SensorThingsAPI for the EDS: a standardised service for real time Internet of Things (IoT)

A deliverable report for the AMPLIFY-EDS project

Date: 26 November 2025

Authors:

Alexandra Kokkinaki[1] https://orcid.org/0000-0001-8042-6391

Andrew Kingdon[2] https://orcid.org/0000-0003-4979-588X

Colm Walsh [1] https://orcid.org/0009-0009-0994-1003

James Clare [1] https://orcid.org/0009-0009-6112-2208

Gwenaëlle Moncoiffé [1] https://orcid.org/0000-0001-6559-4178

Louise Darroch [1] https://orcid.org/0000-0003-4163-9575

Affiliations:

1 - National Oceanography Centre, 2 - British Geological Survey

Contents

1. Context 3

2. Background 4

3. Approach 4

4. Project Outcomes 4

5. The Narrow Middle Architecture 5

6. Architecture and Technical Implementation 6

7. Sensor data streams across the EDS 8

8. MQTT Integration 9

9. The OGC Building Blocks 9

Building Blocks and Their Implementation 10

10. Investigation of an alternative to the FROST Server 11

11. Processing Steps vocabulary 11

Processing steps compilation 13

12. Interoperability Demonstrations 13

13. Recommendations and lessons learnt 15

14. Lessons learnt 16

1. Context

This report is designed for users of live environmental sensor streaming data, including organisations monitoring dynamic processes, modellers, and scientists requiring multi-disciplinary environmental data. The Environmental Data Service (EDS), funded by NERC, provides a focal point for all environmental science domains to ensure data is available, accessible, and reusable for the long term.

2. Background

The AMPLIFY-EDS project, funded by the UKRI DRI Phase II call, ran from March 2024 to October 2025. It aimed to co-design and develop a live-data prototype service delivering environmental sensor data and metadata through standardised workflows. This work builds upon previous initiatives like the ENHANCE and BOOST projects and the NERC Data Commons roadmap available here: https://nora.nerc.ac.uk/id/eprint/538375/.

3. Approach

The project utilised a commons-based approach, defining a community-managed space for asset discovery and re-use. Central to this was the "narrow middle" architectural pattern, which prioritises a minimal set of core services while allowing for technological innovation at the "edges" of data ingestion and visualisation.

4. Project Outcomes

- A narrow middle for sensor data commons
- Standardisation of near real-time sensor data publication using the OGC SensorThings API (STA).
- Enablement of interoperability between different sensor vendors and platforms.
- Demonstration of interoperability and usability through interactive tools like Leaflet web maps and Jupyter Notebooks.
- Development and publication of community profiles of SensorThings for the Environmental Data Service (EDS).
- Semantic enhancement of the SensorThings data model using NVS vocabularies and community-agreed standards.
- Contributed to the development of the OGC Building Blocks by testing the prototype system to semantically uplift SensorThings API json to json-LD.
- A draft vocabulary and approach for the capture of information about processing actions performed on sensor data, applicable across domains, discussed and agreed across EDS. This will lead to the creation of a cross domain “Processing steps” terminology that notifies the end user what kind of transformation and processing has been applied to the data.

- Investigation of an alternative sensorThingsAPI technical implementation (Hydroserver 2 Python STA) using data stored in a public cloud architecture (Amazon Web Services)

5. The Narrow Middle Architecture

The EDS Sensor Commons mandate only essential services to ensure interoperability, without constraining innovation. Only the minimum essential services and standards required for cross-system integration are mandated. All other technologies remain free to evolve at the edges and everything outside this core (storage, processing, analytics, user interfaces) is intentionally out of scope.

These are divided into four layers:

- Persistent Identification (PIDs): Using ORCID[1](people), ROR[2] (organisations) and DOIs (publications). Using BOOST-Instrument PID [3] where appropriate else select an instrument model PID from the NVS L22[4] vocabulary or an instrument category PID from the NVS L05[5] vocabulary
- Semantics & Meaning: Shared meaning is established through the use of:
- Controlled vocabularies and ontologies encoded in the Resource Description Framework (RDF) and SKOS and OWL respectively
- The I-ADOPT[6] framework for describing observable properties

More specifically for the sensor commons the following suggestions are made in terms of terminology:

Observable Properties: CF Standard Names – [NVS P077], the CF standard names vocabulary is widely used and already includes terms that are decomposed according to the I-ADOPT framework.

