Digital Content Analysis Technology Ltd

Fusion Platform®

Python SDK

Version 1.10

16th January 2024

Table of Contents

Revision History

Draft	7th February 2022	Initial draft for review.
Draft	9th February 2022	Updated following internal review.
Draft	22nd February 2022	Updated during implementation.
Draft	3rd March 2022	Version candidate.
Draft	8th March 2022	Revised following internal review to include link to code documentation.
Draft	20th April 2022	Added find_organisations.
Version 1.0	29th April 2022	Issued for Fusion Platform® version 1.0.
Draft	6th May 2022	Revised examples to use published demonstration examples.
Version 1.1	12th May 2022	Issued for Fusion Platform® version 1.1.
Draft	14th June 2022	Revised to include downloading of file previews.
Version 1.2	15th June 2022	Issued for Fusion Platform® version 1.2.
Draft	27th June 2022	Revised to include search terms in find methods.
Draft	21st July 2022	Added aggregator to detailed example.
Version 1.3	22nd July 2022	Issued for Fusion Platform® version 1.3.
Draft	1st September 2022	Added data item copy.
Draft	1st September 2022	Added description of how to find out what the inputs and options are for a process.
Version 1.4	1st September 2022	Issued for Fusion Platform® version 1.4.
Draft	24th November 2022	Added ability to copy a process.
Version 1.5	29th November 2022	Issued for Fusion Platform® version 1.5.
Version 1.6	8th February 2023	Issued for Fusion Platform® version 1.6.
Draft	3rd May 2023	Added changed_delete_expiry method for process execution.
Draft	23rd May 2023	Added the ability to include dispatchers within a process.
Version 1.7	7th June 2023	Issued for Fusion Platform® version 1.7.
Draft	26th June 2023	Extracted and tidied examples.
Draft	7th July 2023	Added AWS S3 dispatcher example.
Version 1.8	8th August 2023	Issued for Fusion Platform® version 1.8.
Version 1.9	4th January 2024	Issued for Fusion Platform® version 1.9.
Draft	11th January 2024	Added email dispatcher.
Draft	12th January 2024	Added KMS key policy example for AWS S3 dispatcher and condition option to all dispatchers.
Version 1.10	16th January 2024	Issued for Fusion Platform® version 1.10.

1 Introduction

1.1 Purpose

This document describes the Python SDK used to interact with the Fusion Platform®. The Fusion Platform® provides enhanced remote monitoring services. By ingesting remotely sensed Earth Observation (EO) data, and data from other sources, the platform uses and fuses this data to execute algorithms which provide actionable knowledge to customers.

1.2 Scope

The Python SDK is designed to enable interaction with the Fusion Platform® via its API. As such, the SDK therefore allows software to login, upload files, create and execute processes, monitor their execution and then download the corresponding results. Additional functionality is available directly via the API, and this is defined within the corresponding OpenAPI 3.0 specification, which can be obtained via a support request.

Support for the SDK can be obtained by contacting Digital Content Analysis Technology Ltd via support@d-cat.co.uk.

1.3 Overview

This document first presents a brief overview of the Fusion Platform® and the general requirements for the SDK. This is followed by an example of how to get started with the SDK for those who wish to try things out. Subsequent sections describe the functionality in detail. Finally, a more detailed example is provided, along with a full list of the SDK methods available, together with a glossary of terms.

Python code snippets within this document are shown as follows. These assume a basic level of Python knowledge. Each block is a snippet of code which may rely upon previous code examples to be completed to successfully execute.

Python

```
# This is a Python code snippet.
print('Hello world!')
```

Where appropriate, shell commands are also shown. These should work regardless of the host operating system (for example, Windows PowerShell, Mac Terminal or Linux command line):

Command

```
echo "Hello world!"
```

When there are important notes or warnings, they are shown like this:

Important

This is an important note!

2 Overview

The Fusion Platform® is a cloud-based service which allows users to execute algorithms published by algorithm providers. Execution may be on-demand or at regular intervals using a variety of available EO and other data inputs. This data is automatically ingested by the platform to satisfy execution requirements. Users pay for execution and storage of files through pre-purchased credits.

2.1 User Accounts, Organisations and Credits

All access to the Fusion Platform® is via a registered user account. Account registration is performed separately to the SDK and must therefore be completed before the SDK can be used.

All user accounts are linked to an email address which can receive notifications. A mobile telephone number may also be provided with an account so that notifications may be sent by text message as well. Accounts are accessed using an email address or user identifier and a password.

Each user account is linked to one or more organisations. Each organisation may be associated with one or more user accounts. Each user account is limited by privileges within an organisation. Privileges are allocated to a user account by the organisation manager to allow the account to view processes and their outputs, and to execute processes. Organisations are used to separately hold processes and their data which have been executed on behalf of the organisation by account holders. Files and processes created for an organisation are not available to any other organisation.

Each organisation is associated with pre-purchased credits, which are used to pay for process executions and for the storage of files. Credits are purchased separately to the SDK. If an organisation does not have sufficient credits to execute a process, then execution will be prevented. All uploaded files and outputs from process executions incur storage costs, which are accounted for daily. If the organisation has insufficient credits to store files, then upload will be prevented. If an organisation's credits run out and it has stored files, these may be deleted to prevent further costs from being incurred without payment.

2.2 Workflow

A typical workflow for the Fusion Platform® using the SDK is as follows:

1. Login to the Fusion Platform® and select an organisation to work with

All access to the Fusion Platform® is via a registered user account. Once logged in, an organisation must be selected for which processes are to be executed. The list of organisations available to a user account can be obtained from the user information.

2. Upload a geospatial vector file which describes a region of interest

The Fusion Platform® predominately works with geospatial data, and consequently the algorithms available typically require a region of interest to be defined to produce outputs which correspond to this region. Regions of interest are defined using geospatial vector files which describe one or more polygons. The first step in executing a process is therefore to upload a vector file defining the region of interest. For example, this could be a GeoJSON file, or an ESRI Shapefile. Vector files can be created using tools such as QGIS [1], or geojson.io [2].

3. Select the service that is to be executed

The Fusion Platform® provides a variety of services which can be selected for execution. This step therefore involves obtaining the list of services available to find the one which is needed.

4. For the selected service, configure a process to run for the region of interest

Once selected, a process can be created from the service. The inputs and options for the process must be defined so that the process can be executed. Once created, and prior to execution, the process is associated with a price giving the number of credits needed to execute it. This price will be charged per execution of the process.

5. Execute the configured process

Once the process has been correctly configured to obtain a price per execution, the process can be executed. Execution can be once or scheduled to run at one or more times in the future. Each subsequent execution will incur the associated price being charged.

