI.
- An *endpoint* is a location identified by a URL and can service a request. The API accesses resources that correspond to an endpoint and performs a requested function on the resource.
- Upon a HTTP *request* to an endpoint by a client, the server acts upon the resource and :
    - sends metadata such as status code, in the HTTP *response header*
    - if expected, the actual content in a chosen representation format in the *response body*

All requests and responses in the Publish and Share API use the JSON format for data representation and exchange.

The intended audience for this guide are software developers aiming to develop applications or tools using Publish and Share APIs or to integrate it into their existing systems. Others who will benefit from reading this manual include system architects and developers who wish to extend the Publish and Share platform and customize it as per their requirements. Besides this document, reading the Publish and Share concepts manual is highly recommended to gain a broader understanding of the platform and project goals.

Reading the Guide

This is meant to be a practical hands-on guide to work with the platform. The following visual aids are used in the document:

```
This is a code block. It denotes commands to try out or the output observed
upon running a command.
```

This is a note. It denotes special points to pay attention to or behaviour to expect when using a platform.

This is a tip. It is intended to offer helpful pointers to the reader.

Getting Started

We first start by making simple API calls via HTTP requests. These requests can be made in several ways. We can do this :

- using popular applications such as a web browser, which offer limited means of interacting with the APIs
- third party command line tools such as cURL (https://curl.haxx.se/) or GUI tools such as Postman (https://www.getpostman.com/). These are more powerful and specifically tailored for this purpose
- programatically using libraries offered by many popular programming languages

For the purposes of this tutorial, we will be using the second option since they allow us to interactively make requests and observe responses. It is expected that the user has installed cURL and is able to run it from the command line. Other tools such as Postman provide a convenient GUI to make requests and observe responses and can likewise be used.

The API interactive documentation is viewable here - https://platform.its4land.com/api/

Let's start by listing all the projects on Publish and Share. For this we make a simple GET request to the API's `/projects` endpoint using cURL.

```
curl -X GET  https://platform.its4land.com/api/projects
```

If the request is successful, you should see an output similar to:

```
{
  "type": "FeatureCollection",
  "name": "i4lProject",
  "crs": {
    "type": "name",
    "properties": { "name": "urn:ogc:def:crs:OGC:1.3:CRS84"}
  },
  "features": [
    {
      "type": "Feature",
      "properties": {
        "Description": "Its4land project study area in Ensoka, Kenya",
        "Name": "Ensoka",
        "UID": "49d89c8b-5d44-4e9b-831a-d8276aa26950",
        "Models": [],
        "SpatialSources": [],
        "Tags": []
      },
      "geometry": {
        "type": "Polygon",
        "coordinates": [[
            [36.74229572626, -2.35793262136],
            [36.79906250924, -2.346579264764],
            [36.803245324828,-2.3776516091319],
            [36.796672328904,-2.4027485026598],
            [36.747076086932,-2.3854196952239],
            [36.735125185252,-2.3704810681239],
            [36.74229572626,-2.35793262136]
          ]]
      }
    }
  ]
}
```


The above response is in the JSON format, more specifically GeoJSON, which is meant to represent spatial data. GeoJSON can easily be created and viewed using third party tools such as QGIS (https://www.qgis.org/).

To observe the response headers, try passing the `-i` switch to cURL

```
curl -i -X GET https://platform.its4land.com/api/projects
```

In addition to the response body shown previously, cURL also prints the headers on stdout.

```
HTTP/1.1 200 OK
X-Powered-By: Express
Content-Type: application/json
Content-Length: 3075
Date: Wed, 10 Jul 2019 15:22:00 GMT
Connection: keep-alive
```

The status code 200 signifies that the request was successfully processed. Other status codes signify other states. Check the API documentation for details on status codes and their relevance for a given endpoint.

It is possible to filter the list of results using parameters. To return only projects which have the name 'MyProject', try the following call.

```
curl -i -X GET https://platform.its4land.com/api/projects?name=MyProject
```

Other parameters can be used to filter by uid, the area of interest, tags etc. For a complete list of valid parameters for an endpoint, refer to the API documentation.

