Python - Workflow Interface

- Create Version from Blob Upload
- Workflow Definition
- Read a Product property
- Read parameters as DataFrame
- Search Time Series
- Get Time Series Data
- Create Report
- Create View

Create Version from Blob Upload

This example shows how to create a new version for a product by uploading files to a designated workflow blob drop-in:

```
import os
from azure.storage.blob import BlobServiceClient
from io import BufferedReader
from io import BytesIO
import json

CONTAINER_NAME = '!!!YOUR CONTAINER NAME HERE!!!'
connectin_string = "!!!YOUR CONNECTION STRING HERE!!!"

def uploadFileToBlob(filename, buffer):
    blob_service_client = BlobServiceClient.from_connection_string(connection_string)
    blob_client = blob_service_client.get_blob_client(container=CONTAINER_NAME, blob=filename)
    blob_client.upload_blob(buffer, overwrite=True)

files = [
    {

        "name": "file01.obj",

        "path": "W:\\mydata\\"

    },
    {

        "name": "file02.ftr",

        "path": "W:\\mydata\\"

    }
```

```
]

filesUpload = []
for f in files:
    with open(f["path"]+f["name"], "rb") as data:
        uploadFileToBlob(f["name"], data)
    filesUpload.append(f["name"])

manifest = {
    "user":  "me@gmail.com",
    "product": "MY_WORKFLOW_PRODUCT_ID",
    "comment": "A test version",
    "files": filesUpload
}

bytes=bytes(json.dumps(manifest), 'utf-8')
buffer = BytesIO(bytes)
uploadFileToBlob("manifest.json", buffer)
```

Workflow Definition

```
# This code runs inside the Voxel Farm platform

# Import the workflow lambda api
from voxelfarm import workflow_lambda

# Also import the voxelfarm client
from voxelfarm import voxelfarmclient

# Define the workflow callback functions
def receive_data(
        vf : voxelfarmclient.rest,
        request : workflow_lambda.request,
        host : workflow_lambda.workflow_lambda_host):
    host.log('Function called, doing nothing...')
    return {'success': True, 'complete': False, 'error_info': 'None'}

def stage_done(
        vf : voxelfarmclient.rest,
        request : workflow_lambda.request,
        host : workflow_lambda.workflow_lambda_host):
    host.log('Function called, doing nothing...')
    return {'success': True, 'complete': False, 'error_info': 'None'}

# The workflow hierarchy is defined as JSON structure
workflow = {
    'name' : 'Ore Body Model',
    'icon' : 'root',
    'id' : 'OBDP',
    'tracks' : [
        {
            'name' : 'Block Models',
            'icon' : 'voxbm',
            'id' : 'GM',
            'on_receive_data' : receive_data,
            'on_stage_done' : stage_done
```

```
},
{
   'name' : 'Drill Holes',
   'icon' : 'voxdh',
   'id' : 'DH',
   'tracks' : [
      {
         'name' : 'Blast Holes',
         'icon' : 'voxdh',
         'id' : 'DH_BLASTHOLES',
         'on_receive_data' : receive_data,
         'on_stage_done' : stage_done
      },
      {
         'name' : 'Hardness',
         'icon' : 'voxdh',
         'id' : 'DH_HARDNESS',
         'on_receive_data' : receive_data,
         'on_stage_done' : stage_done
      },
      {
         'name' : 'Ore Quality',
         'icon' : 'voxdh',
         'id' : 'DH_ORE_QUALITY',
         'on_receive_data' : receive_data,
         'on_stage_done' : stage_done
      },
      {
         'name' : 'Sondajes',
         'icon' : 'voxdh',
         'id' : 'DH_SONDAJES',
         'on_receive_data' : receive_data,
         'on_stage_done' : stage_done
      }

   ]
},
{
   'name' : 'Surveys',
   'icon' : 'voxsurf',
```

```
            'id' : 'SURF',
            'on_receive_data' : receive_data,
            'on_stage_done' : stage_done
        }
    ]
}

# Obtain the workflow_api interface
workflow_api = workflow_lambda.workflow_lambda_host()

# Set the workflow definition
workflow_api.set_workflow_definition(workflow)
```

Read a Product property

The following example reads the value of a property for a given product:

```
# Import voxel farm client and workflow lambda
from voxelfarm import voxelfarmclient
from voxelfarm import workflow_lambda

# Obtain the workflow_api interface
workflow_api = workflow_lambda.workflow_lambda_host()

# Read a property from another workflow product
value = workflow_api.get_product_property('MY_PRODUCT', 'my_property')
```

Read parameters as DataFrame

The following example loads a DataFrame for a provided parameter product in the workflow:

```
# Import voxel farm client and workflow lambda
from voxelfarm import voxelfarmclient
from voxelfarm import workflow_lambda

# Obtain the workflow_api interface
workflow_api = workflow_lambda.workflow_lambda_host()

# Read the product as a dataframe
df_stats = workflow_api.get_parameter_dataframe('META_COMPOSITE_STATS')
```

Search Time Series

The following code performs a search in the Time Series database:

```
data_frame = vf.search_time_series(
    spatial_id="Z602_404_812_408_ALH_5",
    object_id="CRUSHER3",
    property="DRY_TONNES",
    # An array of two ISO8601 dates
    timespan=["2017-01-17T10:00:11.68Z", "2022-04-03T13:45:11.68Z"],
    project_id="1B282E62D9574512AD6CDC88111EC31D",
)
```

Get Time Series Data

This example shows how to retrieve event data from Time Series for an array of time series identifier:

```
timeseries_ids = [
    {
        "spatialID": "B101_408_807_408_BAH_14",
        "objectID": "CRUSHER3",
        "property": "DRY_TONNES",
    },
    {
        "spatialID": "B101_408_805_408_BAH_8",
        "objectID": "CRUSHER3",
        "property": "WET_TONNES",
    },
]

data_frame = vf.get_time_series(
    timeseries_id=timeseries_ids,
    # An array of two ISO8601 dates.
    timespan=["2018-01-17T10:00:11.68Z", "2022-03-23T13:45:11.68Z"],
    project_id="1B282E62D9574512AD6CDC88111EC31D",
)
```

Create Report

The following example creates a report by running a lambda Report function:

```
result = voxelfarm_workflow.create_report(vf, request, 'pending_reports',
    'Survey Volume Difference', 'lambdas/surface-volume-difference.py', region, 4,
    {
        'previous_surface_id' :  f'{previous_terrain_id}',
        'current_surface_id' :  f'{voxel_terrain_id}',
        'non_mining_surface_id' : f'{active_hist_surf}',
        'nat_surface_id' : f'{active_nat_surf}',
    })
```

Create View

The following example shows how to create a view using the workflow interface:

```
voxelfarm_workflow.create_view(vf, request, 'Composite Difference', None, 'lambdas/view-terrain-composite-difference.py',
{
   'previous_surface_id' : f'{previous_terrain_id}',
   'current_surface_id' : f'{voxel_terrain_id}',
   'non_mining_surface_id' : f'{active_hist_surf}',
   'nat_surface_id' : f'{active_nat_surf}',
}, {})
```