

```

# Interface connections for incoming interface.crate objects
import json
from pathlib import Path

# Load the interface.crate
crate_path = Path("interface.crate/ro-crate-metadata.json")
with crate_path.open() as f:
    interface_crate = json.load(f)

# Index by @id for convenience
graph = interface_crate["@graph"]
by_id = {entry["@id"]: entry for entry in graph}

# A quick helper to create readable dates
from datetime import datetime

def parse_iso8601(dt_str):
    try:
        # Remove 'Z' if present and parse
        dt_str = dt_str.rstrip("Z")
        dt = datetime.fromisoformat(dt_str)
        # Format as "YYYY-MM-DD HH:MM:SS"
        return dt.strftime("%Y-%m-%d %H:%M:%S")
    except Exception:
        return dt_str # fallback to original if parsing fails

# Load metadata for E1: Data Producer Output
import xml.etree.ElementTree as ET
import re

e1_data = by_id.get("#E1-data-producer", {})
e1_files = e1_data.get("hasPart", [])
e1_file_ids = [f["@id"] for f in e1_files if "@id" in f]

# Representative image
e1_thumbnail_id = next((f for f in e1_file_ids if f.endswith("-ql.jpg")), None)
e1_thumbnail = f"interface.crate/{e1_thumbnail_id}" if e1_thumbnail_id else None

# Find the MTD_MSIL2A.xml file among e1_files
mtd_filename = "MTD_MSIL2A.xml"
mtd_id = next((fid for fid in e1_file_ids if fid.endswith(mtd_filename)), None)
mtd_path = Path("interface.crate") / mtd_id if mtd_id else None
e1_metadata = {}

def get_namespace(root):
    m = re.match(r"\{(.*)\}", root.tag)

```

```

        return {"n1": m.group(1)} if m else {}

def extract_child_texts(parent, ns):
    return {
        child.tag.replace(f"{{{ns['n1']}}}", ""): child.text
        for child in parent
        if child.text is not None and child.text.strip()
    }

if mtd_path.exists():
    tree = ET.parse(mtd_path)
    root = tree.getroot()
    ns = get_namespace(root)

    # Helper: Find tag and extract key:value pairs
    def extract_metadata(tag):
        elem = root.find(f".//{tag}", ns)
        return extract_child_texts(elem, ns) if elem is not None else {}

    e1_product_info = extract_metadata("Product_Info")
    e1_platform_info = extract_metadata("Datatake")
    e1_image_quality = extract_metadata("Image_Content_QI")

    # Round all float values in e1_image_quality to two decimal places (if possible)
    for k, v in e1_image_quality.items():
        try:
            e1_image_quality[k] = round(float(v), 2)
        except (ValueError, TypeError):
            pass

    e1_product_info_human = dict(e1_product_info)
    e1_platform_info_human = dict(e1_platform_info)

    for key in ["PRODUCT_START_TIME", "PRODUCT_STOP_TIME", "GENERATION_TIME"]:
        if key in e1_product_info_human:
            e1_product_info_human[key + "_HUMAN"] = parse_iso8601(e1_product_info_human[key])

    for key in ["DATATAKE_SENSING_START"]:
        for k in list(e1_platform_info_human.keys()):
            e1_platform_info_human[k + "_HUMAN"] = parse_iso8601(e1_platform_info_human[k])

    e1_metadata = {
        "product_info": e1_product_info_human,
        "platform_info": e1_platform_info_human,
        "image_quality": e1_image_quality
    }

```

```

    }

    # Load metadata for E2.1: Workflow Infrastructure
    e2_1_data = by_id.get("#E2.1-workflow-infrastructure", {})
    e2_1_parts = e2_1_data.get("hasPart", [])
    e2_1_dockerfile = next((f["@id"] for f in e2_1_parts if f["@id"] == "Dockerfile"), None)

    e2_1_dockerfile_content = None
    if e2_1_dockerfile:
        dockerfile_path = Path("interface.crate") / e2_1_dockerfile
        if dockerfile_path.exists():
            with dockerfile_path.open() as f:
                e2_1_dockerfile_content = f.read()

    e2_1_container_url = next((f["@id"] for f in e2_1_parts if "docker.com" in f["@id"]), None)

    # --- NEW: Load the provenance_output.crate ---
    # Get the path to the nested crate from the E2.2 entry
    e22_wms = by_id.get("#E2.2-wms", {})
    provenance_crate_path = e22_wms.get("hasPart", [{}])[0].get("@id", None)

    # Load the nested provenance crate if the path is found
    provenance_data = {}
    if provenance_crate_path:
        provenance_manifest = Path("interface.crate") / provenance_crate_path / "ro-crate-metadata"
        if provenance_manifest.exists():
            with provenance_manifest.open() as f:
                provenance_data = json.load(f)

    provenance_graph = provenance_data["@graph"]
    provenance_by_id = {entry["@id"]: entry for entry in provenance_graph}
    workflow = next((e for e in provenance_graph if e.get("@type") == ["File", "SoftwareSourceCode"]), None)
    steps = sorted([e for e in provenance_graph if e.get("@type") == "ControlAction"], key=lambda e: e.get("start"))

    FormalParameters = [e for e in provenance_graph if e.get("@type") == "FormalParameter"]

    step_summaries = []
    for step in steps:
        for e in provenance_graph:
            if e.get("@id") == step.get("object").get("@id"):
                create_action = e

        inputs = create_action.get("object")
        outputs = create_action.get("result")

        input_entities = []