6. Monitor the execution of the process

Once scheduled for execution (once or many times), each execution may be monitored to determine its progress and to find out when it is finished.

For large regions of interest, the Fusion Platform® will automatically cut up the region into smaller slices to make processing more efficient. Each slice will be executed in parallel so that the outputs are obtained more quickly. When a region is cut up, multiple executions will be created automatically for a process (each with a proportion of the overall price), and all of them placed into a single group. Monitoring may therefore take place of single or grouped executions.

When the execution (or group of executions) has run to completion, an optional notification will be sent via email or text message to the user account which executed the process to notify that the execution or group of executions has finished.

It is also possible to stop all executions of a process. This will stop all ongoing and future executions. Any partial executions will be subject to the associated portion of the charges incurred.

7. Download the outputs from the executed process

When an execution has completed successfully, the outputs can be downloaded. All the inputs to and outputs from an execution are available for download. It also possible to configure a process to automatically dispatch outputs to, for example, an AWS S3 bucket or email, so that the download is unnecessary.

8. Delete completed process

When a process and its outputs are no longer needed, the process can be deleted. Executions will be automatically deleted after a delay configured when the process is created. Deleting a process will delete all its executions and their outputs (and therefore stop any further storage charges being incurred). Files which have been uploaded explicitly by the user are not automatically deleted, only executions and their generated data.

9. Delete the region of interest file

When a region of interest file is no longer required, and it is not used by any process, then it can be explicitly deleted by the user (and therefore stop any further storage charges being incurred for it).

2.3 Installation

The SDK has been built for Python 3, and is tested against Python 3.8, 3.9, 3.10 and 3.11. To install the SDK into a suitable Python environment containing "`pip`", execute the following:

Command

```
pip install fusion-platform-python-sdk
```

This will install the SDK and all its dependencies.

To update an existing installation to the latest version, execute the following:

Command

```
pip install fusion-platform-python-sdk --upgrade
```

2.4 Usage

Within a Python file, the SDK can then be used by importing it. The following shows how the version of the SDK can be displayed.

Python

```
import fusion_platform

# Print the current version of the Fusion Platform(r) SDK.
print(fusion_platform.__version__)
print(fusion_platform.__version_date__)
```

3 Quick Example

The following shows a simple example of how to use the SDK to create a process, execute it and download the resulting data. This example assumes that a suitable user account has been created with an organisation which has sufficient credits to run the process, and that the SDK has been set up as described in section 2.3. To use a different user account, replace the email address ‘`me@org.com`’ and password ‘`MyPassword123!`’.

The example uses a pre-defined region of interest for the Glasgow in the UK, and then obtains elevation data for the region. It should take approximately 5 minutes to run.

Python

```
import os
import fusion_platform


# Login with email address and password.
user = fusion_platform.login(email='me@org.com',
                             password='MyPassword123!')


# Select the organisation which will own the file. This is
# the first organisation the user belongs to in this example,
# and which is therefore assumed to have sufficient credits.
organisation = next(user.organisations)


# Create a data item for the Glasgow region of interest:
glasgow = organisation.create_data(
              name='Glasgow',
              file_type=fusion_platform.FILE_TYPE_GEOJSON,
              files=[fusion_platform.EXAMPLE_GLASGOW_FILE],
              wait=True)


# Find the elevation service.
service, _ = organisation.find_services(keyword='Elevation')


# Create a template process from the service.
process = organisation.new_process(name='Example',
                                   service=service)


# Configure the process to use Glasgow as the region.
process.update(input_number=1, data=glasgow)

# Create the process, which will validate its options and inputs.
process.create()
```

Python

```
# Before execution, review the price in credits.
print(f"Price: {process.price}")

# Now execute the process and wait for it to complete.
process.execute(wait=True)

# Get the corresponding execution of the process. This is assumed
# to be the most recently started execution.
execution = next(process.executions)

# Now download all the outputs.
for i, component in enumerate(execution.components):
    print(f"Downloading {component.name}")
    component_dir = f"component_{str(i)}"

    for j, output in enumerate(component.outputs):
        dir = os.path.join(component_dir, str(j))
        for file in output.files:
            file.download(path=os.path.join(dir, file.file_name))

# Now tidy everything up by deleting the process and the region.
process.delete()
glasgow.delete()
```

4 Authentication

To login to the Fusion Platform® using the SDK, the user account credentials must be obtained first. These can be obtained, for example, from Digital Content Analysis Technology Ltd via support@d-cat.co.uk. The user account credentials require an email address for notifications. An optional user identifier may be supplied along with an automatically generated password. This password should be changed after first login. Passwords must be at least 8 characters long and contain at least an uppercase and lowercase letter, a number and a special character from '`^$*.[]{}()?"!@#%&/\,><':;|_~`=+-`'.

To login with the SDK, use the following:

Python

```python
import fusion_platform


# Login with email address and password.
user = fusion_platform.login(email='me@org.com',
                              password='MyPassword123!')
```

If you have a user identifier, you can also use:

Python

```python
# Login with user identifier and password.
user = fusion_platform.login(
              id='ff1fbd91-bd64-4ad7-ac80-d0b3f22f9d19',
              password='MyPassword123!')
```

Here, the registered email address is '`me@org.com`', user identifier is '`ff1fbd91-bd64-4ad7-ac80-d0b3f22f9d19`' and password is '`MyPassword123!`'.

If login is successful, a user object will be returned. If unsuccessful, an exception will be raised giving details of the error encountered.

The created user object provides all access to the Fusion Platform® through an authenticated session with an appropriate configuration. All sessions last for a maximum of 1 day. After this, login will be required again to obtain a new session.

Important

When using multiple threads, separate user objects should be created within each thread to ensure that no session data is changed asynchronously across threads.

When logging in, the following configuration parameters may be set in addition to the user account credentials:

`api_url`	The URL of the Fusion Platform®. This will default to `https://api.thefusionplatform.com`, but may be modified as required for specific uses.

For example:

```Python
# Login with email address, password and custom API URL.
user = fusion_platform.login(
           email='me@org.com',
           password='MyPassword123!',
           api_url='https://custom.api.thefusionplatform.com')
```

The user object represents all information about the user, as well as the active Fusion Platform® session. It can be used to modify key user details, including the user’s given name, family name, telephone number and notification preferences. For example:

```Python
# Modify the user's name:
user.update(given_name='Joe', family_name='Bloggs')
```

If the update fails for any reason, such as invalid parameter values, then an exception will be thrown. Section 11 describes how to get help on the content of the user information.