[1] https://info.orcid.org/what-is-orcid/

[2] https://ror.org/

[3] https://pid-sms.bodc.uk/

[4] http://vocab.nerc.ac.uk/collection/L22/current/

[5] http://vocab.nerc.ac.uk/collection/L05/current/

[6] https://i-adopt.github.io/

[7] http://vocab.nerc.ac.uk/collection/P07/current/

Units of Measurement: [NVS P06[8]]: BODC-approved data storage units

Sensor models: [NVS L22[9]]: SeaVoX Device Catalogue

Sensor categories: [NVS L05[10]]: SeaDataNet device categories

Platform Categories: [NVS L06[11]]: SeaVoX Platform Categories

Real-Time Data Model: Standardised on the OGC SensorThings API, following the EDS SensorThings Profile and streaming support (e.g. MQTT)[12] . Created SensorThings community profiles for EDS available under the SensorCommons GitHub repository: https://github.com/NERC-EDS/sensorCommons/tree/main/STAprofile

Discovery: Alignment with the European Science Cloud (EOSC) Interoperability Framework (IF), the Cross-Domain Interoperability Framework (CDIF) Schema.org and DCAT profiles like geoDCAT-AP, DCAT-FE etc. for cross domain and international discoverability.

6. Architecture and Technical Implementation

The system design was built around:

MQTT message brokers for real-time data streaming: MQTT was chosen as it has low overheads, is easy to setup and has very low bandwidth, thus making it ideal for setup on moving platforms such as ships.

EDS RabbitMQ broker: RabbitMQ was used to receive the MQTT messages and is then used as a broker for the Python app to connect to. RabbitMQ provides a useful user interface where the pace of messages can easily be monitored.

EDS Python data relay application: The Python app was developed to be the core of the application. Consuming messages from RabbitMQ, performing quality control, validation and then orchestrating the sending of entities to the FROST Server.

[8]http://vocab.nerc.ac.uk/collection/P06/current/

[9] http://vocab.nerc.ac.uk/collection/L22/current/

[10] http://vocab.nerc.ac.uk/collection/L05/current/

[11] http://vocab.nerc.ac.uk/collection/L06/current/

[12] https://mqtt.org/

FROST Server[13] **as the SensorThings API backend:** FROST implements an easy-to-use OGC SensorThings API which we can post to, and provide easy access to the data, for both our frontend system and stakeholders via a publicly accessible API.

A React based Frontend to present data: The simple React frontend site gives a simplified view of the data & metadata for each platform, sensor, location that the Python app is subscribed to, as well as an indication of the status of each stream.

OGC SensorThings v1.1 endpoints for structured and queryable data access shown in Figure 1. The OGC SensorThings API was chosen as the core standard for the Amplify project because it provides a mature, open, and widely adopted framework for managing and sharing Internet of Things (IoT) sensor data. It offers a flexible, modular data model that can represent both simple and complex environmental observations, while supporting real-time data streams and historical records within a single interface. Its RESTful and MQTT-compatible design makes it ideal for scalable, event-driven architectures.


```
{
  "value": [
    {
      "name": "Datastreams",
      "url": "https://linkedsystems.uk/sensorthings/v1.1/Datastreams"
    },
    {
      "name": "FeaturesOfInterest",
      "url": "https://linkedsystems.uk/sensorthings/v1.1/FeaturesOfInterest"
    },
    {
      "name": "HistoricalLocations",
      "url": "https://linkedsystems.uk/sensorthings/v1.1/HistoricalLocations"
    },
    {
      "name": "Locations",
      "url": "https://linkedsystems.uk/sensorthings/v1.1/Locations"
    },
    {
      "name": "Observations",
      "url": "https://linkedsystems.uk/sensorthings/v1.1/Observations"
    },
    {
      "name": "ObservedProperties",
      "url": "https://linkedsystems.uk/sensorthings/v1.1/ObservedProperties"
    },
    {
      "name": "Sensors",
      "url": "https://linkedsystems.uk/sensorthings/v1.1/Sensors"
    },
    {
      "name": "Things",
      "url": "https://linkedsystems.uk/sensorthings/v1.1/Things"
    }
  ],
```