A Simple its4land Workflow Using the API

In this section, we will go through a basic its4land work flow using the Publish and Share API. The workflow consists of the following steps:

1. Create a Project
2. Add a content item i.e. a file containing spatial data, to the platform
3. Create a Spatial Source under the project which refers to the content item added previously
4. Add an Additional Document attached to the spatial source just created
5. Run the UAV Ortho Generator tool to generate an orthophoto and register it in the platform

Create a Project

We have previously used the API to get a list of projects, now we will learn how to create one. A project in Publish and Share is represented as a GeoJSON **polygon feature**. The feature's `properties` field is used to specify the project name and other metadata associated with the project.

Creating a Project is done via the HTTP POST method to the */projects* endpoint. The body of the request should be a valid GeoJSON containing a feature with polygon geometry.

```
curl -X POST http://i4ldev1/sub/projects -H 'Content-Type: application/json' -d
'{
    "type":"FeatureCollection",
    "name":"i4lProject",

"crs":{"type":"name","properties":{"name":"urn:ogc:def:crs:OGC:1.3:CRS84"}},
    "features":[{
        "type":"Feature",
        "properties":{
            "Name":"MyProject",
            "Description":"My first Publish and Share project"
            },
        "geometry":{
            "type":"Polygon",
            "coordinates":[[[37.05439567565918,-
2.034532098209328],[37.05418109893799,-2.0540032643721338],[37.07954406738281,-
2.0533170598941815],[37.07945823669433,-2.040107566381197],[37.07551002502441,-
2.034146103545014],[37.06263542175293,-2.0332454489691165],[37.05439567565918,-
2.034532098209328]]]
        }
    }]
}'
```

You can create a project request body easily in QGIS by exporting a polygon layer as GeoJSON and editing it to add required properties

If successful (HTTP response code should be `201 Created`), the following response body is obtained (the UID field will vary). We have created an empty project in the area of interest. In the next steps we will add resources to the project.


```
{
    "type": "FeatureCollection",
    "name": "i4lProject",
    "crs":
{"type":"name","properties":{"name":"urn:ogc:def:crs:OGC:1.3:CRS84"}},
    "features": [
        {
            "type": "Feature",
            "properties": {
                "Description": "My first Publish and Share project",
                "Name": "MyProject",
                "UID": "7e97e65f-30bf-46ba-8274-bde794f0f597",
                "Models": [],
                "SpatialSources": [],
                "Tags": []
            },
            "geometry": {
                "type": "Polygon",
                "coordinates":[[[37.0543956756592,-
2.03453209820933],[37.054181098938,-2.05400326437213],[37.0795440673828,-
2.05331705989418],[37.0794582366943,-2.0401075663812],[37.0755100250244,-
2.03414610354501],[37.0626354217529,-2.03324544896912],[37.0543956756592,-
2.03453209820933]]]
            }
        }
    ]
}
```


Adding Content Items

A content item is a term adapted from Content/Document Management Systems. A content item is a file containing binary, textual, image or spatial data. The Publish and Share platform does not differentiate between different file types and formats. When a content item is added, it is stored as an object with a unique id (UUID - https://en.wikipedia.org/wiki/Universally_unique_identifier) on an Amazon Web Services S3 backend store.

For the purposes of demonstration, we will be using adding a zip archive containing aerial imagery obtained from an UAV. You can download this file here (37 MB). To add this file as a

content item, we carry out a POST request to the `/contentitems` endpoint. The cURL syntax to make POST request to upload a file is as shown below.

```
curl -X POST https://platform.its4land.com/api/contentitems/ -H 'content-type:
multipart/form-data' -F 'Description=UAV Imagery' -F newcontent=@images.zip
```

Following a successful request (HTTP Status code - 201), you should obtain a response containing the UUID of the newly added content item.

```
{
  "ContentType": "application/octet-stream",
  "ContentID": "7f53cc40-e4d9-4510-96cf-a003b142b362"
}
```

Depending on the file size and network speeds, the POST request can take a substantial amount of time to complete. Please note that currently there is no support for resuming partially completed uploads in case of interruption. In case the type of uploaded content cannot be recognized, the content mime-type is set to a generic `application/octet-stream` to indicate binary data.