Changing the password uses a separate method since the old password must be provided:

```Python
# Modify the password.
user.change_password(old='MyOldPassword123!',
                     new='MyNewPassword123!')
```

Note that the password must comply with the password policy enforced at the time. Currently, this requires a minimum length of 8 characters with at least one number, one lowercase letter, one uppercase letter and one of the special characters “`.[]{}()?-!@#%&,><:;|_~`”.

The user object can also be used to provide an iterator through the organisations to which the user account belongs:

```Python
# Get the organisation information for all the user's
# organisations.
for organisation in user.organisations:
    print(organisation.name)
```

Organisation objects can also be searched for by name:

```Python
# Search for organisations by name:
organisation, organisations = user.find_organisations(
                                  name='My Organisation')
```

Searching by name looks for those organisations which have the corresponding name. Names are not unique, hence the ‘`organisation`’ return is the first item that has been found (or ‘`None`’ if no items found) and the ‘`organisations`’ return is an iterator through all items found. A case-insensitive search is performed for organisations whose name starts with the supplied value.

A specific organisation object can also be obtained from its organisation identifier, which is listed as part of its information.

Python

```
# Get the organisation by its identifier:
organisation, _ = user.find_organisations(
                   id='981352c2-a5b8-4650-acdc-9ab886a63163')
```

An organisation object is used to access files and processes owned by the organisation. It can also be used to modify key organisation details if the user account has the appropriate privilege. For example:

Python

```
# Modify the organisation's name:
organisation.update(name='My Organisation')
```

If the update fails for any reason, such as invalid parameter values or insufficient privileges, then an exception will be thrown. Section 11 describes how to get help on the content of the organisation information.

5 Uploading Files

Most algorithms available in the Fusion Platform® analyse and output data based upon a given region of interest. Regions of interest are defined using geospatial vector files which describe one or more polygons, which can be created using tools such as QGIS [1], or geojson.io [2]. Supported geospatial vector file formats include GeoJSON, ESRI Shapefile, KML and KMZ. All regions of interest are defined within a single file. For example, a GeoJSON file is a text file containing JSON which defines the region.

Important

An ESRI Shapefile is different in that the region is defined by a series of files with the same name, but different extensions. To be uploaded as a region of interest, all the files which go to make up the ESRI Shapefile must be zipped into a single archive file with no directory structure included. This single file can then be uploaded as an ESRI Shapefile.

Other algorithms may work with other types of input, such as geospatial raster or CSV files. The type of file required is detailed with the service information.

To be able to provide a region of interest file to an algorithm, the file must first be uploaded to the platform. A data object represents one of more files grouped into a single object, and data objects are used as input to, and output from, processes. For example, a region of interest defined in a GeoJSON file '`region.geojson`' found in the local directory is uploaded to a data object as follows:

Python

```
# Select the organisation which will own the file. This is
# the first organisation the user belongs to in this example:
organisation = next(user.organisations)


# Now create a data object and upload the file to it.
# This will return as soon as the data object is created, and
# will not wait for the files to be uploaded or analysed.
data = organisation.create_data(name='My Region',
                 file_type=fusion_platform.FILE_TYPE_GEOJSON,
                 files=['region.geojson'])
```

This will create the associated data object for the organisation, give it a name and then upload the GeoJSON file to the Fusion Platform®. Multiple files of the same type can be uploaded by extending the list of paths in the array. If the create fails for any reason, such as invalid parameter values, insufficient privileges or not enough credits, then an exception will be thrown. The default usage of this method is not to wait for the upload and analysis of the files to complete, but instead to return once the data object has been created successfully.

Important

The amount of time it takes to upload files will depend upon the size of the files and the bandwidth of your network connection. Once uploaded, the Fusion Platform® will perform some analysis on the files before the files can be used. The default version of `create_data` will not wait for all these steps to complete, but these steps must be complete before the data object can be used within an execution.

To find out if the upload and analysis have completed, use the following:

Python

```
import time

# Loop waiting for the create to complete.
while not data.create_complete():
    time.sleep(10)  # Sleep for 10 seconds.

# The create is complete without exception.
print('Create successfully completed')
```

If the upload and analysis have not yet completed, then this method will return ‘`False`’. If it has completed successfully, this will return ‘`True`’. If, however, the upload and analysis have completed but failed, then this method will raise an exception containing the details of why it failed.

To have the method wait for all uploads and the analysis to complete, then the ‘`wait`’ flag can be turned on:

Python

```
# Now create a data item and upload the file to it.
# This will wait for the upload and analysis to complete.

data = organisation.create_data(name='My Region',
              file_type=fusion_platform.FILE_TYPE_GEOJSON,
              files=['region.geojson'],
              wait=True)
```

Important

If the upload and analysis of the data object’s files has not completed, then the data object will not be available for use within a process. The upload will take place in a series of background threads, and hence if the program is closed, the upload will terminate prematurely causing the uploaded files to be in an invalid state. The data object must then be deleted, and the upload started again.

A data object can also be created by copying another data object:

Python

```
# Copy a data object:
data_copy = data.copy(name='Copy of My Region')
```

This creates a copy of the object with the specified name. Note that the copy is treated as if it was uploaded (compared to data objects which are generated by process executions).

An iterator through the data objects which have been uploaded to the platform for the organisation can be obtained using:

Python
```python
# Get an iterator of the data items belonging to the
# organisation:
for data in organisation.data:
    print(data.name)
```

Data items can also be searched for by name:

Python
```python
# Search for data items by name:
data, data_items = organisation.find_data(name='My Region')
```

Searching by name looks for those services which have the corresponding name. Names are not unique, hence the '`data`' return is the first item that has been found (or '`None`' if no items found) and the '`data_items`' return is an iterator through all items found. A case-sensitive search is performed for data objects whose name starts with the supplied value. A case-insensitive search on relevant object fields can be conducted using the '`search`' parameter instead.

A specific data item can also be obtained from its data identifier, which is listed as part of its information.

Python
```python
# Get the data item by its identifier:
data, _ = organisation.find_data(
              id='e1fca0a4-e14d-4a9a-8c76-82a6a6831860')
```

A data object can be used to modify the name. For example:

Python
```python
# Modify the name of the first uploaded data item:
data = next(organisation.data)
data.update(name='Other Region')
```

If the update fails for any reason, such as invalid parameter values or insufficient privileges, then an exception will be thrown. Section 11 describes how to get help on the content of the data information.

A data object can be used to iterate through and download one or more of the associated files:

Python

```python
# Download all files from the data object to the local
# directory. Each download will return immediately once started.
for file in data.files:
    print(f"Downloading {file.file_name} ({file.size} bytes)")
    file.download(path=file.file_name)
```

Section 11 describes how to get help on the content of the file information.