Figure 1: The EDS SensorThings API implementation

[13] https://github.com/FraunhoferIOSB/FROST-Server

A bespoke Python data relay application was built to manage the connection to the MQTT broker, receive the data, perform basic checks and then apply quality control to data coming from a moving platform. The app was then responsible for publishing prepared data to the FROST Server.

The implementation was streamlined through the use of the FROST Server, an open-source, production-ready reference implementation that supports rapid deployment and customisation. By adopting SensorThings, we aligned with an internationally recognised OGC standard, ensuring interoperability with other systems and initiatives worldwide. Its strong compatibility with json and straightforward data structure made integration with existing tools, such as Leaflet and Jupyter Notebooks, efficient and user-friendly. Overall, SensorThings provided the right balance of standardisation, flexibility, and openness, making it particularly well-suited for federated environmental data streams across the EDS network.

A frontend application was also developed (Figure 2) which provided a view to select the available platforms via a dropdown menu, and would then display observations, locations, and relevant metadata about the selected stream. The purpose of the frontend was to supply a user-friendly and easily accessible method to view the data and the relevant metadata.

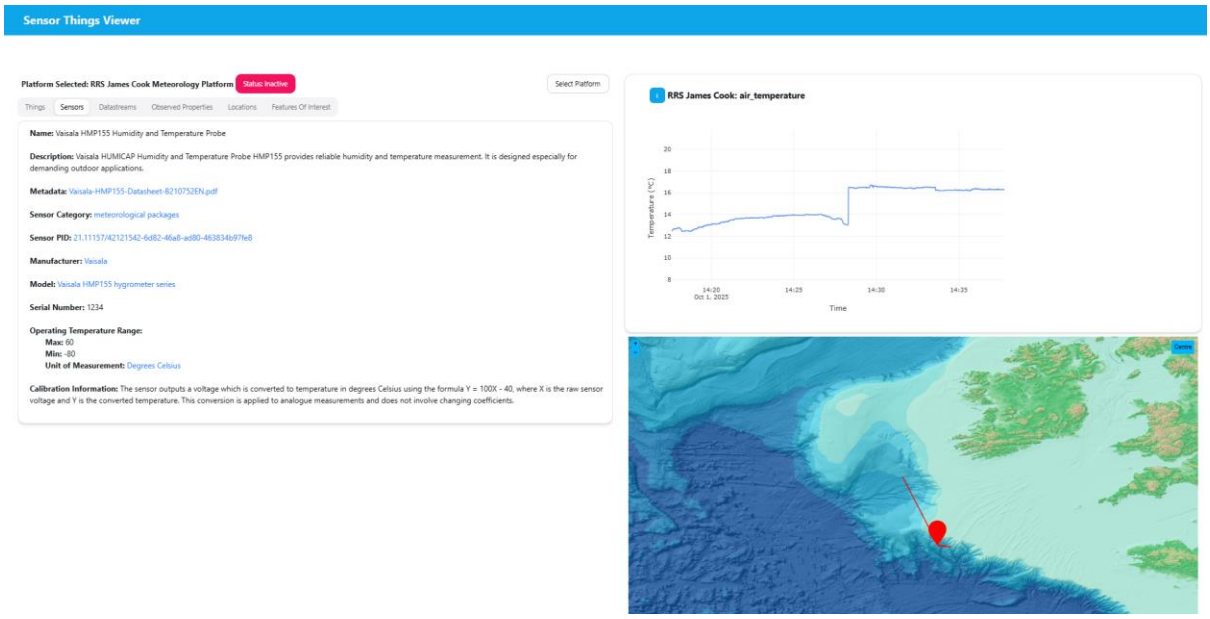

Figure 2. EDS SensorThings API User Interface

7. Sensor data streams across the EDS

The project integrated four distinct sensor streams, each contributed by different EDS centres, and exposed them through a unified SensorThings API endpoint. To ensure consistency and comparability across the datasets, all centres agreed to focus on a common observed property—atmospheric temperature—which was published by each centre through its respective stream.