Content items are stored as objects and do not have any file extension to recognize the file type. It is up to the uploader to keep track of the original file metadata. Alternatively, basic file metadata can be obtained by the following GET request.

```
curl -i -X GET "https://platform.its4land.com/api/contentitems?uid=7f53cc40-
e4d9-4510-96cf-a003b142b362"
```

This returns.

```
[
  {
    "ContentID": "7f53cc40-e4d9-4510-96cf-a003b142b362",
    "ContentName": "images.zip",
    "ContentType": "application/octet-stream",
    "ContentSize": 38995983
  }
]
```

In general, whenever we deal with data originating from files, in Publish and Share the recommended flow is to use the `/contentitems` endpoint to upload the file and use it's resulting content item object's UUID to associate it with other supported resources. The platform treats content items as raw data without any associated semantics. To make

content items more meaningful, they must be identified as a resource of a recognized type. The zip archive added above can be thought of as a source of spatial data. In the next section, we use the `/spatialsources` endpoint to denote it as such.

Adding a Spatial Source

A spatial source is any resource containing spatial information. This includes properly georeferenced data in formats such as shape files, ortho-mosaic imagery, geoJSON files. Even hand drawn data such as sketch maps or simply a textual document can be spatial sources.

We will now add the previously uploaded content item as a spatial source to our project via the POST `/projects/{project_uid}/SpatialSources` endpoint .

```
curl -X POST https://platform.its4land.com/api/projects/7e97e65f-30bf-46ba-
8274-bde794f0f597/SpatialSources -H 'Content-Type: application/json' -d '{
    "Name": "Demo UAV Data",
    "Description": "RAW Georeferenced aerial imagery",
    "Type": "UAVimagery",
    "ContentItem": "7f53cc40-e4d9-4510-96cf-a003b142b362"
}'
```

A successful request will identify the content item as a spatial source and add it to the project context. The UUID of the newly created spatial source is now part of the project body.


```
{
  "type": "FeatureCollection",
  "name": "i4lProject",
  "crs": {"type":"name","properties":{"name":"urn:ogc:def:crs:OGC:1.3:CRS84"}},
  "features": [
    {
      "type": "Feature",
      "properties": {
        "Description": "My first Publish and Share project",
        "Name": "MyProject",
        "UID": "7e97e65f-30bf-46ba-8274-bde794f0f597",
        "Models": [],
        "SpatialSources": [{ "UID": "33a343e7-beea-4cd5-8f26-1a902c07b7c6" }],
        "Tags": []
      },
      "geometry": {
        "type": "Polygon",
        "coordinates": [[[37.0543956756592,-
2.03453209820933],[37.054181098938,-2.05400326437213],[37.0795440673828,-
2.05331705989418],[37.0794582366943,-2.0401075663812],[37.0755100250244,-
2.03414610354501],[37.0626354217529,-2.03324544896912],[37.0543956756592,-
2.03453209820933]]]
      }
    }
  ]
}
```

Next, we will attach some documents to the newly added spatial source.

Add Additional Documents to a Spatial Source

A spatial source can have a number of supporting documents. Some of these, for instance, can provide additional metadata about the spatial source or differing versions of the source. The `/spatialsource/{spatialsource_uid}/AdditionalDocument` endpoint is used in such cases to add information about the resource.

To add an additional document, we first upload the document to be added using the `/contentitems` endpoint and then add the newly created contentitem as an additional document to the spatial source. We will be uploading a file `metadata.json` with the following content as a content item

```
{
    "Flight Misson": "Ruhengeri",
    "Date of flight": ["07.02.2019"],
    "UAV": "DJI Insipre 2",
    "Camera": "DJI FC652",
    "Flights carried out by": ["Charis UAS Ldt"],
    "Number of images": "3",
    "Flight height (m)": "120",
    "GSD (cm)": "2.3",
    "Area captured (km2)": "",
    "Context": "Urban",
    "Sideward Overlap": "70",
    "Forward Overlap": "80",
    "Flight mode": "stand alone",
    "Georeferencing Mode": "No GCPs",
    "Number of GCPs": "0"
}
```

Code Block 1 metadata.json

```
curl -X POST https://platform.its4land.com/api/contentitems/ -H 'content-type:
multipart/form-data' -F 'Description=UAV Imagery Metadata' -F
newcontent=@metadata.json
```

```
{
  "ContentType": "text/plain",
  "ContentID": "b78e0b54-518a-49a8-83bb-5cb3c15c52f8"
}
```