The default usage of the ‘`download`’ method is not to wait for the for the download to complete, but instead to return immediately.

Important

The amount of time it takes to download a file will depend upon the size of the files and the bandwidth of your network connection. The download will take place in a background thread, and hence if the program is closed, the download will terminate prematurely causing the downloaded file to be in an invalid state.

To find out the progress of a download, and if the download has completed, use the following:

Python

```python
import time

# Loop waiting for the download to complete.
while not file.download_complete():
    url, destination, size = file.download_progress()
    print(f"Downloaded {size} of {file.size} bytes")

    time.sleep(10)  # Sleep for 10 seconds.

# The download is complete without exception.
print('Download successfully completed')
```

To have the method wait for the download to complete, turn the ‘`wait`’ flag on:

Python

```python
# Download all files from the data object to the local
# directory. Each download will only return when complete.
for file in data.files:
    print(f"Downloading {file.file_name} ({file.size} bytes)")
    file.download(path=file.file_name, wait=True)
```

File may also have a small preview available (as a PNG). To download the preview file:

Python

```python
# Download the preview files for all files in the data object.
# An error will be raised if a file does not have a preview.
# Here we also wait for the downloads to complete.
for i, file in enumerate(data.files):
    print(f"Downloading preview for {file.file_name}")
    file.download(path=f"{i}.png", preview=True, wait=True)
```

When the data item is not being used by any process, it may be deleted:

Python

```python
# Delete the data item:
data.delete()
```

This will raise an exception if the data item is in use by a process, or because of insufficient privileges, otherwise the item will be deleted. Deleting a data item ensures that no further storage costs are incurred for its files. Storage charges are incurred daily.

6 Browsing Services

The purpose of the Fusion Platform® is to provide a convenient way of efficiently executing a range of algorithms. Each algorithm is bundled into a service which defines what data is required for the algorithm to execute so that when the service is run, it will obtain all the required data, then run the algorithm to produce the desired outputs.

The following provides an iterator through the services that are available to an organisation:

Python

```python
# Select the organisation to browse services. This is
# the first organisation the user belongs to in this example:
organisation = next(user.organisations)


# Now iterate through the available services:
for service in organisation.services:
    print(service.name)
    print(service.documentation_summary)
```

A service object contains all the details of a service. This includes its name and documentation. Services can also be searched for by name or by keyword:

Python

```python
# Search for services by name. This finds all demo services.
service, services = organisation.find_services(name='Demo')


# Search for services by keyword. This finds the elevation service.
service, services = organisation.find_services(
                                             keyword='Elevation')
```

A case-sensitive search is performed for services whose name or keywords start with the supplied values. A case-insensitive search on relevant object fields can be conducted using the '`search`' parameter instead. Names are not unique, hence the '`service`' return is the first item that has been found (or '`None`' if no items found) and the '`services`' return is an iterator through all items found.

A specific service can also be obtained from its service identifier or SSD ID. Both are listed as part of the service information.

Python

```python
# Both these get the same elevation service:
service, _ = organisation.find_services(
              id='d87a62b1-80e4-435f-88d8-11dccbabf94f')
service, _ = organisation.find_services(
              ssd_id='c35e7d40-9b66-4db3-ba2d-9252d39c8d8e')
```

Section 11 describes how to get help on the content of the service information.

Important

All services in the Fusion Platform® are uniquely identified by an SSD ID. At any time, there will always be only one service available with the unique SSD ID. When a new version of a service is released, it will have the same SSD ID, but it will have a different version number and individual service identifier. The service identifier therefore uniquely identifies a specific version of a service, whereas an SSD ID is used to find the most recent version of a service, and should therefore be the preferred way of finding a service through an identifier.

7 Creating Processes

Once a service has been selected, a process can be created from it. To obtain a template process from a service, use the following:

Python

```
# Get a template process from a service object:
process = organisation.new_process(name='Example',
                                    service=service)
```

Here, to create the template process, a service object is required. If the request fails for any reason, such as invalid parameter values or insufficient privileges, then an exception will be thrown. See section 6 for a description of how to get a service object.

Important

The returned process object is a template which has not been configured for execution. Because it has not been configured, this template object has not been persisted to the Fusion Platform®. Only a successfully configured process object can be persisted.

The process object contains three important parts that must be completed before the process object can be saved to the Fusion Platform®. First, all mandatory options must be completed. Second, all input data items need to be specified. Third, the execution schedule needs to be defined.

The list of available options and inputs can be obtained from the process model using the following:

Python

```
# Find out what options are available. This will list all
# the options, their name, data type, whether they are required,
# current value and description.
for option in process.options:
    print(option)


# Find out what inputs are required. This will list all the
# inputs, their name, file type, current value and description.
for input in process.inputs:
    print(input)
```

Section 11 describes how to get help on the content of the process information.

Options can be completed as per the following example:

Python

```python
from datetime import datetime, timezone

# Set the mandatory options which need to be completed:
for option in process.options:
    if option.required and (value is None):
        # This option is mandatory. Set the value by its type.
        if option.data_type == fusion_platform.DATA_TYPE_NUMERIC:
            process.update(option=option, value=1)
        elif option.data_type == fusion_platform.DATA_TYPE_CURRENCY:
            process.update(option=option, value=1)
        elif option.data_type == fusion_platform.DATA_TYPE_BOOLEAN:
            process.update(option=option, value=True)
        elif option.data_type == fusion_platform.DATA_TYPE_DATETIME:
            process.update(option=option, value= datetime.now(timezone.utc))
        elif option.data_type == fusion_platform.DATA_TYPE_STRING:
            process.update(option=option, value='Hello world!')
        elif option.data_type == fusion_platform.DATA_TYPE_CONSTRAINED:
            process.update(option=option, value=option.constrained_values[0])
```

This example loops through all the options, finds the mandatory options which have not got a default value, and completes each with an example value of the correct data type. Alternatively, individual options can be set using their name:

Python

```python
from datetime import datetime, timedelta, timezone

# Get the list of options and their description.
for option in process.options:
    print(option.name)
    print(option.description)

# Set an option using its name.
process.update(option_name='latest_date', value=False)

start_date = datetime.now(timezone.utc) - timedelta(days=25)
process.update(option_name='start_date', value=start_date)
```

To provide inputs, there must be an existing data model which has been uploaded and completed its analysis, as shown in section 5.