BAS stream: UK Polar Data Centre established a Virtual Machine (rocky9) to host NOC-developed sda-mock-generator application. Mock messages include: $GPGGA, $SFMET. MQTT Mosquito brokering accessed via host: damp-link-testing.nerc-bas.ac.uk port 1883 (designed specifically for rbq2.ceda.ac.uk). The application is containerised and orchestrated using Apptainer (formerly known as Singularity), ensuring lightweight, secure and stable deployment.

NMF stream: NMF have established MQTT streams that transmit near real time data from the RRS James Cook and RRS Discovery. These are received by a shoreside virtual machine at based at NOC Southampton. The MQTT messages include multiple topics including GPGGA, NDMET and SFMET and are accessible via host: shipsystems.noc.soton.ac.uk and port 1883.

NCAS stream: an MQTT stream sending data from a Davis Vantage Pro2 weather station was set up from the AMOF yard (in Leeds - 53.804732 N, -1.561933 E).

BGS stream: contributed with an existing sensorThingsAPI implementation of groundwater loggers measuring temperature, conductivity and pressure, used as an exemplar of the appropriateness of this technology to sensor data export.

8. MQTT Integration

MQTT (Message Queuing Telemetry Transport) served as the near real-time messaging backbone of the Amplify project, enabling efficient, event-driven data exchange between sensor networks and the SensorThings API. The protocol's publish–subscribe model aligns naturally with the SensorThings data model, where MQTT topics correspond to entities such as Things, Datastreams, and Observations. Each sensor or platform publishes observations to a designated MQTT topic, which the FROST Server then ingests and translates into SensorThings-compliant resources. This tight integration ensures that new observations are instantly reflected in the API without manual intervention.

The use of the EDS-managed RabbitMQ broker coupled with the EDS Python data relay application provided both reliability and scalability, supporting multiple data centres streaming observations concurrently. MQTT's lightweight design minimised bandwidth consumption, making it ideal for distributed IoT deployments and moving platforms. Overall, the combination of MQTT and SensorThings delivered a robust and interoperable framework for real-time environmental data exchange.

9. The OGC Building Blocks

At BODC, we have implemented the conversion of a plain json OGC sensorThingsAPI into json - LD working together with OGC colleagues and using the OGC Building Blocks framework. This modular approach enables technology-agnostic relationships, supporting FAIR principles and improving semantic interoperability. By publishing these Building Blocks in the OGC GitHub repository, we make it easy for other standards and applications, such as SensorThings API, to reuse and integrate this work. This achievement represents a significant step towards interoperability, demonstrating how OGC standards can be more discoverable, interoperable, and ready for the semantic web.

Building Blocks and Their Implementation

A Building Block is a modular component of a specification that can be packaged for reuse across other specifications. This approach directly supports interoperability and adherence to FAIR principles, as outlined in the OGC Design Principles.

Historically, many specifications have described their relationships with others through textual statements or by providing implementation artefacts such as schemas—often without making these dependencies or relationships explicit. The OGC Building Blocks framework addresses this by enabling technology-agnostic dependency and relationship management, effectively forming the data management foundation of OGC's knowledge graph (OGC RAINBOW).

To ensure these Building Blocks are discoverable and reusable by other standards and applications, their definitions should be published in the OGC Register once identified. Normative information about OGC Standards Building Blocks is available in the Technical Committee Policies and Procedures.