Next, we add the additional document

```
curl -X POST https://platform.its4land.com/api/spatialsource/33a343e7-beea-
4cd5-8f26-1a902c07b7c6/AdditionalDocument -d '{
    "Description":"Flight Metadata",
    "Type":"Metadata",
    "ContentItem":"b78e0b54-518a-49a8-83bb-5cb3c15c52f8"
}'
```

Running Tools

The Publish and Share platform API features allow for managing data and resources for use in Land Administration Systems. The APIs are designed to be flexible enough to allow easy integration into an organizations processes. There can however be a number of intermediate data processing steps to achieve the desired result. Tools are the mechanism by which this can be achieved. Tools can be thought of as being similar to browser plugins which perform additional related tasks and add value to the core application.

A tool can perform different tasks. An *entrypoint* of a tool performs the desired task can optionally take in input parameters. Tools are packaged as containers and run for as long as the container lasts. This provides benefits of isolation, security and scalability. Tools are essentially designed to run in the batch mode i.e. take input, run the application and terminate. It is also possible, under certain circumstances, to run tools interactively. A tool, for instance, can offer a web-based interface to allow user to make better use of it.

Listing Tools and their Entrypoints

The UAV Ortho Generator tool is one such tool which is part of the Publish and Share platform. The tool allows one to create an orthomosaic from an archive of UAV images and register it as a layer on an existing Geoserver (http://geoserver.org/) installation. Registration on a Geoserver allows one to server the layer using OGC services such as WMS. To actually view the results, one must have access to the Geoserver. Alternatively, the results from running the tool can be observed in the platform's demo GUI client.

The list of tools and their associated entrypoints can be obtained by the GET `/tools` endpoint. This provided information about all the tools registered in the platform. One can also filter the tool list by name.To get information about the UAV Ortho Generator tool, we can use the following request:

```
curl -L -X GET
'https://platform.its4land.com/api/tools?name=UAV%20Ortho%20Generator'
```

Running the above returns the list of tools and the entrypoints for the **latest version** of the UAV Ortho Generator tool. Since tools are intended to be produced by external developers,

the documentation can be available at a separate location. The documentation should cover the different entrypoints of the tool and what parameters they accept.


```
[
  {
    "Description": "Tool for creating orthomosaics from photos captured by
Unmanned Aerial Vehicles (UAVs)",
    "LongDescription": "The UAV Ortho Generator is based on OpenDroneMap (<a
href=\"https://www.opendronemap.org\"
target=\"_blank\">www.opendronemap.org</a>), a state-of-the-art open-source
image processing application. The tool uses modern photogrammetry and
structure-from-motion to generate 3D point clouds, digital surface models and
orthomosaics of UAV imagery.",
    "Name": "UAV Ortho Generator",
    "Supplier": "ITC",
    "UID": "34d4f329-59bf-4f31-ad72-48c4987944c0",
    "Version": "0.0.1",
    "Image": {
      "UID": "facf6b49-7b8a-41c6-a216-d55db086ad98",
      "ReleaseDate": "2019-11-19T18:20:13.000Z",
      "Image": "4e9da2a8f759"
    },
    "EntryPoints": [
      {
        "UID": "0662f078-9bb7-4896-a88e-02104b8ae79f",
        "Name": "Orthophoto",
        "Description": "Process Orthophotos"
      }
    ]
  }
]
```


Starting a Process using a Tool

We use the term *process* to denote a task that is run when using a tool. A process is always associated with a given project. In To API request to start a process contains the Project ID and tool information - name, version, entrypoint and input parameters.