Python

```
# Find the region of interest and a raster file, both of
# which have been uploaded previously.
region, _ = organisation.find_data(name='My Region')
raster, _ = organisation.find_data(name='My Raster')


# Specify the inputs to the process.
for input in process.inputs:
    if input.file_type == fusion_platform.FILE_TYPE_GEOJSON:
        process.update(input=input, data=region)
    elif input.file_type == fusion_platform.FILE_TYPE_GEOTIFF:
        process.update(input=input, data=raster)
```

Alternatively, inputs can be set using their input number, with the number for the first input to the process '1'.

Python

```
# Set the input by its number.
process.update(input_number=1, data=region)
```

Finally, the execution schedule can be defined. This is either to run the process once, or to run it using a particular schedule:

Python

```
from datetime import datetime, timedelta, timezone


# Run the process once...
process.update(run_type=fusion_platform.RUN_TYPE_RUN_ONCE)


# ...or using a schedule, which will run the job once, tomorrow.
then = datetime.now(timezone.utc) + timedelta(days=1)
process.update(run_type=fusion_platform.RUN_TYPE_RUN_SCHEDULE,
               repeat_count=1, repeat_start=then)
```

Full details on options, inputs and scheduling can be found in the process information. Section 11 describes how to get help on the content of the process information.

In addition to correctly configuring the process input, options and schedule, the process can also be configured to dispatch outputs using the available set of dispatchers. For example, a dispatcher can be used to automatically send the outputs of the process to an AWS S3 bucket, or via email (albeit limited by the size of the attachments).

The list of available dispatchers and their options can be obtained from the process model using the following:

Python

```python
# Find out what dispatchers are available. This will list
# all the dispatchers with their name and brief description, plus
# also print out their options.
for dispatcher in process.available_dispatchers:
    print(dispatcher)

    for option in dispatcher.options:
        print(option)
```

A dispatcher can be added to the process using the following:

Python

```python
# Add the first dispatcher from the list of those available.
process.add_dispatcher(number=1)

# A dispatcher can also be added using its name.
process.add_dispatcher(name='AWS S3')
```

Once a dispatcher has been added to the process, it can be configured and its options set using the following:

Python

```python
# Configure the options of the first dispatcher which has
# been added to the process. Note that an option object cannot be
# used to modify a dispatcher option.
process.update(dispatcher_number=1, option_name='bucket',
               value='my-bucket')
```

To view the list of dispatchers which have been added to a process, use the following:

Python

```python
# Find out what dispatchers have been added to a process.
# This will list all the dispatchers with their name and brief
# description, plus also print out their options.
for dispatcher in process.dispatchers:
    print(dispatcher)

    for option in dispatcher.options:
        print(option)
```

To remove a dispatcher, use the following:

Python

```python
# Remove the first dispatcher from the list of those added.
process.remove_dispatcher(number=1)
```

As an example, the AWS S3 dispatcher can be used to automatically write the outputs from a process to any destination S3 bucket. The dispatcher must be configured with the name of the destination bucket via the ‘`bucket`’ option. Where credentials are required to write to the bucket, then these may be set via the ‘`access_key_id`’ and ‘`secret_access_key`’ options, and the optional ‘`session_token`’.

Important

As an alternative to providing credentials to the AWS S3 dispatcher, a policy can be set on the destination bucket which will allow the dispatcher to automatically write to the bucket. This is:

```json
{
 "Version": "2012-10-17",
 "Statement": [
  {
   "Sid": "DelegateS3Access",
   "Effect": "Allow",
   "Principal": {
   "AWS": [
     "arn:aws:iam::542134873422:role/FargatePrivilegedTaskRole"
    ]
   },
   "Action": "s3:PutObject",
   "Resource": [
    "arn:aws:s3:::<NAME-OF_BUCKET>",
    "arn:aws:s3:::<NAME-OF_BUCKET>/*"
   ]
  }
 ]
}
```

Where ‘`<NAME-OF-BUCKET>`’ should be replaced with the name of the bucket. Note that if an alternative ‘`api_url`’ is being used during login, then a different AWS account number will be required in this bucket policy. For example, for the ‘`api_url`’ of ‘`https://api.thefusionplatform.com.au`’, the account number is ‘`324181581397`’.

When using the AWS S3 dispatcher with an explicit customer managed KMS key which is used to encrypt the outputs in the destination bucket, its ARN can be specified via the ‘`kms_key_id`’ option.

Important

When using a customer managed KMS key to encrypt objects in the destination bucket, then the key's policy must be modified to allow the Fusion Platform® to use it when encrypting the objects. An example of this policy is:

```
{
 "Version": "2012-10-17",
 "Statement": [
  {
   "Sid": "DelegateEncryption",
   "Effect": "Allow",
   "Principal": {
   "AWS": [
     "arn:aws:iam::542134873422:role/FargatePrivilegedTaskRole"
    ]
   },
   "Action": "kms:GenerateDataKey",
   "Resource": "*"
  }
 ]
}
```

Note that if an alternative '`api_url`' is being used during login, then a different AWS account number will be required in this bucket policy. For example, for the '`api_url`' of '`https://api.thefusionplatform.com.au`', the account number is '`324181581397`'.

Similarly, the email dispatcher allows the outputs from a process to be emailed to a list of recipient email addresses via the '`emails`' option. Note that all these recipients must be registered users of the Fusion Platform®. Any which are not registered users will be ignored. An optional message can be included in the email via the '`message`' option.

All dispatchers have a '`condition`' option which is checked prior to dispatch. Dispatch only takes place if the condition evaluates to true. The '`condition`' option is therefore a Python expression which has available variables which equate to the number of inputs to the dispatcher ('`inputs`'), the number of files in each input ('`input_n_files`', where '`n`' is the input number starting from 1) and the name ('`input_n_file_m_name`', where '`m`' is the file number starting from 1) and number of lines ('`input_n_file_m_lines`') in each file.

When the process has been correctly configured, it can be saved to the Fusion Platform®. During this creation, the process will be validated and its corresponding price per execution calculated. If the process fails validation, then an exception will be thrown, and the process will not be saved.

Python

```
# Attempt to create the process which will first validate it.
# This will throw an exception if the validation fails.
process.create()


# Once created, the price will be available to review.
print(process.price)
```

Important

Prior to execution, the name, options, inputs, schedule and dispatchers of a process may be changed using the ‘`update`’ method. However, once the process has been submitted for execution, no values can be changed. If the process is stopped, then the name and schedule only may be changed. To change any other values, use a copy of the process instead. To obtain a copy, use the ‘`copy`’ method, which will provide a template process with the same values as the original.

A template process, which is a copy of a process which has already been created or executed, can be obtained as follows. This will copy the options, inputs, schedule and dispatchers from the previous process, set the name, and then the new template process can be used to modify any of these values before being created.