At BODC, we collaborated with OGC to apply this framework and publish a set of Building Blocks that enable conversion from plain json to json-LD at the entity level. This implementation demonstrates that a standard such as SensorThings API can be instantly expressed as json-LD, without breaking its interoperability with other sensorThings implementations, and significantly improving its semantic interoperability. The resulting Building Blocks have been published in the OGC GitHub repository for use by the SensorThings API community, and further collaboration is underway to advance this work.

The first part of this work was to establish mappings/crosswalks between the existing sensorThings properties e.g. “name” per entity type e.g. Things using the @context jsonLD element. The @context is a set of rules for interpreting a json-LD document as described in The Context section of json-LD 1.1, and normatively specified in the Context Definitions section of json-LD 1.1. An example of the @context for the entity type Things is this: https://british-oceanographic-data-centre.github.io/bblocks-sta/build/annotated/api/sta/Thing/context.jsonld. The second part of the work was to evaluate the creation of a building block following OGCs instructions and publish the “context” in order to be reused by others.

10. Investigation of an alternative to the FROST Server

While FROST offers a relatively simple, open-source implementation for the SensorThings API, it does have some limitations. Principally, it assumes that data is stored in a PostgreSQL database with its own table layout and offers no support for alternative data storage solutions.

This means that for some programmes to realise a SensorThings data stream, they would need to maintain and manage a live copy of all data in a separate database. This may be inefficient and impractical for many on cost and performance grounds.

Therefore, we initiated an investigation into an alternative implementation of SensorThings that utilised Python as a shim over existing metadata/data APIs using data from the UKCEH-led Floods and Drought Research Infrastructure (FDRI) programme.

A full report into the investigation of this alternative SensorThings implementation is available at https://nora.nerc.ac.uk/id/eprint/540576/

11. Processing Steps vocabulary

To ensure users understand the type and status of the data they receive, a draft vocabulary for data processing pipeline applicable at the datastream level, was developed. A Miro board was created to explicitly compile transformations applied to the data. Existing frameworks, such as NASA’s Processing Levels (L0–L5/6) pioneered in Earth Observation, have been adapted by different communities to indicate the level of processing. After compiling known community-specific processing levels and holding several discussions with EDS partners, it was concluded that none of the existing vocabularies fully met Amplify’s needs. Starting with the terms compiled by the various partners on the Miro board (Figure 3), a draft controlled vocabulary was developed to describe processing steps and actions independently of when these actions occur in the data lifetime. A diagram was then produced to visualise the relationships between actions (Figure 4) and guide the creation of an optimum list of terms identifying processing steps.