```

```

output_entities = []

for e in provenance_graph:
    for input in inputs:
        if input.get("@id") == e.get("@id"):
            input_entities.append(e)

for e in provenance_graph:
    for output in outputs:
        if output.get("@id") == e.get("@id"):
            output_entities.append(e)

for e in provenance_graph:
    if create_action.get("instrument").get("@id") == e.get("@id"):
        softwareApplication = e

for e in provenance_graph:
    if create_action.get("containerImage").get("@id") == e.get("@id"):
        ContainerImage = e

# replace the startTime and endTime with human-readable format
create_action["startTime"] = parse_iso8601(create_action["startTime"])
create_action["endTime"] = parse_iso8601(create_action["endTime"])

step_summaries.append({
    "CreateAction": create_action,
    "SoftwareApplication": softwareApplication,
    "ContainerImage": ContainerImage,
    "Inputs": input_entities,
    "Outputs": output_entities,
})

# Extract E3 result info
e3_dataset = by_id.get("#E3-experimental-results", {})
zenodo_entry = e3_dataset.get("hasPart", [{}])[0].get("@id", None)

```

## Example LivePublication -- dynamic narratives that reflect experimental states

some body text here

### Computational Workflow

#### Parameters

```
{python} parameter["name"]
```

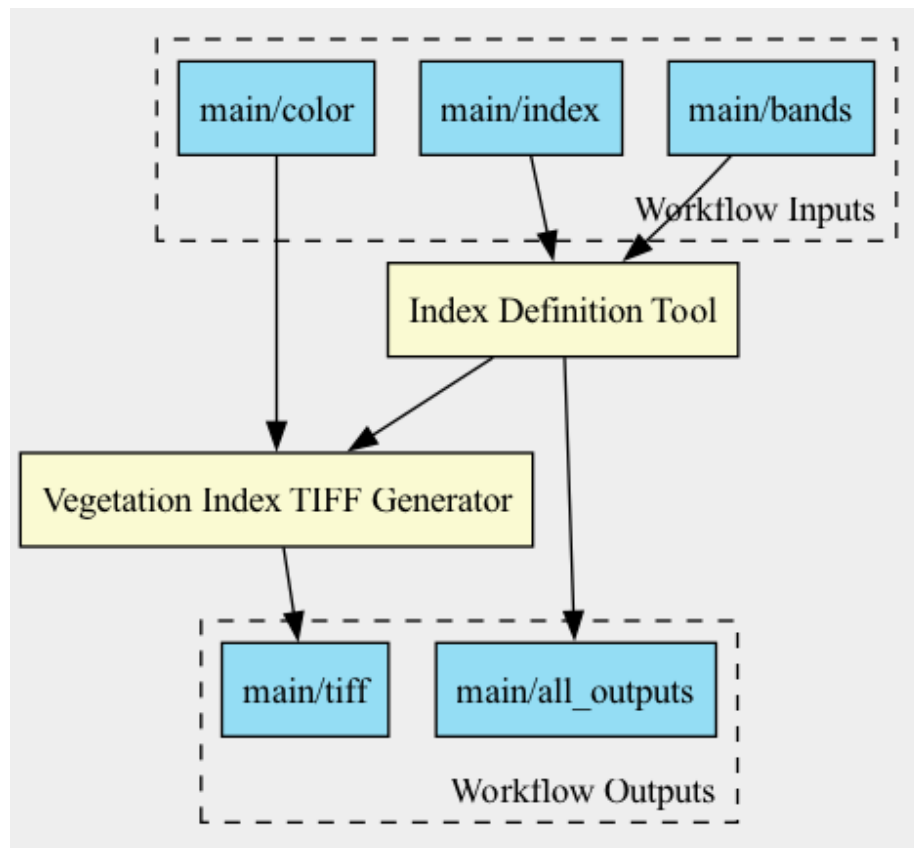


Figure 1: Workflow Preview

Description	Type
<code>{python} parameter["description"]</code>	<code>{python} parameter["additionalType"]</code>

## Steps

Step `{python} step["SoftwareApplication"]["name"]`

This step, `{python} step["CreateAction"]["name"]`, uses the tool *{python} step["SoftwareApplication"]["name"]*.