Python

```
# Create a copy of a process which has previously been
# created or executed. This copies the name, options, inputs
# and schedule.
process_copy = process.copy(name='Copy of Example')


# Just like any other new template process, the values can be
# updated. Here we schedule the execution.
then = datetime.now(timezone.utc) + timedelta(days=1)
process_copy.update(
               run_type=fusion_platform.RUN_TYPE_RUN_SCHEDULE,
               repeat_count=1, repeat_start=then)


# The copied process may then be created.
process_copy.create()
```

When the process is no longer needed, it may be deleted:

Python

```
# Delete the process:
process.delete()
```

This will raise an exception in case of insufficient privileges, otherwise the process will be deleted. This will first stop any current executions and prevent any further executions taking place. Then this will delete any existing executions and their associated outputs, before deleting the process itself. This deletion cannot be undone.

8 Executing and Monitoring Processes

An iterator through the processes which have been validated and saved for an organisation can be obtained as follows:

Python
```python
# Select the organisation to find processes. This is
# the first organisation the user belongs to in this example:
organisation = next(user.organisations)


# Now iterate through the processes:
for process in organisation.processes:
    print(f"{process.name}: {process.process_status}")
```

Each entry is a process object, which contains all the details of a created process. Processes can also be searched for by name:

Python
```python
# Search for processes by name:
process, processes = organisation.find_processes(name='Process')
```

Searching by name looks for those processes which have the corresponding name. All searches are case sensitive. Names are not unique, hence the '`process`' return is the first item that has been found (or '`None`' if no items found) and the '`processes`' return is an iterator through all items found. A case-sensitive search is performed for processes whose name starts with the supplied value. A case-insensitive search on relevant object fields can be conducted using the '`search`' parameter instead.

A specific process can also be obtained from its process identifier, which is listed as part of its process information.

Python
```python
# Get the process by its identifier:
process, _ = organisation.find_processes(
                  id='6a4417de-e885-4749-9f75-f7eeb2aac7ab')
```

Section 11 describes how to get help on the content of the process information.

A process may be executed using the following:

Python
```python
# Execute the configured process. See later for how to
# wait for completion.
process.execute()
```

This will execute a process according to its options, inputs and schedule. Every execution will incur the price listed with the process. This method will return immediately and will not wait for the execution to complete.

Important

Once a process has been executed, it may not be modified. To modify the name or schedule, the process must first be stopped. To modify any other values, a copy can be created of the process.

To wait for the execution to start after calling ‘`execute`’, use the following:

Python

```
# Wait for the next execution to start.
process.wait_for_next_execution()
```

Waits for the next execution of the process to start, according to its schedule. If executions are already started, this method will return immediately. Otherwise, this method will block until the next scheduled execution has started.

Important

If the process is configured with a schedule, this method will wait until the scheduled time, and hence this method may block for a considerable time.

For every execution of a process, a corresponding process execution object is created, and these can be obtained as follows:

Python

```
# Get all the executions for the process:
for execution in process.executions:
    print(f"{execution.id}: {execution.progress})
```

The progress of an execution can be determined from its ‘`progress`’ attribute, which is its percentage progress through execution. When the ‘`progress`’ is 100%, then the execution is complete and the ‘`success`’ flag will state whether the execution was successful (‘`True`’) or failed (‘`False`’). When an execution completes, an optional notification will be sent by email and/or text message to the user account that executed the process. Whether such notifications are sent can be configured for the user account (see section 4 for how to modify account information).

For convenience, the following can be used to determine if the execution has finished, raising an exception if the execution was unsuccessful:

Python

```
import time

# Loop waiting for the execution to complete.
while not execution.check_complete():
    time.sleep(10)  # Sleep for 10 seconds.

# The execution has completed without exception.
print('Execution successfully completed')
```

To have the method wait for the execution to complete, turn the ‘`wait`’ flag on:

Python

```
# Wait for the execution to complete.
execution.check_complete(wait=True)

# The execution has completed without exception.
print('Execution successfully completed')
```

To execute and wait, the ‘`wait`’ flag can be turned on:

Python

```
# Execute the configured process and wait for completion:
process.execute(wait=True)
```

Important

The amount of time it takes to execute the process depends upon the selected service. When the ‘`wait`’ flag is turned on, the method will wait for the next scheduled execution to start, and then block until all current executions of the process have completed. If the process is configured to repeat the execution more than once, only the first execution will be considered.

For large regions of interest, the Fusion Platform® will automatically cut up the region into smaller slices to make processing more efficient, and each slice will be allocated to a separate execution within a group. Just the executions for a group can be obtained as follows:

Python

```
# Get all the executions for a group:
execution, executions = process.find_executions(
              group_id='1cf57d5c-f1c9-4cac-b6c7-076cc9c3ec89')
```

The ‘`execution`’ return is the first execution that has been found (or ‘`None`’ if no items found) and the ‘`executions`’ return is an iterator through all items found. When an execution is part of a group, only one completion notification will be sent, instead of one per execution in the group.

Some executions when grouped produce a combined output from all the executions. These aggregated results are obtained from an extra execution which is included in the group. For example, a ‘`Statistics Aggregator`’ may be added to a group to combine the statistic outputs from each execution in the group.

Every process is configured with an output storage period in days which determines how long each execution is kept before it is automatically deleted. Prior to deletion, an optional notification may be sent via email or text message to the user account which executed the process to warn that the execution will be deleted shortly.

If the automatic deletion should be delayed further, then this delete expiry can be modified for an execution as follows:

Python

```
# Change the delete expiry for an execution to delay it
# from being deleted for 2 days from now.
delete_expiry = datetime.now(timezone.utc) + timedelta(days=2)
execution.change_delete_expiry(delete_expiry=delete_expiry)
```

If the update fails for any reason, such as invalid parameter values or insufficient privileges, then an exception will be thrown. If the execution is part of a group, then this will also change the delete expiry for all executions in the same group. Extending the expiry will mean that additional storage costs are incurred for the extra time that the execution's outputs are stored. All storage charges are collated daily. Section 11 describes how to get help on the content of the execution information.

Process executions can be stopped at any time by using the following:

Python

```
# Stop all executions of a process, aborting those running:
process.stop()
```

This will first stop the process from running any more executions as per its schedule. It will then abort any executions which are in progress. Partial charges will still be incurred for any incomplete executions and the corresponding outputs which have been created.

Important

If the process is configured to run only once, then when it has been executed, the process will automatically be set to stopped. This enables the process to be run again using the same inputs and options. When stopped, the name and schedule for a process may be modified.