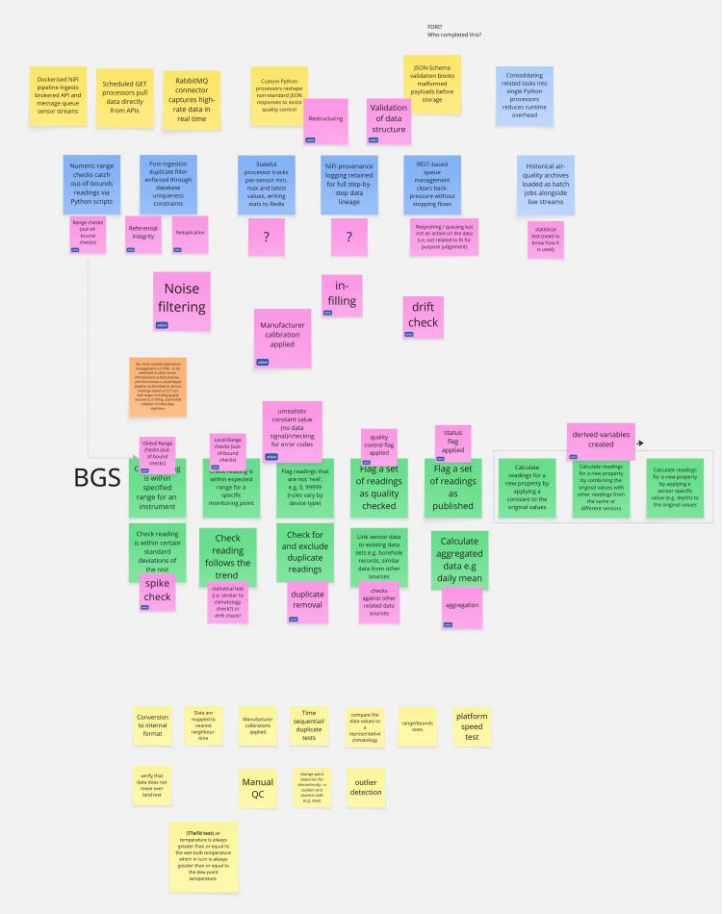

Figure 3: Miro board showing sensor data actions

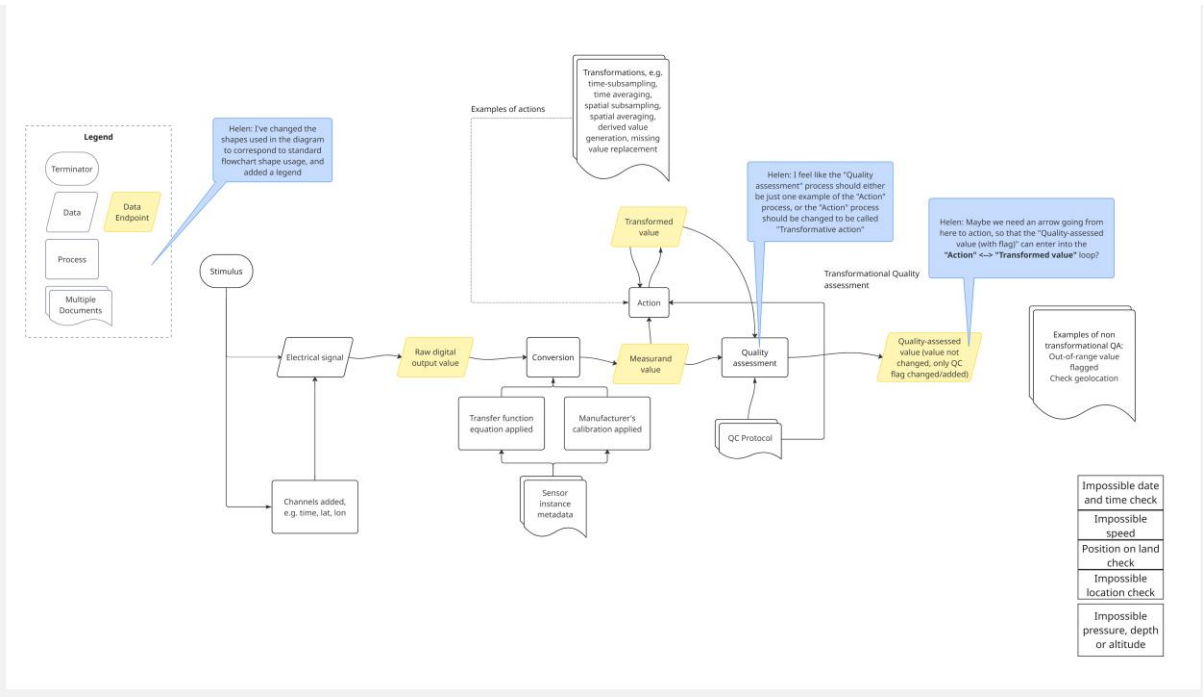

Figure 4: Sensor data diagram helping understand and collect processing steps

Processing steps are manual or automated interventions ("actions") on data and they differ from "Processing Levels" which are statements related to the degree of processing applied to a dataset or product. Below is a preliminary list of possible terms to be added to a new vocabulary for "Processing Steps": manual or automated interventions ("actions") on data.