`{python} step["SoftwareApplication"]["description"]`

It was executed from `{python} step["CreateAction"]["startTime"]` to `{python} step["CreateAction"]["endTime"]`, using the container image `{python} step["ContainerImage"]["name"]`.

## Inputs

Name	Reference
<code>{python} input["name"] if "name" in input else input["@id"]</code>	<code>{python} ", ".join(e["@id"] if</code>

## Outputs

Name	Reference
<code>{python} output["name"] if "name" in output else output["@id"]</code>	<code>{python} ", ".join(e["@id"]</code>

## Sentinel-2A Data Product Overview

This publication uses a **Sentinel-2A** Level-2A product acquired during orbit 122 on 2015-09-29 10:03:58. The dataset, identified by this [DOI]([https://doi.org/10.5270/S2\\_-znk9xsj](https://doi.org/10.5270/S2_-znk9xsj)), was processed using baseline 05.00 (see here for information on baseline processing algorithms) on 2023-10-16 04:49:51.

## Data Alerts

```
# Create boolean flags for data Alerts
data_alert_flags = {
    "cloudy_pixels": float(e1_image_quality["CLOUDY_PIXEL_OVER_LAND_PERCENTAGE"]) > 50.0,
    "thin_cirrus": float(e1_image_quality["THIN_CIRRUS_PERCENTAGE"]) > 30.0,
    "saturated_pixels": float(e1_image_quality["SATURATED_DEFECTIVE_PIXEL_PERCENTAGE"]) > 0.0,
    "cloud_shadow": float(e1_image_quality["CLOUD_SHADOW_PERCENTAGE"]) > 10.0,
    "low_vegetation": float(e1_image_quality["VEGETATION_PERCENTAGE"]) < 5.0
```

```

}

# Ranked list of all flags
priority_order = [
    "cloudy_pixels",
    "thin_cirrus",
    "cloud_shadow",
    "saturated_pixels",
    "low_vegetation"
]

active_ranked_flags = [flag for flag in priority_order if data_alert_flags.get(flag)]

# Workaround: ensure it's at least 2 items so the loop will execute
if len(active_ranked_flags) == 1:
    active_ranked_flags.append("no_op")

# You can also define a narrative for "no_op" that renders nothing

```

The Sentinel-2A scene was assessed for conditions that may impact analysis reliability.

A large proportion of the land surface is cloud-covered (`{docsql} e1_image_quality["CLOUDY_PIXEL_OVER_LAND_PERCENTAGE"]%`), which may significantly distort GNDVI signals.

Thin cirrus clouds are present (`{docsql} e1_image_quality["THIN_CIRRUS_PERCENTAGE"]%`), potentially elevating NIR values and distorting vegetation estimates.

Cloud shadows affect part of the scene (`{docsql} e1_image_quality["CLOUD_SHADOW_PERCENTAGE"]%`), possibly reducing GNDVI by lowering NIR reflectance.

Saturation has been detected in `{docsql} e1_image_quality["SATURATED_DEFECTIVE_PIXEL_PERCENTAGE"]%` of pixels, indicating possible data corruption in bright areas.

Vegetation coverage is low (`{docsql} e1_image_quality["VEGETATION_PERCENTAGE"]%`), which can make GNDVI more sensitive to atmospheric noise or edge effects.

Analysts should carefully consider these conditions before using this dataset in quantitative workflows.

### Image Quality Summary

Property	Value
<b>Cloudy Pixels Over Land</b>	54.14%
<b>No Data Pixels</b>	60.53%
<b>Saturated/Defective Pixels</b>	0%

Property	Value
<b>Dark Features</b>	0.01%
<b>Cloud Shadow</b>	5.32%
<b>Vegetation</b>	26.68%
<b>Not Vegetated</b>	10.98%
<b>Water</b>	0.22%
<b>Unclassified</b>	2.73%
<b>Medium Probability Clouds</b>	14.68%
<b>High Probability Clouds</b>	39.25%
<b>Thin Cirrus</b>	0.13%
<b>Snow/Ice</b>	0%
<b>Radiative Transfer Accuracy</b>	0
<b>Water Vapour Retrieval Accuracy</b>	0
<b>AOT Retrieval Accuracy</b>	0
<b>AOT Retrieval Method</b>	SEN2COR_DDV
<b>Granule Mean AOT</b>	0.09
<b>Granule Mean Water Vapour</b>	1
<b>Ozone Source</b>	AUX_ECMWFT
<b>Ozone Value</b>	296.93