The command below runs the UAV Ortho Generator tool version 0.0.1. The entrypoint **AddLayer** accepts two parameters - the coverage (layer) name to register in GeoServer and the actual content item id containing the spatial imagery data that is to be registered. The entrypoint registers the layer in GeoServer and additionally adds it to the map layer of the demo client.

```
curl -X POST 'https://platform.its4land.com/api/processes' -H 'Content-Type:
application/json' -d '{"Project":{"UID":"7e97e65f-30bf-46ba-8274-
bde794f0f597"}, "Tool":{"ToolName":"UAV Ortho
Generator","Version":"0.0.1","EntryPoint":{"EntryPointName":"Orthophoto","Param
eter":[{"ParameterName":"--spatial-source-id","ParameterValue":"33a343e7-beea-
4cd5-8f26-1a902c07b7c6"},{"ParameterName":"--texturing-nadir-
weight","ParameterValue":"urban"},{"ParameterName":"--opensfm-depthmap-
method","ParameterValue":"BRUTE_FORCE"},{"ParameterName":"--min-num-
features","ParameterValue":"10000"}]}}}'
```

The response should contain the newly created Process' UID, metadata containing timestamps as well as initial logs

```
{
  "CreatedAt": "2020-01-15T13:50:35.926Z",
  "CreatedBy": "dummy",
  "LastModifiedAt": "2020-01-15T13:50:35.926Z",
  "LastModifiedBy": "dummy",
  "Project": {
    "UID": "7e97e65f-30bf-46ba-8274-bde794f0f597",
    "Name": "_Mapping Parcels in Ruhengeri_Clip1, Rwanda",
    "Description": "The project demonstrates updating land tenure records for
Residential and agricultural parcels from orthophotos image using the Boundary
Delineator and Participatory Mapping"
  },
  "Status": "RUNNING",
  "UID": "cc455c2c-d53d-469e-9841-848cef49c5c1",
  "Tool": {
    "ToolName": "UAV Ortho Generator",
    "Version": "0.0.1",
    "EntryPoint": {
      "EntryPointName": "Orthophoto",
      "Parameter": [
        {
          "ParameterName": "--spatial-source-id",
          "ParameterValue": "33a343e7-beea-4cd5-8f26-1a902c07b7c6"
        },
        {
          "ParameterName": "--texturing-nadir-weight",
          "ParameterValue": "urban"
        },
        {
          "ParameterName": "--opensfm-depthmap-method",
          "ParameterValue": "BRUTE_FORCE"
        },
        {
          "ParameterName": "--min-num-features",
          "ParameterValue": "10000"
        }
      ]
    }
  },
  "Logs": [
    {
      "SeqNr": 4,
      "LogDate": "2020-01-15T13:50:37.003Z",
      "LogSource": "PublishAndShare",
      "LogLevel": "info",
      "LogMsg": "Process status changed to RUNNING"
    },
    {
      "SeqNr": 3,
      "LogDate": "2020-01-15T13:50:36.216Z",
      "LogSource": "PublishAndShare",
      "LogLevel": "info",
      "LogMsg": "Process status changed to WAITING"
    },
    {
      "SeqNr": 2,
      "LogDate": "2020-01-15T13:50:36.176Z",
      "LogSource": "PublishAndShare",
      "LogLevel": "info",
      "LogMsg": "Process status changed to CREATED"
    },
    {
```

```
        "SeqNr": 1,
        "LogDate": "2020-01-15T13:50:35.934Z",
        "LogSource": "PublishAndShare",
        "LogLevel": "info",
        "LogMsg": "Starting process"
      }
    ],
    "Results": []
  }
```


At present there is no service to automatically notify the process status when finished. Checking whether a process has finished running is done manually via an API endpoint request to `/processes/{uid}` and examining the status field.

A status of `FINISHED` indicates the tool container terminated normally. The status `ABORTED` indicates abnormal termination. It must be noted that neither termination statuses indicate that the tool's functionality was successfully completed. It is an indicator of the status of the Docker container within whose environment the tool executed. For example, the UAV Ortho Generator tool may fail to create an orthomosaic, but the container which runs the tool might still terminate cleanly with a status of `FINISHED`.

Error messages dealing with the tool's functionality need to make use of the POST `/processes/{uid}/logs` endpoint.

In this guide, we have covered a rudimentary its4land workflow and demonstrated how to use the API in practice. Using cURL enables us to craft a request and observe the response immediately. When using the API to develop applications or integrate it in existing workflows, the practice will be to use programming languages and libraries to make HTTP requests and hand JSON data. For instance, when using Python, the 'requests' library is one of the many options that can be used. It is impractical to cover even popularly used programming language due to the sheer number of libraries available which enable the API to be used. Our goal here was to demonstrate the features of the API using a tool which provided immediate feedback. Adapting this to a programming language is left as an exercise to the reader.