9 Obtaining Execution Inputs and Outputs

Once an execution has completed, then its inputs and outputs can be downloaded. Each execution is composed of the different operations that have been executed to make up the full processing chain of the service. For example, this will involve ingesting data and pre-processing it before the associated algorithm is run. Each of these components therefore corresponds to a separate, individual service executed as part of the chain. Each of these components can be obtained from an execution as follows:

Python

```python
# Select the execution for a process. This is the last
# execution which was started for the process in this example:
execution = next(process.executions)


# Now iterate through the components of the execution:
for component in execution.components:
    print(f"{component.name}: {component.runtime}")
```

If an execution is still executing, then it and its components can still be accessed using the above methods. However, the associated data will be incomplete. Section 11 describes how to get help on the content of the execution component information.

Each component may have inputs and outputs associated with it. Typically, in a processing chain, the first component in the chain will take as input the region of interest. Its outputs will then be fed as inputs to the second component in the chain, and so on. Whereas a typical first input is a region of interest defined as a geospatial vector file (see section 5), outputs from executions may be any supported file type. This includes geospatial vector files, such as GeoJSON, ESRI Shapefile, KML and KMZ and geospatial raster files, including DEM, GeoTIFF, JPEG2000. Non-geospatial images, such as PNG and JPEG files, as well as other types of file may also be output, including CSV files. Types of file unknown to the Fusion Platform® will be listed as ‘`Other`’.

All the inputs and outputs are data objects which are associated with one or more files each. (See section 5 for how to work with data objects.)

Each of the files associated with the input data objects can be downloaded as follows:

Python

```python
# Select the first input for a component in this example:
input = next(component.inputs)


# Download all files from the input to the local directory:
for file in input.files:
    print(f"Downloading {file.file_name} ({file.size} bytes)")
    file.download(path=file.file_name)
```

Similarly, the files from an output can also be downloaded:

Python

```python
# Select the first output for a component in this example:
output = next(component.output)

# Download all files from the output to the local directory:
for file in output.files:
    print(f"Downloading {file.file_name} ({file.size} bytes)")
    file.download(path=file.file_name)
```

Executions will be automatically deleted according to the output storage period defined for the associated process (see section 8). However, an execution can be deleted before this expiry as follows:

Python

```python
# Delete the execution:
execution.delete()
```

This will raise an exception in case of insufficient privileges or if the process is still executing, otherwise the execution and its associated outputs will be deleted. This deletion cannot be undone.

Important

The inputs and outputs of an execution are transient and are only available while the execution exists. If the execution is deleted, then its inputs and outputs are also deleted. To persist any input or output beyond an execution, it can be copied using the '`copy`' method for the corresponding data object. All copies are treated as if they have been uploaded and therefore exist until they are explicitly deleted.

10 Detailed Example

The following shows a detailed example of how to use the SDK to create a process, execute it and download the resulting data. This example assumes that a suitable user account has been created with an organisation which has sufficient credits to run the process, and that the SDK has been set up as described in section 2.3. To use a different user account, replace the email address ‘`me@org.com`’ and password ‘`MyPassword123!`’. Note that the ‘`pandas`’, ‘`rasterio`’ and ‘`matplotlib`’ packages will also need to be installed.

The example uses a pre-defined region of interest of the Lake District in the UK, and then performs cloud classification over the region for a specific date using Sentinel-2. Since the region is relatively large, its execution is split. The example shows how to obtain aggregated statistics and how the split results can be combined into a single GeoTIFF, as well as how to work with the metadata generated for each file.

Python

```python
from datetime import datetime, timezone
import rasterio
from rasterio.merge import merge
import matplotlib.pyplot as pyplot
import pandas


import fusion_platform


# Login with email address and password.
user = fusion_platform.login(email='me@org.com',
                             password='MyPassword123!')


# Select the organisation which will own the file. This is
# the first organisation the user belongs to in this example,
# and which is therefore assumed to have sufficient credits.
organisation = next(user.organisations)


# Create a data item for the Lake District region of interest:
lake_district = organisation.create_data(
        name='Lake District National Park',
        file_type=fusion_platform.FILE_TYPE_GEOJSON,
        files=[fusion_platform.EXAMPLE_LAKE_DISTRICT_FILE],
        wait=True)


# Find the cloud classification service using its name.
service, _ = organisation.find_services(
                 name='Demo: Sentinel-2 Cloud Classification')


# Create a template process from the service.
process = organisation.new_process(name='Example',
                                   service=service)
```

Python

```
# Configure the process to use Lake District National Park
# as the region of interest.
process.update(input_number=1, data=lake_district)


# Configure the service to work on historic data.
process.update(option_name='latest_date', value=False)
process.update(option_name='start_date',
               value=datetime(2021, 3, 27, tzinfo=timezone.utc))
process.update(option_name='end_date', value=None)


# Create the process, review its price, then execute it.
process.create()
print(f"Price: {process.price}")
process.execute(wait=True)


# Because the Lake District region of interest is relatively
# large, the process will be executed in a group, so find all of
# the executions in the group. These are assumed to be the most
# recently started group obtained from the last execution.
execution = next(process.executions)
_, executions = process.find_executions(
                                group_id=execution.group_id)


# Download the cloud classification output for each execution
# together with the aggregated statistics. We also keep the
# metadata for each raster file.
files = []
selectors = []
statistics_path = None
raster_paths = []


for i, execution in enumerate(executions):
    for component in execution.components:
        # Download the aggregated statistics and classification.
        if component.name == 'Statistics Aggregator':
            print(f"Downloading statistics from {component.id}")
            output = next(component.outputs)
            file = next(output.files)  # First file only.
            files.append(file)
            statistics_path = file.file_name
            file.download(path=statistics_path)
```

Python

```
        elif component.name == 'Sentinel-2 Cloud Classification':
            print(f"Downloading output from {component.id}")
            output = next(component.outputs)
            file = next(output.files)  # First file only.
            files.append(file)
            raster_paths.append(f"{i}_{file.file_name}")
            file.download(path=raster_paths[-1])

            # The file has a single band for the classification.
            selectors.append(file.selectors[0])

# Wait for all downloads to complete.
for file in files:
    file.download_complete(wait=True)

# Load the statistics into a Pandas data frame.
statistics = pandas.read_csv(statistics_path)
print(statistics)

# Create a single cloud classification image as a mosaic.
rasters = []

for raster_path in raster_paths:
    rasters.append(rasterio.open(raster_path))

mosaic, transform = merge(rasters)
mosaic_meta = rasters[0].meta.copy()
mosaic_meta.update({'height': mosaic.shape[1],
                    'width': mosaic.shape[2],
                    'transform': transform})

with rasterio.open('mosaic.tif', 'w', ** mosaic_meta) as merged:
    merged.write(mosaic)

# Display the mosaic image.
with rasterio.open('mosaic.tif', 'r') as source:
    pyplot.imshow(source.read(1, masked=True), cmap='coolwarm')
    pyplot.show()
```