Processing steps compilation	
Manufacturer's calibration applied	QC flag applied
Data cleaning, "denoising"	Drift check
Check impossible geospatial coordinates	Drift correction applied
Check impossible date/time	Restructuring
Outlier detection	Validation of data structure
Field calibration applied	Referential integrity check
Reference calibration applied	Deduplication
Addition of derived variables	In-filling
Conversion to internal format	Noise filtering
Spike check	Status flag added
Global range check (global out-of-bound check)	Unrealistic value check
Local range check (local out-of-bound check)	Aggregated value added

12. Interoperability Demonstrations

Two user-facing demonstrations showcased the practical interoperability of the sensor commons and the narrow middle:

- Leaflet Web Map Interface – Visualises federated sensor streams across multiple endpoints as depicted in Figure 5. Available at: https://britishgeologicalsurvey.github.io/sensor-things-api-demo/web-maps/federated.html
- Jupyter Notebook – Demonstrates live querying, analysis, and visualisation of SensorThings data. A snapshot of the notebook is shown in Figure 6. Notebooks are available at: https://github.com/NERC-EDS/sensorCommons/blob/notebooks/Notebooks/sensorthings%20plot.ipynb https://github.com/NERC-EDS/sensorCommons/blob/notebooks/Notebooks/Query%20Frost%20STA.ipynb

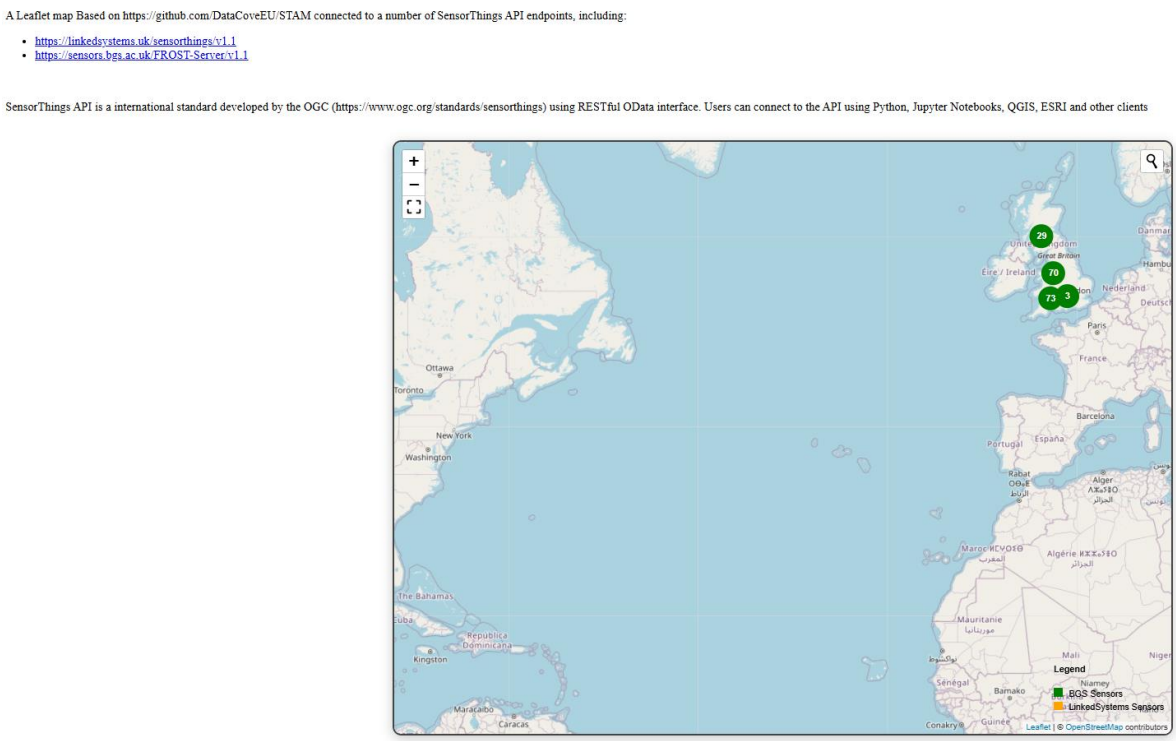

Figure 5: A Leaflet map connected to the EDS sensorThingsAPI endpoints of EDS and BGS