Python

```
# Display the histogram.
histogram = [0, 0, 0, 0, 0, 0, 0, 0, 0, 0]

for metadata in selectors:
    histogram = [first + second for first, second in
                 zip(histogram, metadata['histogram'])]

# There are only 4 classes, while the histogram has 10 bins.
x = ['Ground', 'Shadow', 'Thick Cloud', 'Thin Cloud']
y = [histogram[0], histogram[3], histogram[6], histogram[9]]

pyplot.bar(x, y)
pyplot.show()

# Tidy everything up by deleting the process and the input file.
process.delete()
lake_district.delete()
```

11 SDK Object Attributes and Methods

The SDK consists of package functions which control the overall behaviour of the SDK, and model objects, which are created in response to requests to the Fusion Platform®. Key model objects include:

User	The current logged in user associated with the active session with the Fusion Platform®.
Organisation	An organisation that the user belongs to. Organisations own data and processes.
Credit	The credits assigned to an organisation.
Data	A data item which holds one or more files of a specific type.
File	A data object's file and its associated metadata.
Service	An available service which may be chosen to be configured as a process.
Process	Describes the specific inputs and options for a service which may be executed to produce output data objects.
Execution	A specific execution of a process.
Component	Each service is composed of individual components used to complete the desired processing. For example, the ingestion of data followed by its classification. Each execution is therefore made up of one or more execution components representing these stages of processing.
Log	The log output from the execution of a component.

Each model object has three elements:

Attributes	These represent the information associated with each object, such as a '`name`'. Attributes may also be used to obtain an iterator through child objects, such as '`executions`' for a process. All attributes are read-only.
Common Methods	Common methods include '`update`', which, where allowed, permit one or more of the object's attributes to be updated. For example, '`process.update(name="My process")`'. Other common methods include '`find_<object_type>`' and '`delete`', where relevant and permitted.
Specific Methods	Specific methods are only available to certain object types. For example, '`organisation.new_process(service=service)`', which allows a template process to be initialised for an organisation.

11.1 Attributes and Methods

Detailed information about the available attributes and methods for the package and its classes used to form model objects can be obtained from:

https://www.d-cat.co.uk/public/fusion_platform_python_sdk/

Similar information can also be obtained in Python using the following:

Python

```
import fusion_platform

# Get help on the package methods:
help(fusion_platform)
```

Help on individual classes can be obtained in a similar way:

Python

```
from fusion_platform.models.user import User

# Get help on the user class:
help(User)
```

12 Glossary of Terms

Amazon Resource Name (ARN)	A unique reference provided by AWS for each resource.
Amazon Web Services (AWS)	Cloud computing platform.
Application Programming Interface (API)	Typically refers to a software interface (rather than a user interface) used to access services or resources, performing actions or storing or retrieving data.
Comma Separated Values (CSV)	A text file format used to provide row-based data separated by commas.
Digital Elevation Model (DEM)	A type of three-dimensional model file for geolocated elevations.
Earth Observation (EO)	Gathering of physical characteristics of the Earth, typically via remote sensing.
ESRI Shapefile	A type of vector data file which allows attributes to be assigned to locations using points, lines or polygons.
GeoJSON	JSON data used to encode geospatial data.
Geospatial raster file	A file which is contains raster data which is geospatially located. GeoTIFF and JPEG2000 are both types of geospatial vector file.
Geospatial vector file	A file which is used to describe geographic areas or points of interest. GeoJSON and ESRI Shapefiles are both examples of geospatial vector files.
GeoTIFF	A type of image file which allows the embedding of geospatial metadata alongside image data (in TIFF format).
Hypertext Transfer Protocol (HTTP)	Communication protocol used to exchange data with a web server. In this document, HTTP is synonymous with the corresponding HTTP Secure version (HTTPS).
Joint Photographic Experts Group (JPEG)	A type of image file format which uses lossy data compression.
JPEG2000	A type of image file which allows the embedding of geospatial metadata alongside image data (in JPEG format).
JavaScript Object Notation (JSON)	Text-based format used to label data for data interchange.
Keyhole Mark-up Language (KML)	A text-based mark-up language used to encode geospatial data, with the zipped version being KMZ.
Keyhole Mark-up Language Zipped (KMZ)	A KML file and associated resources which has been zipped into a single file to reduce the file size.

Key Management Service (KMS)	Specifically for AWS this provides a service to manage and use encryption keys.
OpenAPI 3.0 Specification [3]	The standardised specification of a RESTful API interface which can be used to document or access the API with suitable software which can read the specification.
Portable Network Graphics (PNG)	A type of image file format which uses lossless data compression.
Process	Refers to process which has been created from a selected service, and which defines the required inputs and options for the process to execute.
Remote Sensing	Refers to the use of satellite (or aircraft or drone) technologies to remotely sense the Earth. A variety of sensing technologies can be used, including optical, hyperspectral or radar sensors.
Service	Services within the Fusion Platform® can be selected to execute a particular algorithm given appropriate inputs and options.
Simple Storage Service (S3)	The AWS cloud storage solution which provides buckets into which objects (files) can be stored.
Software Development Kit (SDK)	Term used to define software used by developers to perform a specific function. In this document, the SDK is used to interact with the Fusion Platform®.
Standardised Service Description (SSD)	Within the Fusion Platform® this defines the input, algorithm options and output requirements of a service which may be selected and configured as a process to be run. An SSD ID uniquely identifies a service which can be selected.
Uniform Resource Locator (URL)	A reference to a resource held on the web. A URL consists of a protocol (for example, “https://”) a resource name (“thefusionplatform.com”), a path (“/platform”) and parameters (“?id=1&name=User”).
Zip Archive	A single file which contains other files which have been compressed.

13 References

[1] QGIS, “Welcome to the QGIS project!,” 2022. [Online]. Available: https://www.qgis.org. [Accessed February 2022].

[2] Mapbox, “geojson.io,” 2022. [Online]. Available: https://geojson.io. [Accessed February 2022].

[3] Linux Foundation, “OpenAPI Specification v3.0.3 | Introduction, Definitions, & More,” 2020. [Online]. Available: https://spec.openapis.org/oas/v3.0.3.html. [Accessed March 2022].