Basic plot

In [3]:

```
import matplotlib.pyplot as plt
import matplotlib.dates as mdates

# covnert str datetimes to datetime
df['phenomenonTime'] = pd.to_datetime(df['phenomenonTime'], errors='coerce')

fig, ax = plt.subplots()
ax.plot(df.phenomenonTime, df.result, marker='', linestyle='-')


ax.xaxis.set_major_locator(mdates.AutoDateLocator())
ax.xaxis.set_major_formatter(mdates.DateFormatter('%Y-%m-%d'))

plt.xticks(rotation=45, ha='right')

# Adjust layout to prevent clipping
plt.tight_layout()

plt.xlabel('phenomenonTime')
plt.ylabel('Air Temperature C')
plt.title('Time series of Air Temperature')

plt.show()
```

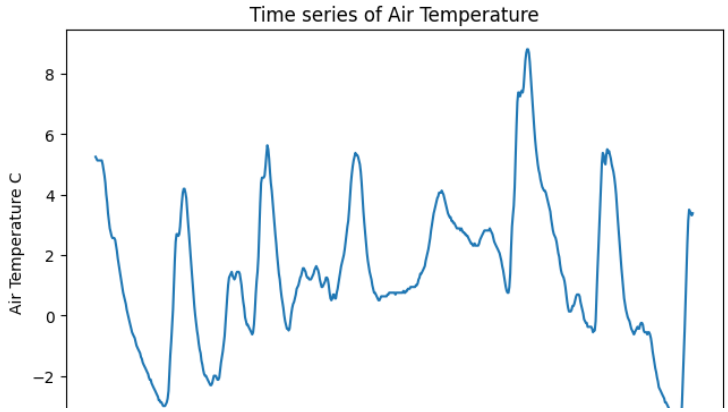

Figure 6: Output from the developed jupyter notebook.

13. Recommendations and lessons learnt

- Adopt the Narrow/Thin Middle Architecture
- Use globally resolvable identifiers:
- ORCID for people
- ROR for organisations
- Create Persistent Identifiers for sensors via BOOST PID service (fallback to NVS L22/L05).
- DOI for data/publications
- NVS/EnvThes URIs for semantic concepts
- Annotate your data and metadata with controlled vocabularies and ontologies (SKOS, OWL).
- Recommended vocabularies:
    - CF Standard Names (NVS P07) for observable properties
    - NVS P06 for units of measurement
    - NVS L22 for sensor models
    - NVS L05 for sensor categories

    - NVS L06 for platform categories
- Adopt I-ADOPT framework for observable properties.
- Standardise on OGC SensorThings API
- Develop and publish community profiles to enforce domain-specific constraints.
- Reuse OGC Building Blocks where possible for semantic uplift of the standards
- Ensure datasets/services are findable via:
- Schema.org / FE-DCAT-AP metadata
- EOSC-compatible harvesting
- Align with Cross-Domain Interoperability Framework (CDIF) recommendations.
- Use any technology that will support your implementation using the specified standards (FROST is not the only one)

14. Lessons learnt

Implementing STA was relatively straightforward. The main challenges arose when dealing with the nuances of the FROST server, and the relationship between the entities which was often undocumented. Necessitating considerable investigation through trial and error. When uplifting our STA implementation from json to json-LD, key challenges included harmonising metadata across stakeholders, validating context files, and adapting FROST to support '@context' properties. We addressed these by defining a metadata model, performing individual file validation using the Building Blocks and json-LD Playground, and engaging with the FROST community to enable context posting into FROST.

While the OGC Building Blocks framework is effective for publishing, its bulk validation produced ambiguous errors compared to more targeted individual entity validation. Additionally, its validator was less strict compared to other json-LD validators. Consequently, all context files required additional verification using json-LD Playground before being republished through the Building Blocks framework.

Future OGC STA implementations would benefit from clearer guidance on json-LD integration with FROST and more robust validation would greatly boost the OGC Building Blocks framework. Other key lessons learned include the importance of early metadata alignment, precise context mapping, and validating each context file independently to ensure semantic accuracy